THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ENGINEERING SCIENCE AND MECHANICS

A MATHEMATICAL MODEL FOR BRAIN TISSUE

BRADFORD JOSEPH LAPSANSKY
SPRING 2014

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Engineering Science
with honors in Engineering Science

Reviewed and approved* by the following:

Corina S. Drapaca
Assistant Professor, Department of Engineering Science and Mechanics
Thesis Supervisor/Honors Advisor

Francesco Costanzo
Professor, Department of Engineering Science and Mechanics
Faculty Reader

Judith A. Todd
P. B. Breneman Department Head Chair
Professor, Department of Engineering Science and Mechanics

*Signatures are on file in the Schreyer Honors College and The Engineering Science and Mechanics Office

Abstract

Brain tissue is very sensitive to both mechanical forces and chemical imbalances. These imbalances can cause functional and/or structural changes of the tissue which can lead to the onset and evolution of neurological diseases. Accurate mathematical models of brain chemo-biomechanics that increase our understanding of both healthy tissue and disease mechanisms in the brain greatly aid the development of better diagnostic and therapeutic tools and protocols. This thesis models the brain as a mixture material made of three phases: solid, fluid, and ionic. The equations that govern the chemo-biomechanics of the brain are linearized and considered in a limiting one-dimensional case so that the accuracy of numerical solutions developed for these equations may be verified by using an analytic solutions represented as Fourier series. The model is then coupled to the classic Hodgkin-Huxley equations to predict the displacement field of neurons as a result of an applied electric potential.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First, I would like to greatly thank Dr. Corina Drapaca for her guidance, patience, and knowledge in helping me complete my research. In addition, I would like to thank Dr. Francesco Costanzo for his assistance on the mathematics involved and Dr. Patrick Drew for his assistance on the biology involved in my research. Without their help, this thesis would not be as well developed as it is.

I would also like to thank Dr. Thomas Winter of Penn State Wilkes-Barre for his guidance and support along with some of the code used to create the pictures present in this thesis. I would also like to thank Dr. Joseph Jumpeter of Penn State Wilkes-Barre for his guidance and support over the last four years. I would also like to thank Dr. Judith Todd, Ms. Melissa Fink, Ms. Emily Gallagher, and the rest of the faculty and staff in the Engineering Science and Mechanics department for their help and assistance with both this thesis and my education over the last two years. Lastly, I would like to thank my parents and family for their support they have given me throughout college and life.

Chapter 1
Introduction

Brain is a soft biological tissue that is electrically active (Kandel et al., 2012). Continuum models for soft biological tissues have been developed for charged porous media by Lai et al. (1991); Malakpoor et al. (2006); Sun et al. (1999), and they have been successfully used to simulate the behavior of cartilage (Gu et al., 1998; Lai et al., 1991). Mathematical modeling is a tool that can be useful for making predictions about an observed system as well as providing guidance for better experiments. The models by Gu et al. (1999); Lai et al. (1991); Sun et al. (1999) are attractive for use in brain research because, like cartilage, the microstructure of the brain can be modeled as a charged porous medium (Drapaca and Fritz, 2012; Elkin et al., 2010). This type of continuum model that is called triphasic, since the brain tissue is modeled as a mixture of three phases: solid (cell membranes of the neuron and glial cells), fluid (cerebrospinal fluid (CSF) and blood), and ionic (charged particles that flow between the intracellular and extracellular space of the tissue) (Bowen, 1976; Kandel et al., 2012; Sun et al., 1999). Such a model has the potential to predict the brain tissue response to traumatic brain injury, tumor growth, and neurodegenerative diseases. More specifically, the model in question can provides insight into how mechanical loading affects brain response as well as how chemical imbalances change the brain's mechanical response. These insights can lead to improved diagnostic and treatment protocols. For instance, the model can be used to study cortical spreading depression (CSD), a condition associated with a noticeable decline in local random electrical activity caused by an electrical or mechanical stimulation (Leao, 1944). It has been observed by Grafstein (1956); Obrenovitch and Zilkha (1995) that potassium (K^+) concentrations in the cortex influence the spread of CSD. Since the model takes potassium and other ions into

account, it would be suited for studying this phenomenon. The study of the mechanics of brain tissue in response to changing chemical concentrations could lead to an improved understanding of this disease.

Drapaca and Fritz (2012); Elkin et al. (2010) have successfully modeled chemo-mechanical interactions in the brain. The development of our model follows the one by Gu et al. (1998, 1999) since it takes into account an arbitrary number of species in the ionic phase. The ions that are active in the processes of brain tissue are potassium, sodium, calcium, and chlorine (Kandel et al., 2012) and any accurate model of brain should be able to account for all four. For simplicity, the model developed in this thesis only accounts for potassium, sodium, and chlorine since we are not modeling the synapses of neurons. At the synapse of a neuron, calcium plays a significant role in neurotransmitter release (Kandel et al., 2012). The equations for a triphasic model with a general number of ions in the ionic phase is presented in chapter 2 so that it can be adapted for any possible scenario.

The equations presented in chapter 2 need to be utilized to solve any boundary-value problem pertaining to different situations in the brain. Therefore a numerical solver needs to be developed since it is not always possible to develop an analytical solution to a set of boundary conditions. A one-dimensional numerical solution to the linearized triphasic model by Lu et al. (2010), which has already been used to model brain mechanics by Drapaca and Fritz (2012), has been developed since the numerical solution can be compared to one possible analytical solution for accuracy. While comparing the numerical solution to an analytical solution is not the only way to verify the accuracy of the numerical solver, it is the way that was chosen out of convenience. The results of a one-dimensional numerical solution obtained via the finite-difference method are verified in chapter 3 against one possible analytical solution.

In chapter 4, the equations of the Hodgkin-Huxley model (Hodgkin and Huxley, 1952a,b) are presented so that the cell membrane potential (voltage) of an axon could be calculated. A depolarization of an axon causes a change in the axon's membrane potential which leads to a flow of ions through the membrane (Kandel et al., 2012). In order to model the flow of ions through the membrane, along with the displacement of the membrane itself, another set of one-dimensional governing equations are derived to account for both the motion of ions inside the neuron and the neuron's membrane displacement.

Using the derived governing equations presented in chapter 5, the concentration of all ions along with the membrane displacement were calculated for a set of specific boundary and initial conditions. The values of various parameters were chosen from Kandel et al. (2012); Lide (2007); Medvedev (2005); Weiss (1996) in order to best model a normal, healthy neuron. The behaviors of the membrane displacement and ionic concentrations can be determined in response to the applied (cell) membrane potential. The results are shown in chapter 6. The mechanical behavior of a neuron's membrane, which will be assumed to be the solid phase of the model mixture, due to an applied electrical stimulus has been investigated through computer simulations via a developed numerical solver in MATLAB, assuming the intracellular space and membrane of a neuron to be a triphasic mixture. The proposed model has not been validated experimentally yet. The results presented in chapter 6 do not actually model natural phenomena in brain. However, the results should show "proof-of-concept" that the triphasic model can be adapted in the future for use in modeling brain phenomena.

Chapter 2
Brief Review of the Triphasic Mixture Theory

2.1 Balance of Mass

Following Gu et al. (1998), the volume fraction of each of the three phases that constitute a triphasic mixture will be denoted as follows: ϕ^β for $\beta = s,\ w,\ I$ where $s,\ w,\ I$ represent the solid, fluid, and ionic phase, respectively. In particular the volume fraction of the ionic phase is given by $\phi^I = \sum_{\alpha=1}^{n} \phi^\alpha$ where ϕ^α is the volume fraction of each ion species. The saturation condition states that the sum of all of the volume fractions for all phases in the mixture is equal to one (Gu et al., 1998; Sun et al., 1999):

$$\phi^s + \phi^w + \sum_{\alpha=1}^{n} \phi^\alpha = 1. \tag{2.1}$$

The volume fraction of each constituent of the mixture relates the bulk density, ρ^β of that constituent to the constituent's respective true density, ρ^β_{T} (Sun et al., 1999). The true densities of the solid and fluid phases can be thought of as: $\rho^s_{\mathrm{T}} = m_s/V_s$ for the solid phase and $\rho^w_{\mathrm{T}} = m_w/V_w$ for the fluid phase where $m_{s,w}$ and $V_{s,w}$ are respectively the mass and volume of the solid and fluid phases (Bowen, 2010, 1980). The true densities of the ionic species are related to the molar concentrations and the molecular weights of each ion in the mixture (Sun et al., 1999). Following Sun et al. (1999):

$$\rho^s = \phi^s \rho^s_{\mathrm{T}}, \tag{2.2}$$

$$\rho^w = \phi^w \rho^w_{\mathrm{T}}, \tag{2.3}$$

$$\rho^\alpha = \phi^\alpha \rho^\alpha_{\mathrm{T}} = \phi^w c_\alpha M^\alpha, \tag{2.4}$$

for $\alpha = 1,\ 2,\ \ldots,\ n$. The volume fractions of the ionic species are negligible compared to the volume fractions of the solid and fluid phases $\left(\phi^I \ll 1\right)$, so the saturation condition can be rewritten as (Gu et al., 1998; Sun et al., 1999):

$$\phi^s + \phi^w \cong 1. \tag{2.5}$$

According to Gu et al. (1999); Sun et al. (1999), the local form of the balance of mass for each species in the mixture when chemical reactions are neglected is:

$$\frac{\partial \rho^\beta}{\partial t} + \operatorname{div}(\rho^\beta \boldsymbol{v}^{\boldsymbol{\beta}}) = 0; \quad \beta = s,\ w,\ 1,\ 2,\ \ldots,\ n. \tag{2.6}$$

In equation (2.6), $\boldsymbol{v}^{\boldsymbol{\beta}}$ is the velocity of the β^{th} phase (Bowen, 2010, 1976, 1980). Using the relations in equations (2.2) – (2.4) and the assumption that the mixture is incompressible yields:

$$\frac{\partial \phi^s}{\partial t} + \operatorname{div}(\phi^s \boldsymbol{v}^{\boldsymbol{s}}) = 0, \tag{2.7}$$

$$\frac{\partial \phi^w}{\partial t} + \operatorname{div}(\phi^w \boldsymbol{v}^{\boldsymbol{w}}) = 0, \tag{2.8}$$

for the solid and fluid phases of the mixture (Gu et al., 1999). Since the concentrations of each ion can change, the balance of mass of the ions will take on a different form (Sun et al., 1999):

$$\frac{\partial \left(\phi^w c_\alpha\right)}{\partial t} + \operatorname{div}(\phi^w c_\alpha \boldsymbol{v}^{\boldsymbol{\alpha}}) = 0;\ \alpha = 1,\ 2,\ \ldots,\ n. \tag{2.9}$$

2.2 Electroneutrality Condition

The solid phase of the mixture has an electric charge which is measured by a quantity called the fixed charge density (FCD) denoted by c^F (Gu et al., 1998; Sun

et al., 1999). Gu et al. (1999) defines c^F as the "equivalent moles of mono-valent ions per unit of water volume in the mixture." The electroneutrality condition, which states that there is a zero net charge at all material points in the mixture, is defined by Gu et al. (1999) as:

$$\sum_{\alpha=1}^{n} z_\alpha c_\alpha + \omega c^F = 0. \tag{2.10}$$

In equation (2.10), z_α is the valence of the α^{th} species in the mixture and the quantity ω denotes the valence of the FCD (Gu et al., 1999; Sun et al., 1999).

Elkin et al. (2010) showed experimentally that brain tissue has an FCD with a negative valence ($\omega = -1$). The value of the FCD of the solid phase changes as a result of deformation, the fluid volume fraction, and other factors such as changes in pH levels of the mixture (Gu et al., 1998; Lai et al., 1991). For simplicity in the derivations that follow, the FCD will only depend on the deformation and fluid volume fraction (Gu et al., 1998; Sun et al., 1999):

$$c^F = \frac{c_r^F}{1 + \frac{\mathrm{tr}(\boldsymbol{\varepsilon})}{\phi_r^w}} \cong c_r^F \left(1 - \frac{\mathrm{tr}(\boldsymbol{\varepsilon})}{\phi_r^w}\right);\ \frac{\mathrm{tr}(\boldsymbol{\varepsilon})}{\phi_r^w} \ll 1. \tag{2.11}$$

In equation (2.11), $\boldsymbol{\varepsilon}$ is the infinitesimal strain of the mixture and $\mathrm{tr}(\boldsymbol{\varepsilon})$ is the dilatation (Lai et al., 1991). c_r^F and ϕ_r^w are respectively the FCD and the fluid volume fraction in the reference configuration (Gu et al., 1998; Lai et al., 1991; Sun et al., 1999). According to Sun et al. (1999), the volume fraction of the solid phase can be represented in a similar form as equation (2.11) due to the intrinsic incompressibility of the mixture:

$$\begin{aligned} \phi^s &= \frac{\phi_r^s}{1 + \mathrm{tr}(\boldsymbol{\varepsilon})}, \\ &\cong \phi_r^s\Big(1 - \mathrm{tr}(\boldsymbol{\varepsilon})\Big);\ \mathrm{tr}(\boldsymbol{\varepsilon}) \ll 0. \end{aligned} \tag{2.12}$$

By invoking equation (2.5) and equation (2.12), the volume fraction of the fluid phase can be represented as a function of the fluid volume fraction in the reference configuration, ϕ_r^w, and dilatation, $\mathrm{tr}(\boldsymbol{\varepsilon})$ as (Gu et al., 1998):

$$\phi^w = \frac{\phi_r^w + \mathrm{tr}(\boldsymbol{\varepsilon})}{1 + \mathrm{tr}(\boldsymbol{\varepsilon})},$$

$$\cong \phi_r^w + \left(1 - \phi_r^w\right)\mathrm{tr}(\boldsymbol{\varepsilon});\ \mathrm{tr}(\boldsymbol{\varepsilon}) \ll 0. \tag{2.13}$$

The FCD, c^F, must be conserved during deformation according to Sun et al. (1999):

$$\frac{\partial\left(\phi^w c^F\right)}{\partial t} + \mathrm{div}(\phi^w c^F \boldsymbol{v^s}) = 0. \tag{2.14}$$

2.3 Volume Fluxes

The volume flux of the fluid phase and the ionic molar fluxes can be written with respect to the solid phase since the volume fraction of the ionic phase is negligible (Gu et al., 1999). According to Gu et al. (1999); Sun et al. (1999), the volume flux of the fluid and the ionic molar fluxes of the α^{th} ionic species can be represented as:

$$\boldsymbol{J_w} = \phi^w\left(\boldsymbol{v^w} - \boldsymbol{v^s}\right), \tag{2.15}$$

$$\boldsymbol{J_\alpha} = \phi^w c_\alpha\left(\boldsymbol{v^\alpha} - \boldsymbol{v^s}\right). \tag{2.16}$$

When ions move in a mixture, an electric current is generated that accompanies each ion. In Figure 2.1, ions move in and out of brain cells through ion channels (and pumps which are not pictured in Figure 2.1) and generate an electric current (Kandel et al., 2012). The current density associated with each ionic species can be represented as (Gu et al., 1999):

$$\left(\boldsymbol{I_e}\right)_\alpha = \mathrm{F_c} z_\alpha \boldsymbol{J_\alpha};\ \alpha = 1,\ 2,\ \ldots,\ n, \tag{2.17}$$

where $\mathrm{F_c}$ is Faraday's Constant. The definition provided by Gu et al. (1999) for the electric current density carried by all ions and fixed charges is:

$$\boldsymbol{I_e} = \mathrm{F_c}\sum_{\alpha=1}^{n} z_\alpha \boldsymbol{J_\alpha}. \tag{2.18}$$

Using equation (2.9), equation (2.14), and equation (2.10), it can be shown that (Sun et al., 1999):

$$\mathrm{div}\,\boldsymbol{I_e} = 0. \tag{2.19}$$

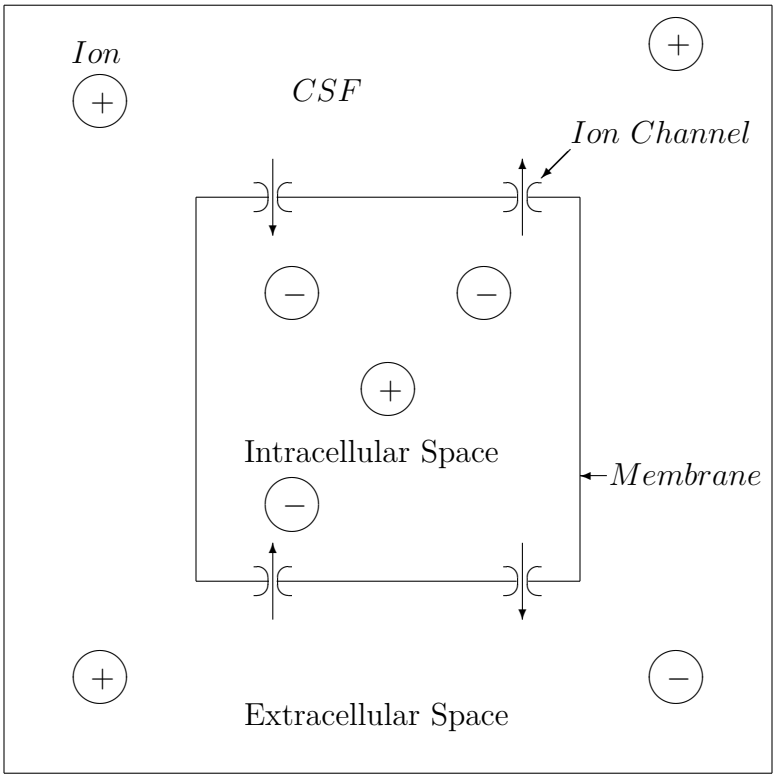

Figure 2.1. Visual representation of neurons: the inner square represents a neuron and the space outside of the inner square represents the extracellular space

2.4 Momentum Equations

The balance of momentum of the mixture when various approximations in Gu et al. (1998, 1999); Lai et al. (1991); Sun et al. (1999) are applied is:

$$\operatorname{div} \boldsymbol{\sigma} = \mathbf{0}. \tag{2.20}$$

The momentum equations for the fluid phase and ionic species ($\alpha = 1,\ 2,\ \ldots,\ n$) are (Gu et al., 1998, 1999):

$$-\rho^w \operatorname{grad}(\mu^w) + \sum_{\beta=s,w,1}^{n} f_{w\beta}\left(\boldsymbol{v}^{\boldsymbol{\beta}} - \boldsymbol{v}^{\boldsymbol{w}}\right) = \mathbf{0}, \tag{2.21}$$

$$-\rho^\alpha \operatorname{grad}(\mu^\alpha) + \sum_{\beta=s,w,1}^{n} f_{\alpha\beta}\left(\boldsymbol{v}^{\boldsymbol{\beta}} - \boldsymbol{v}^{\boldsymbol{\alpha}}\right) = \mathbf{0}, \tag{2.22}$$

where μ^w and μ^α are the chemical potentials of the fluid and ionic phases respec-

tively. The scalar values $f_{\alpha\beta}$ and $f_{w\beta}$ are frictional coefficients between the two components of the mixture denoted by subscripts $\alpha,\ \beta,\ w$ (Gu et al., 1999; Lai et al., 1991; Sun et al., 1999). Equation (2.22) can be used to solve for the relative velocities of the fluid and ionic species with respect to the velocity of the solid phase and all ionic species $\alpha = 1,\ 2,\ \ldots,\ n$ as (Gu et al., 1999):

$$\boldsymbol{v^w} - \boldsymbol{v^s} = \sum_{\beta=w,1}^{n} B_{w\beta}\rho^{\beta}\,\mathrm{grad}\,(\mu^w), \tag{2.23}$$

$$\boldsymbol{v^\alpha} - \boldsymbol{v^s} = \sum_{\beta=w,1}^{n} B_{\alpha\beta}\rho^{\beta}\,\mathrm{grad}\,(\mu^\alpha). \tag{2.24}$$

The coefficients $B_{w\beta}$ and $B_{\alpha\beta}$ are given by (Gu et al., 1998, 1999):

$$B_{w\beta} = -\frac{1}{f_{ws}};\ \beta = w,\ 1,\ 2,\ \ldots,\ n, \tag{2.25}$$

$$B_{\alpha\beta} = -\frac{1}{f_{ws}} - \frac{\delta_{\alpha\beta}}{f_{w\alpha}};\ \begin{cases} \beta = 1,\ 2,\ \ldots,\ n \\ \alpha = 1,\ 2,\ \ldots,\ n \end{cases}, \tag{2.26}$$

where $\delta_{\alpha\beta}$ is the Kronecker delta. According to Gu et al. (1998, 1999); Lai et al. (1991); Sun et al. (1999), the frictional coefficients have the properties:

$$f_{ij} = f_{ji};\ \begin{cases} i = s,\ w,\ 1,\ 2,\ \ldots,\ n \\ j = s,\ w,\ 1,\ 2,\ \ldots,\ n \\ i \neq j \end{cases}, \tag{2.27}$$

and

$$f_{ii} = 0;\ i = s,\ w,\ 1,\ 2,\ \ldots,\ n. \tag{2.28}$$

2.5 Constitutive Equations

As in Sun et al. (1999), the constitutive equation of an "isotropic hydrated charged mixture with infinitesimal deformation" (Sun et al., 1999) is:

$$\boldsymbol{\sigma} = -p\boldsymbol{I} - T_c\boldsymbol{I} + \lambda_s \mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I} + 2\mu_s\boldsymbol{\varepsilon}, \tag{2.29}$$

where p is the hydrostatic pressure, λ_s and μ_s are the Lamè coefficients of the solid

phase, and T_c is the chemical expansion stress (Gu et al., 1999; Lai et al., 1991).

The constitutive equations for the fluid chemical potential and ionic electro-chemical potentials can be adapted from previous triphasic theories (Gu et al., 1998, 1999; Sun et al., 1999). The constitutive equation for the chemical potential of the fluid is (Gu et al., 1999):

$$\mu^w = \mu_r^w + \frac{1}{\rho_{\mathrm{T}}^w}\left(p - RT\sum_{\alpha=1}^{n}(\Phi_\alpha c_\alpha) + \Xi_w \mathrm{tr}(\boldsymbol{\varepsilon})\right), \tag{2.30}$$

where R is the universal gas constant, T is the absolute temperature (in Kelvins), and Ξ_w is a coupling coefficient associated with the fluid phase (Gu et al., 1999). The electro-chemical potentials of the ions ($\alpha = 1,\ 2,\ \ldots,\ n$) are (Gu et al., 1998, 1999; Sun et al., 1999):

$$\mu^\alpha = \mu_r^\alpha + \left(\frac{RT}{M^\alpha}\right)\ln(\gamma_\alpha c_\alpha) + \frac{z_\alpha \mathrm{F_c}\psi}{M^\alpha}, \tag{2.31}$$

where γ_α are the activity coefficients for the α^{th} ionic species and ψ is the electrical potential of the tissue (Gu et al., 1999; Sun et al., 1999). The constitutive relation for the volume flux, obtained from combining equation (2.15) with equations (2.23) and (2.30) is:

$$\begin{aligned}\boldsymbol{J_w} = -k_0\Bigg(&\mathrm{grad}\,(p) + RT\sum_{\alpha=1}^{n}(1-\Phi_\alpha)\,\mathrm{grad}\,(c_\alpha)\\ &+ \Xi_w\,\mathrm{grad}\,(\mathrm{tr}(\boldsymbol{\varepsilon})) - \omega \mathrm{F_c} c^F\,\mathrm{grad}\,(\psi)\Bigg),\end{aligned} \tag{2.32}$$

where $k_0 = (\phi^w)^2/f_{ws}$ (Gu et al., 1998, 1999). The ionic molar flux for the α^{th} ion can be redefined in a similar manner as the fluid volume flux using the constitutive equations for the ionic electro-chemical potential, equation (2.31), as (Gu et al., 1999; Sun et al., 1999):

$$\boldsymbol{J_\alpha} = c_\alpha \boldsymbol{J_w} - \phi^w D_\alpha\,\mathrm{grad}\,(c_\alpha) - \phi^w D_\alpha c_\alpha\left(\frac{z_\alpha \mathrm{F_c}}{RT}\right)\mathrm{grad}\,(\psi), \tag{2.33}$$

where:

$$D_\alpha = \frac{RT\phi^w c_\alpha}{f_{w\alpha}}, \tag{2.34}$$

are the ionic diffusitivities for each ion species in the mixture (Gu et al., 1999).

2.6 Governing Equations

By applying the local form of the balance of momentum, equation (2.20), to the definition for the stress given in equation (2.29), the governing equation for the balance of momentum can be stated as (Sun et al., 1999):

$$\operatorname{div}\left(\lambda_s \operatorname{tr}(\boldsymbol{\varepsilon})\boldsymbol{I} + 2\mu_s\boldsymbol{\varepsilon}\right) - \operatorname{grad}\left(p\right) - \operatorname{grad}\left(T_c\right) = \mathbf{0}. \tag{2.35}$$

The balance of mass for the fluid phase, equation (2.8), can be represented in terms of the volume flux of the fluid, equation (2.15) as:

$$\operatorname{div}\left(\boldsymbol{J_w}\right) + \operatorname{div}\left(\boldsymbol{v^s}\right) = 0. \tag{2.36}$$

The divergence of the fluid volume flux can be obtained as:

$$\begin{aligned}\operatorname{div}\left(\boldsymbol{J_w}\right) = -k_0\Bigg(&\nabla^2\left(p\right) + RT\sum_{\alpha=1}^{n}\left(1-\Phi_\alpha\right)\nabla^2\left(c_\alpha\right) + \Xi_w\nabla^2\left(\operatorname{tr}(\boldsymbol{\varepsilon})\right)\\ &- \omega \mathrm{F_c}\left(\operatorname{grad}\left(c^F\right)\cdot\operatorname{grad}\left(\psi\right) + c^F\nabla^2\left(\psi\right)\right)\Bigg),\end{aligned} \tag{2.37}$$

where $\nabla^2\left(\cdot\right) = \operatorname{div}\left(\operatorname{grad}\left(\cdot\right)\right)$. Using the form of $\operatorname{div}\left(\boldsymbol{J_w}\right)$ in equation (2.37), equation (2.36) can be rewritten as:

$$\begin{aligned}\operatorname{div}\left(\boldsymbol{v^s}\right) - k_0\Bigg(&\nabla^2\left(p\right) + RT\sum_{\alpha=1}^{n}\left(1-\Phi_\alpha\right)\nabla^2\left(c_\alpha\right) + \Xi_w\nabla^2\left(\operatorname{tr}(\boldsymbol{\varepsilon})\right)\\ &- \omega \mathrm{F_c}\left(\operatorname{grad}\left(c^F\right)\cdot\operatorname{grad}\left(\psi\right) + c^F\nabla^2\left(\psi\right)\right)\Bigg) = 0.\end{aligned} \tag{2.38}$$

Following the same derivation which led to the result in equation (2.36), equation (2.9) can be combined with the definition of the ionic molar flux, equation (2.16), to result in (Sun et al., 1999):

$$\frac{\partial\left(\phi^w c_\alpha\right)}{\partial t} + \operatorname{div}\left(\phi^w c_\alpha \boldsymbol{v^s}\right) + \operatorname{div}\left(\boldsymbol{J_\alpha}\right) = 0;\ \alpha = 1,\ 2,\ \ldots,\ n. \tag{2.39}$$

Using equation (2.19), an expression analogous to equation 41 in Sun et al. (1999) can be written as:

$$\sum_{\alpha=1}^{n} z_\alpha \operatorname{div}\left(\boldsymbol{J_\alpha}\right) = 0, \tag{2.40}$$

which states how the ionic molar fluxes for each species are related. For simplicity, the following notation is introduced (Lu et al., 2010; Sun et al., 1999):

$$c^k = \sum_{\alpha=1}^{n} c_\alpha. \tag{2.41}$$

By taking a sum over all values of α in equation (2.39), an expression involving c^k can be obtained (Sun et al., 1999) as:

$$\frac{\partial\left(\phi^w c^k\right)}{\partial t} + \operatorname{div}\left(\phi^w c^k \boldsymbol{v^s}\right) + \operatorname{div}\left(\sum_{\alpha=1}^{n} \boldsymbol{J_\alpha}\right) = 0. \tag{2.42}$$

Chapter 3
A Linearized Triphasic Model

3.1 Equations of the Linear Model

Lu et al. (2010) linearized the governing equations of the triphasic model proposed by Gu et al. (1999); Lai et al. (1991). The exact method of linearization along with exact definitions of various quantities are described in the supplementary material of Lu et al. (2010). The linear system results from applying the equations developed in chapter 2 to a mixture of two ionic species ($n = 2$). The two ionic species have valances of $z_{\pm} = \pm 1$ for the positive and the negative species. It should also be noted that the valence of the FCD is negative (Lu et al., 2010). The following system of parabolic partial differential equations is then obtained (Lu et al., 2010):

$$\frac{\partial}{\partial t}\begin{pmatrix} e \\ \gamma \end{pmatrix} = [\boldsymbol{A}]\,\nabla^2 \begin{pmatrix} e \\ \gamma \end{pmatrix}, \tag{3.1}$$

where the unknowns are:

$$e = \mathrm{tr}(\boldsymbol{\varepsilon}), \tag{3.2}$$

$$\gamma = \frac{RT\left(\delta c^k\right)}{\lambda_s + 2\mu_s}. \tag{3.3}$$

$$\tag{3.4}$$

The matrix $[\boldsymbol{A}]$ is a combination of all the physical parameters introduced earlier in the triphasic theory with entries:

$$[\boldsymbol{A}] = \begin{pmatrix} A_1 & -A_2 \\ -A_5 & A_4 \end{pmatrix}. \tag{3.5}$$

Due to the complexity of this matrix, the method of obtaining the values of the entries of $[\boldsymbol{A}]$ will not be presented.[1]

The quantity e is the dilatation of the mixture and γ is a term that is proportional to the sum of the concentrations, c^k, of both ionic species (Gu et al., 1999; Lu et al., 2010). The quantity, c^k, is the sum of the concentrations of all ionic species in the mixture and is defined as (Lu et al., 2010; Sun et al., 1999):

$$c^k = \sum_{\alpha=1}^{n=2} c_\alpha = c_+ + c_-. \tag{3.6}$$

In equation (3.3), δc^k is a small perturbation of c^k from its original value: $\delta c^k = c^k - c_0^k$. Equation (3.1) in one-dimension has an analytical solution:

$$\frac{\partial}{\partial t}\begin{pmatrix} e \\ \gamma \end{pmatrix} = [\boldsymbol{A}]\frac{\partial^2}{\partial z^2}\begin{pmatrix} e \\ \gamma \end{pmatrix}. \tag{3.7}$$

3.2 Analytical Solution

The matrix $[\boldsymbol{A}]$ can be represented as a product of three different matrices $[\boldsymbol{A}] = [\boldsymbol{M}][\boldsymbol{\Lambda}][\boldsymbol{M}]^{-1}$ where the columns of $[\boldsymbol{M}]$ are the eigenvectors of $[\boldsymbol{A}]$ and $[\boldsymbol{\Lambda}]$ is a diagonal matrix in which the elements on the main diagonal are the eigenvalues of $[\boldsymbol{A}]$ (Abdi, 2007). It should be noted that: $dim([\boldsymbol{M}]) = dim([\boldsymbol{\Lambda}]) = dim([\boldsymbol{A}]) = 2$ and the three matrices, $\{[\boldsymbol{A}], [\boldsymbol{M}], [\boldsymbol{\Lambda}]\}$, are invertible. Using the matrix $[\boldsymbol{M}]$, equation (3.7) becomes:

$$\frac{\partial}{\partial t}\begin{pmatrix} e \\ \gamma \end{pmatrix} = [\boldsymbol{M}][\boldsymbol{\Lambda}][\boldsymbol{M}]^{-1}\frac{\partial^2}{\partial z^2}\begin{pmatrix} e \\ \gamma \end{pmatrix}. \tag{3.8}$$

Since $[\boldsymbol{A}]$ is a constant matrix, the eigenvalues and eigenvectors of $[\boldsymbol{A}]$ will also be constant. Knowing this, another column vector can be defined as:

[1] The reader is directed to the supplementary material of Lu et al. (2010).

$$\begin{pmatrix} f \\ g \end{pmatrix} = [\boldsymbol{M}]^{-1} \begin{pmatrix} e \\ \gamma \end{pmatrix}. \tag{3.9}$$

Using this relation, equation (3.8) can be transformed such that a solution can be found for the column vector $(f,\ g)^{\mathrm{T}}$ as:

$$\frac{\partial}{\partial t} \begin{pmatrix} f \\ g \end{pmatrix} = [\boldsymbol{\Lambda}] \frac{\partial^2}{\partial z^2} \begin{pmatrix} f \\ g \end{pmatrix}. \tag{3.10}$$

Equation (3.10) can be solved using the method of separation of variables, but only for favorable boundary conditions. The boundary and initial conditions that were chosen are similar to the ones given by Malakpoor et al. (2006), namely:

$$\begin{pmatrix} e\,(0,t) \\ \gamma\,(0,t) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

$$\frac{\partial}{\partial z} \begin{pmatrix} e\,(\ell,t) \\ \gamma\,(\ell,t) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{3.11}$$

$$\begin{pmatrix} e\,(z,0) \\ \gamma\,(z,0) \end{pmatrix} = \begin{pmatrix} e_0 \\ \gamma_0 \end{pmatrix}, \tag{3.12}$$

where ℓ is the length of the one-dimensional domain occupied by the mixture $(\ell > 0)$ and $(e_0,\ \gamma_0)^{\mathrm{T}}$ are constant values for e and γ at time $t = 0$. Equation (3.12) means that there is a nonzero, constant dilatation and combined ion concentration throughout the domain at $t = 0$. Equation (3.11) state that there is no dilatation or net combined ionic concentration at $z = 0$, and the dilatation and ionic concentrations only change as a function of time at $z = \ell$. The boundary conditions in equation (3.11) were chosen so that a Fourier series analytical solution can be easily obtained and their physical significance is not important for this purpose. The conditions in equations (3.11) and (3.12) can be transformed into values defined for $(f,\ g)^{\mathrm{T}}$ using equation (3.9) in order to form a solution to equation (3.10).

The solution to equation (3.10) can then be transformed by equation (3.9) in order to obtain the (analytical) Fourier-series solution to equation (3.7) as:

$$\begin{pmatrix} e \\ \gamma \end{pmatrix} = \frac{4}{\pi} \sum_{n=0}^{\infty} \left\{ \frac{\sin\left(\omega_n z\right)}{2n+1} \left[\boldsymbol{M}\right] \left[\boldsymbol{R_n}\right] \left[\boldsymbol{M}\right]^{-1} \begin{pmatrix} e_0 \\ \gamma_0 \end{pmatrix} \right\}, \tag{3.13}$$

where $[\boldsymbol{R_n}]$ and ω_n are defined as:

$$[\boldsymbol{R_n}] = \begin{pmatrix} \exp\left(-\omega_n^2 \lambda_{11} t\right) & 0 \\ 0 & \exp\left(-\omega_n^2 \lambda_{22} t\right) \end{pmatrix}, \tag{3.14}$$

$$\omega_n = (2n+1)\,\frac{\pi}{2\ell}. \tag{3.15}$$

This analytical solution is only permissible under the specified conditions in equations (3.11) and (3.12) and if these conditions are not met, then the solution to equation (3.7) described by equation (3.13) is not valid. The analytical solution is very similar to the one obtained by Malakpoor et al. (2006) for a similar system of parabolic partial differential equations (but for different parameters of the mixture). The solution obtained in equation (3.13) is one possible analytical solution to the system in equation (3.7).

3.3 Numerical Solution

3.3.1 Justification for the Numerical Solution

In order to obtain solutions with arbitrary boundary conditions, numerical methods must be implemented to solve the system in equation (3.7). The method that was chosen to solve for $(e,\ \gamma)^{\mathrm{T}}$ in equation (3.7) was the Crank-Nicholson method, which is a finite difference method for solving parabolic partial differential equations (Smith, 1986). The finite difference algorithm solves the system in equation (3.10), which can be written as two separate equations:

$$\frac{\partial f}{\partial t} = \lambda_{11} \frac{\partial^2 f}{\partial z^2}, \tag{3.16}$$

$$\frac{\partial g}{\partial t} = \lambda_{22} \frac{\partial^2 g}{\partial z^2}. \tag{3.17}$$

The important thing to note about equations (3.16) and (3.17) is that they can be solved independently of one another. This means that a standard Crank-Nicholson method for a scalar parabolic PDE can be implemented for solving both equations (3.16) and (3.17).

3.3.2 Crank-Nicholson Method

The standard Crank-Nicholson method as described in Smith (1986) (using notation found in Harder (2012)) will be adapted for use to solve Equations (3.16) and (3.17). Both equations (3.16) and (3.17) can be solved numerically using the same algorithm since only the parameter $\lambda_{ii};\ i = 1,\ 2$ is different between them. In order to demonstrate how the Crank-Nicholson method works, define a function a such that:

$$\frac{\partial a}{\partial t} = \lambda_{ii}\frac{\partial^2 a}{\partial z^2}. \tag{3.18}$$

Numerically, the function $a\left(z_i, t_n\right)$ can be represented by discrete values a_i^n where the i^{th} index indicates a point in space, while n indicates the time interval. The size of the grid is $(N_z \times N_t)$ where N_z is the number of points in the spatial domain and N_t is the number of points in the time domain. Equation (3.18) can be written in the Crank-Nicholson method as follows:

$$\frac{a_i^{n+1} - a_i^n}{\Delta t} = \frac{\lambda_{ii}}{2\left(\Delta z\right)^2}\left(a_{i+1}^{n+1} - 2a_i^{n+1} + a_{i-1}^{n+1} + a_{i+1}^n - 2a_i^n + a_{i-1}^n\right), \tag{3.19}$$

over $i = 1,\ 2,\ \ldots,\ N_z$ and $n = 1,\ 2,\ \ldots,\ N_t$. Defining the quantity:

$$r\left(\lambda_{ii}\right) = \frac{\lambda_{ii}\Delta t}{\left(\Delta z\right)^2}, \tag{3.20}$$

the generic parabolic PDE in equation (3.18) can be simplified to (Smith, 1986):

$$-ra_{i+1}^{n+1} + 2\left(1 + r\right)a_i^{n+1} - ra_{i-1}^{n+1} = ra_{i+1}^n + 2\left(1 - r\right)a_i^n + ra_{i-1}^n. \tag{3.21}$$

The quantities in equation (3.21) can be represented as a matrix-vector system in which:

$$\left\{\boldsymbol{a^{n+1}}\right\} = \begin{pmatrix} a_1^{n+1} \\ a_2^{n+1} \\ \vdots \\ a_{N_z-1}^{n+1} \\ a_{N_z}^{n+1} \end{pmatrix} \quad \text{and} \quad \{\boldsymbol{a^n}\} = \begin{pmatrix} a_1^n \\ a_2^n \\ \vdots \\ a_{N_z-1}^n \\ a_{N_z}^n \end{pmatrix}. \tag{3.22}$$

Since the entries at $i = 1$ and $i = N_z$ in equation (3.22) are represented by boundary conditions, the matrix-vector system that is employed to solve equation (3.21) is:

$$[\boldsymbol{J}]\left\{\boldsymbol{a^{n+1}_{mod}}\right\} = [\boldsymbol{K}]\left\{\boldsymbol{a^{n}_{mod}}\right\} + \begin{pmatrix} ra_1^n \\ 0 \\ \vdots \\ 0 \\ ra_{N_z}^n \end{pmatrix}, \tag{3.23}$$

where the vectors $\left\{\boldsymbol{a^{n+1}_{mod}}\right\}$ and $\left\{\boldsymbol{a^{n}_{mod}}\right\}$ are defined as:

$$\left\{\boldsymbol{a^{n+1}_{mod}}\right\} = \begin{pmatrix} a_2^{n+1} \\ a_3^{n+1} \\ \vdots \\ a_{N_z-2}^{n+1} \\ a_{N_z-1}^{n+1} \end{pmatrix} \quad \text{and} \quad \left\{\boldsymbol{a^{n}_{mod}}\right\} = \begin{pmatrix} a_2^{n} \\ a_3^{n} \\ \vdots \\ a_{N_z-2}^{n} \\ a_{N_z-1}^{n} \end{pmatrix}. \tag{3.24}$$

The matrices $[\boldsymbol{J}]$ and $[\boldsymbol{K}]$ are tridiagonal matrices whose entries are (Smith, 1986):

$$[\boldsymbol{J}] = \begin{pmatrix} 2(1+r) & -r & 0 & 0 & \dots & 0 & 0 \\ -r & 2(1+r) & -r & 0 & \dots & 0 & 0 \\ 0 & -r & \ddots & \ddots & & \vdots & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & -r & 0 \\ 0 & \dots & & 0 & -r & 2(1+r) & -r \\ 0 & \dots & & 0 & 0 & -r & 2(1+r) \end{pmatrix}, \tag{3.25}$$

$$[\boldsymbol{K}] = \begin{pmatrix} 2(1-r) & r & 0 & 0 & \dots & 0 & 0 \\ r & 2(1-r) & r & 0 & \dots & 0 & 0 \\ 0 & r & \ddots & \ddots & & \vdots & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & r & 0 \\ 0 & \dots & & 0 & r & 2(1-r) & r \\ 0 & \dots & & 0 & 0 & r & 2(1-r) \end{pmatrix}. \tag{3.26}$$

To obtain a proper solution for a, equation (3.23) must be solved for every iteration $n = 1, \dots, N_t$. At $n = 1$, the initial condition can be applied as:

$$\left\{\boldsymbol{a}_{\boldsymbol{mod}}^{\boldsymbol{1}}\right\} = \begin{pmatrix} a(z,0) \\ \vdots \\ a(z,0) \end{pmatrix}, \tag{3.27}$$

and then the following algorithm is employed for pure Dirichlet boundary conditions:

1. Modify entries in $[\boldsymbol{J}]$ or $\left\{\boldsymbol{a}_{\boldsymbol{mod}}^{\boldsymbol{n+1}}\right\}$ according to the boundary conditions
2. Solve for $\left\{\boldsymbol{a}_{\boldsymbol{mod}}^{\boldsymbol{n+1}}\right\}$ in equation (3.23)
3. Apply the lower boundary condition to a_1^{n+1}
4. Apply the upper boundary condition to $a_{N_z}^{n+1}$

Every value of a_i^n can be solved by repeating the above steps for $n = 2, \dots, N_t - 1$ (Smith, 1986). Note that in both Appendix B, the Thomas algorithm is employed to invert some matrices involving terms in equation (3.23) (Chapra, Stephen C. and Canale, Raymond P., 2010). Resolving as Dirichlet boundary condition involves simply replacing the value of a_i^{n+1} with the given boundary condition and accounting for it on the right hand side of equation (3.23) as a known quantity as:

$$[\boldsymbol{J}]\left\{\boldsymbol{a}_{\boldsymbol{mod}}^{\boldsymbol{n+1}}\right\} = [\boldsymbol{K}]\left\{\boldsymbol{a}_{\boldsymbol{mod}}^{\boldsymbol{n}}\right\} + \begin{pmatrix} ra_1^n + ra(z_1, t_n) \\ 0 \\ \vdots \\ 0 \\ ra_{N_z}^n + ra(z_{N_z}, t_n) \end{pmatrix}, \tag{3.28}$$

where $a(z_1, t_n)$ is a Dirichlet boundary condition at $z = 0$ and $a(z_{N_z}, t_n)$ is a Dirichlet boundary condition at $z = \ell$.

Incorporating a Neumann boundary condition in the Crank-Nicholson method involves approximating the first order derivative at a boundary (Smith, 1986). It is advantageous to adopt a Neumann boundary condition that employs a higher-order approximation for the first derivative since the error is of a higher order ($\mathbb{O}[\Delta z]^2$. as opposed to $\mathbb{O}[\Delta z]$) (Harder, 2012). For problems where the $z = \ell$ boundary condition is a Neumann boundary condition, the numeric finite difference for the first derivative is:

$$\left.\frac{\partial a}{\partial z}\right|_{z=\ell} \cong \frac{3a_{N_z}^{n+1} - 4a_{N_z-1}^{n+1} + a_{N_z-2}^{n+1}}{2\Delta z}. \tag{3.29}$$

Likewise for a Neumann boundary condition at $z = 0$:

$$\left.\frac{\partial a}{\partial z}\right|_{z=0} \cong \frac{-3a_1^{n+1} + 4a_2^{n+1} - a_3^{n+1}}{2\Delta z}. \tag{3.30}$$

The matrix $[\boldsymbol{J}]$ needs to be modified in the following manner if the Neumann boundary condition is at $z = \ell$:

$$[\boldsymbol{J}] = \begin{pmatrix} 2(1+r) & -r & 0 & 0 & \dots & 0 & 0 \\ -r & 2(1+r) & -r & 0 & \dots & 0 & 0 \\ 0 & -r & \ddots & \ddots & & \vdots & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & -r & 0 \\ 0 & \dots & & 0 & -r & 2(1+r) & -r \\ 0 & \dots & & 0 & 0 & -\frac{2}{3}r & 2+\frac{2}{3}r \end{pmatrix}. \tag{3.31}$$

If the Neumann boundary condition were on the lower limit at $z = 0$, then $[\boldsymbol{J}]$ is modified as:

$$[\boldsymbol{J}] = \begin{pmatrix} 2+\frac{2}{3}r & -\frac{2}{3}r & 0 & 0 & \dots & 0 & 0 \\ -r & 2\,(1+r) & -r & 0 & \dots & 0 & 0 \\ 0 & -r & \ddots & \ddots & & \vdots & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & -r & 0 \\ 0 & \dots & & 0 & -r & 2\,(1+r) & -r \\ 0 & \dots & & 0 & 0 & -r & 2\,(1-r) \end{pmatrix}. \tag{3.32}$$

Implementing the boundary conditions in equations (3.31) and (3.32) satisfy step one of the algorithm if there is one or more Neumann boundary condition present. Solving for a_1^{n+1} and $a_{N_z}^{n+1}$ using equation (3.30) satisfies step three and step four in the algorithm. Steps one through four in the algorithm must be repeated in order to obtain an entire grid of values for $a_i^n;\ i = 1, \dots, N_z$ & $n = 1, \dots, N_t$.

3.3.3 Numerical Solution using Crank-Nicholson Method

The Crank-Nicholson method was used to solve equations (3.16) and (3.17) independently of one another. What must be kept in mind though is that while the equations for f and g can be solved independently of one another, what is ultimately desired is a finite difference solution for e and γ. This means that some constraints must be placed on how the temporal and spatial sizes of the grid are chosen. The first constraint is that N_z and N_t must be the same for both the solutions to f_i^n and g_i^n. The Crank-Nicholson method is (semi) implicit and it converges for any size Δz and Δt, but in order to prevent some oscillations/errors in the solution, another constraint in the form of a CFL stability condition:

$$\Delta t \leq \frac{1}{2} \frac{(\Delta z)^2}{max\,|\lambda_{ii}|}, \tag{3.33}$$

for $i = 1,\ 2$ can be imposed to choose the size of Δt. The CFL condition is not necessary in using the Crank-Nicholson method, but it does provide a good estimate for the size of Δt. In equation (3.33), $max\,|\lambda_{ii}|$ is the largest of the absolute values of the entries of $[\boldsymbol{\Lambda}]$. After the solutions for all values of f_i^n and g_i^n

have been calculated, they must be mapped to $(e,\ \gamma)^{\mathrm{T}}$ using $[\boldsymbol{M}]$ in order to solve equation (3.7). Both e and γ can be calculated point-wise from f and g using:

$$\begin{pmatrix} e_i^n \\ \gamma_i^n \end{pmatrix} = [\boldsymbol{M}] \begin{pmatrix} f_i^n \\ g_i^n \end{pmatrix};\ \begin{cases} i = 1,\ 2,\ \ldots,\ N_z \\ n = 1,\ 2,\ \ldots,\ N_t \end{cases}. \tag{3.34}$$

The values of $(e,\ \gamma)^{\mathrm{T}}$ can then be compared to the analytic solution in equation (3.13).

3.4 Comparison Between the Analytical and Numeric Solutions

The code for generating solutions to the analytic and numeric solutions to equation (3.8) was implemented in MATLAB and given in appendix B. The parameters used in the numerical simulations were taken from Lu et al. (2010); Malakpoor et al. (2006) and are displayed in Table 3.1.

In addition to the parameters in Table 3.1, the length of time chosen was arbitrarily set at 3600 seconds in order to observe a long-term, steady-state solution to equation (3.7). The number of steps in space, N_z, was also arbitrarily set at $N_z = 100$ in order to give enough grid points for an accurate solution. The step size in space was set by $\Delta z = (\ell - 0)/N_z$ and Δt was determined by equation (3.33). N_t was calculated by:

$$N_t = ceil\left(\frac{t_f - 0}{\Delta t}\right), \tag{3.35}$$

where $ceil\,()$ means round up to the greater integer value. Using the value of N_t in equation (3.35), Δt was recalculated as:

$$\Delta t = \frac{t_f - 0}{N_t}. \tag{3.36}$$

By rounding up the value of N_t in equation (3.35), it is not possible to violate the CFL condition, equation (3.33), when recalculating Δt. Equation (3.35) must be rounded up since N_t must be an integer. The graphs in Figures 3.1 and 3.2 show the results of solving the system in equation (3.7) both analytically and numerically.

Graphically Figures 3.1 and 3.2 show outstanding agreement between the nu-

Parameters Used in the Analysis			
Diffusion Coefficient of + Ion:	$D^+ =$	$13.3 * 10^{-10}$	$[m^2/s]$
Diffusion Coefficient of − Ion:	$D^- =$	$20.3 * 10^{-10}$	$[m^2/s]$
Added Lame Coefficients:	$\lambda_s + 2\mu_s =$	$4 * 10^9$	$[Pa]$
Hydraulic Permeability:	$k_0 =$	10^{-18}	$[m^4/(Ns)]$
Initial Concentration of + Ion:	$c_0^+ =$	10^2	$[mol/m^3]$
Initial Concentration of − Ion:	$c_0^- =$	10^2	$[mol/m^3]$
Initial Concentration of FCD:	$c_0^F =$	$-2 * 10^2$	$[mol/m^3]$
Universal Gas Constant:	$R =$	8.3145	$[J/(molK)]$
Tissue Temperature:	$T =$	293	$[K]$
Initial Fluid Volume Fraction:	$\phi_0^w =$	0.2	$[-]$
Final Length:†	$\ell =$	10^{-3}	$[m]$
Initial Length:†	$z_0 =$	0	$[m]$
Initial Dilatation:*	$e_0 =$	10^{-4}	$[-]$
Initial γ*:	$\gamma_0 =$	1.2181×10^{-4}	$[-]$
Quantity in $[\boldsymbol{A}]$†:	$A_1 =$	4.00×10^{-9}	$[m^2/s]$
Quantity in $[\boldsymbol{A}]$†:	$A_2 =$	6.90×10^{-10}	$[m^2/s]$
Quantity in $[\boldsymbol{A}]$†:	$A_4 =$	1.33×10^{-9}	$[m^2/s]$
Quantity in $[\boldsymbol{A}]$†:	$A_5 =$	4.80×10^{-14}	$[m^2/s]$

Table 3.1. Physical and derived parameters used in the analysis. The parameters denoted by † were taken or derived from (Lu et al., 2010), the parameters denoted by * were arbitrarily chosen, and all other parameters were taken from (Malakpoor et al., 2006).

meric and the analytic solutions. The plots in Figures 3.1 and 3.2 agree very well with one another and the relative errors displayed in Table 3.2 are at least three orders of magnitude smaller than the actual values of e and γ at the corresponding times.

It should be noted from Table 3.2 that the relative errors are systematic in that the values obtained for both e and γ of the numeric solution is consistently less than the values obtained from the corresponding analytic solution at any time t. As the number of spatial points, N_z, increases, the relative errors in both e and γ decrease over all points in time. So the numeric solution of both e and γ converges to the analytical solution as N_z increases.

The entire point of developing both an analytic and numeric solution to equation (3.7) was to verify that an accurate numerical solver to the linear triphasic model in Lu et al. (2010) had been developed. Figures 3.1 and 3.2 illustrate that

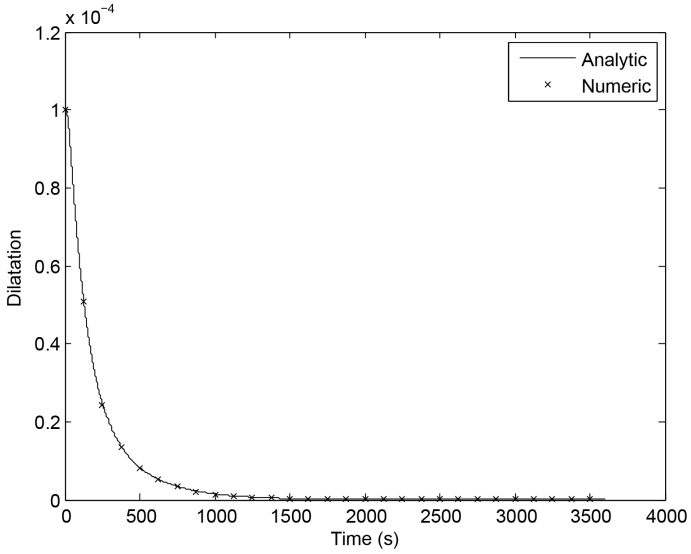

Figure 3.1. Graph of the dilatation, e, versus time at $z = \ell$.

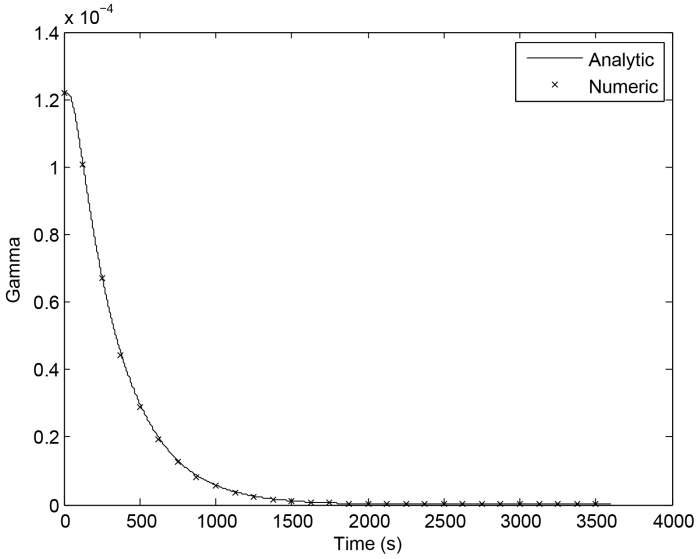

Figure 3.2. Graph of γ versus time at $z = \ell$.

the algorithm developed for solving equation (3.7) numerically is valid. So the same code can be employed to accurately solve the system in Lu et al. (2010) for any boundary/initial conditions.

The equations developed in chapter 2 can be linearized in situations where more than two ionic species are present. The model can be used to track the deformation of the mixture and changes in ionic concentrations in the brain where

Relative Errors		
Time (s)	Error in e	Error in γ
10	-8.4×10^{-9}	-1.4×10^{-14}
20	-1.3×10^{-7}	-8.7×10^{-10}
30	-2.9×10^{-7}	-1.5×10^{-8}
100	-8.0×10^{-7}	-5.9×10^{-7}
500	-3.1×10^{-7}	-9.8×10^{-7}
1000	-9.8×10^{-8}	-3.8×10^{-7}
1500	-2.8×10^{-8}	-1.1×10^{-7}
2000	-7.0×10^{-9}	-2.7×10^{-8}
3000	-3.8×10^{-10}	-1.5×10^{-9}
3600	-6.3×10^{-11}	-2.4×10^{-10}

Table 3.2. The relative error calculated as: $(\cdot)^{Error} = (\cdot)^{Numeric} - (\cdot)^{Analytic}$ for both e and γ. The relative error shows how much the numeric solution lags or leads the analytic solution to equation (3.7).

the ionic phase contains at least three ionic species (K^+, Na^+, and Cl^-) (Kandel et al., 2012). Gu et al. (1999); Lu et al. (2010); Sun et al. (1999) have shown that a triphasic model can track the chemo-mechanics of porous biological material and Figures 3.1–3.2 have shown that a successful finite difference program for mixtures of two ionic species had been developed.

The infrastructure developed for solving equation (3.7) numerically can also be employed to solve the equations of the triphasic model where there are more than two ionic species. If similar methods of linearization as Lu et al. (2010) can be employed, then the finite difference method used in this analysis can be adapted for use in mixtures of more than two ionic species. Since the accuracy of our numerical method has been verified against one possible analytic solution, any adaptation of the numerical solution should yield results that are fairly accurate.

Chapter 4
Hodgkin-Huxley Model

4.1 Model for Membrane Potential

Previous studies by Hodgkin and Huxley (1952a,b) present a model for calculating the membrane potential across an axon. The circuit model in Figure 4.1 is an extension of the model described in Hodgkin and Huxley (1952b) to allow for the passing of an arbitrary number of ions though the cell membrane.

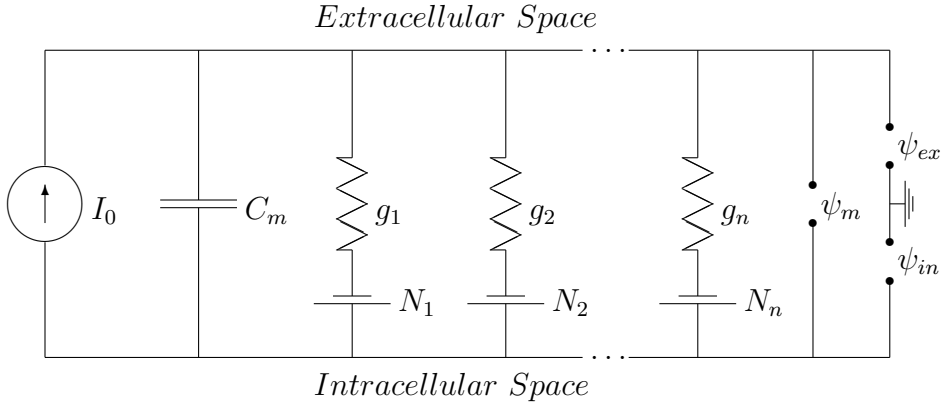

Figure 4.1. Equivalent circuit model for the Hodgkin-Huxley Equations extended for an arbitrary number of ion species which have channels in the membrane of the cell; inspired by Ermentrout and Terman (2010); Hodgkin and Huxley (1952b).

The equivalent circuit model has three main parts: conductance of each ionic species per unit area, g_α, that model the effect of open ion channels for each individual ion species (Na^+, K^+, Cl^-, etc...); a DC voltage source for each ion species which represents the corresponding reversal potential, N_α; and a capacitor which

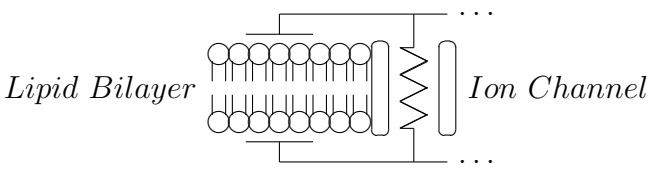

Figure 4.2. Representation of how a capacitor and resistor in parallel model the biological structure of a cell; inspired by Ermentrout and Terman (2010).

models the effect of the lipid bilayer (Ermentrout and Terman, 2010; Hodgkin and Huxley, 1952a) on the membrane potential. The quantity, I_0 is some externally applied, constant DC current per unit area which is used to model the depolarization of the axon which initiates an action potential (Ermentrout and Terman, 2010; Kandel et al., 2012; Medvedev, 2005). Figure 4.2 illustrates the connection between the biology of the cell and the circuit model in Figure 4.1 (Ermentrout and Terman, 2010). The reversal potentials, N_α can be represented using the famous *Nernst* equation as (Ermentrout and Terman, 2010; Gu et al., 1999):

$$N_\alpha = -\frac{RT}{z_\alpha \mathrm{F_c}} \ln\left(\frac{[\gamma_\alpha c_\alpha]_{in}}{[\gamma_\alpha c_\alpha]_{ex}}\right), \tag{4.1}$$

for $\alpha = 1,\ 2,\ \ldots,\ n$ where "in" stands for the intracellular quantities and "ex" stands for the extracellular quantities.

It should be noted that in Figure 4.1, Hodgkin and Huxley (1952b) explicitly states three ion channels for K^+, Na^+, and other ionic species (assumed by the author to be only Cl^-). The notation for "other species" is denoted in Hodgkin and Huxley (1952b) as "l". To keep a general number of ions present in the derivation, the equations presented in Hodgkin and Huxley (1952b) and Ermentrout and Terman (2010) were derived using the circuit in Figure 4.1.

4.2 Derivation of the Hodgkin Huxley Equations

To obtain the membrane potential, ψ_m, the circuit in Figure 4.1 must be solved using Kirchoff's second law (Ermentrout and Terman, 2010; Irwin, J. David and

Nelms, R. Mark, 2011):

$$\sum (I)_{out} - \sum (I)_{in} = 0. \tag{4.2}$$

Equation (4.2) can be applied to the circuit in Figure 4.1 which leads to:

$$C_m \frac{d(\psi_{ex} - \psi_{in})}{dt} - I_0 + \sum_{i=1}^{N} g_i \{\psi_{ex} - (N_i + \psi_{in})\} = 0. \tag{4.3}$$

The voltage, ψ_{ex} is the difference between the electric potential in the extracellular space and some reference electric potential called ground (represented by the three lines on the middle-right of Figure 4.1). Likewise, ψ_{in} is the difference between the electric potential of the intracellular space and ground. The membrane potential, ψ_m, which is the difference between the potential of the intracellular space and extracellular space is defined as:

$$\psi_m = \psi_{in} - \psi_{ex}. \tag{4.4}$$

Equation (4.3) can be rewritten using the quantity ψ_m as:

$$C_m \frac{d\psi_m}{dt} - I_0 + \sum_{i=1}^{N} g_i (\psi_m - N_i) = 0, \tag{4.5}$$

which is an ordinary differential equation that can be used to solve for ψ_m, the quantity that is desired.

4.3 Action Potential Equations

The membrane potential can be calculated for an action potential (Dayan and Abbott, 2001; Ermentrout and Terman, 2010; Hodgkin and Huxley, 1952a,b). This involves choosing $n = 3$ ionic constituents with (Dayan and Abbott, 2001; Ermentrout and Terman, 2010; Hodgkin and Huxley, 1952b):

- $\alpha = 1$, Potassium, K^+
- $\alpha = 2$, Sodium, Na^+
- $\alpha = 3$, "Leak" (l) which consists of all other ions present in the axon

Using these ionic constituents, equation (4.5) can be modified according to Dayan and Abbott (2001); Ermentrout and Terman (2010); Hodgkin and Huxley (1952b) as:

$$C_m \frac{d\psi_m}{dt} - \frac{I_0}{A} + \bar{g}_k n^4 (\psi_m - N_k) + \bar{g}_{Na} m^3 h (\psi_m - N_{Na}) + \bar{g}_l (\psi_m - N_l) = 0 \quad (4.6)$$

The values of $n,\ m$ and h are given by Dayan and Abbott (2001) as:

$$\frac{dn}{dt} = \alpha_n (1 - n) - \beta_n n, \quad (4.7)$$

$$\frac{dm}{dt} = \alpha_m (1 - m) - \beta_m m, \quad (4.8)$$

$$\frac{dh}{dt} = \alpha_h (1 - h) - \beta_h h, \quad (4.9)$$

$$\alpha_n = \frac{0.01 (\psi_m + 55)}{1 - \exp\left(-\frac{\psi_m + 55}{10}\right)}, \quad (4.10)$$

$$\beta_n = 0.125 \exp\left(-\frac{\psi_m + 65}{80}\right), \quad (4.11)$$

$$\alpha_m = \frac{0.1 (\psi_m + 40)}{1 - \exp\left(-\frac{\psi_m + 40}{10}\right)}, \quad (4.12)$$

$$\beta_m = 4 \exp\left(-\frac{\psi_m + 65}{18}\right), \quad (4.13)$$

$$\alpha_h = 0.07 \exp\left(-\frac{\psi_m + 65}{20}\right), \quad (4.14)$$

$$\beta_h = \frac{1}{1 + \exp\left(-\frac{\psi_m + 35}{10}\right)}. \quad (4.15)$$

The parameters, α and β, in equations (4.10) and (4.15) are derived via experiment on the squid giant axon and are explicitly stated by Dayan and Abbott (2001) and are strictly functions of the membrane potential. The definitions of equations (4.10) – (4.15) are originally given in Hodgkin and Huxley (1952b). However, the parameters defined by Dayan and Abbott (2001) solve directly for the membrane potential while the definitions for equations (4.10) – (4.15) given in Hodgkin and Huxley (1952b) solve for a displacement from a reference potential. The rest potential of a neuron, which is the membrane potential present when no external electrical stimuli is applied to it, is between $-70mV$ and $-60mV$ (Kandel

et al., 2012). The parameters, n, m, and h in equation (4.6) are dimensionless parameters that range between 0 and 1 (Hodgkin and Huxley, 1952b). The values for various constants used in equations (4.6) – (4.15) are shown in Table 4.1.

Hodgkin-Huxley Equation Parameters			
Max. Conductance/ unit area, K^+:[4]	$\bar{g}_k =$	36	$[mS/cm^2]$
Max. Conductance/ unit area, Na^+:[4]	$\bar{g}_{Na} =$	120	$[mS/cm^2]$
Max. Conductance/ unit area, l:[4]	$\bar{g}_l =$	0.3	$[mS/cm^2]$
Capacitance/ unit area:[4]	$C_m =$	1	$[\mu F/cm^2]$
Reversal Potential, K^+:[1]	$\bar{N}_k =$	-77	$[mV]$
Reversal Potential, Na^+:[1]	$\bar{N}_{Na} =$	50	$[mV]$
Reversal Potential, l:[1]	$\bar{N}_l =$	-54.387	$[mV]$
Initial Voltage at $t = 0$:[2]	$V_0 =$	-60	$[mV]$
Initial value of n at $t = 0$:[3]	$n_0 =$	0.3208	$[-]$
Initial value of m at $t = 0$:[3]	$m_0 =$	0.0513	$[-]$
Initial value of h at $t = 0$: [3]	$h_0 =$	0.5841	$[-]$
Applied Current Intensity:[3]	$I_0/A =$	0.1	$[A/m^2]$
Current Duration:[3]	$t_{Duration} =$	2×10^{-3}	$[s]$
Action Potential Delay:[3]	$t_{Delay} =$	10^{-2}	$[s]$

Table 4.1. Values used for the various parameters in equation (4.6). The values associated with [1] were taken from Dayan and Abbott (2001); values associated with [2] were taken from Kandel et al. (2012); values associated with [3] were taken from Medvedev (2005); values associated with [4] were taken from Hodgkin and Huxley (1952b).

The values $\bar{N}_i;\ i = k,\ Na,\ l$ are reversal potentials for each ion species because when $\psi_m > \bar{N}_i;\ i = k,\ Na,\ l$, the current associated with that ion species changes sign (Dayan and Abbott, 2001). The reversal potentials for the K^+ and Na^+ channels stem directly from the Nernst equation, equation (4.1), while the reversal potential of the leak channel was chosen to make the total ionic current zero at the resting membrane potential of a typical neuron (Dayan and Abbott, 2001; Hodgkin and Huxley, 1952b). It should be noted that as the ion channels in neurons open and close, the concentrations of ion species extracellular and intracellular to the neurons change. This leads to changes in N_α according to equation (4.1). The reversal potentials, $\bar{N}_i;\ i = k,\ Na,\ l$, chosen in Table 4.1 are sufficient though for approximating the membrane potential in equation (4.6) such that equation (4.6) can now be written as (Dayan and Abbott, 2001; Ermentrout and Terman, 2010; Hodgkin and Huxley, 1952b):

$$C_m \frac{d\psi_m}{dt} - \frac{I_0}{A} + \bar{g}_k n^4 \left(\psi_m - \bar{N}_k\right) + \bar{g}_{Na} m^3 h \left(\psi_m - \bar{N}_{Na}\right) + \bar{g}_l \left(\psi_m - \bar{N}_l\right) = 0. \quad (4.16)$$

The parameters given in Table 4.1 can be coupled with equation (4.16) and equations (4.7) – (4.15) in order to solve for the membrane potential, ψ_m. These equations cannot be solved analytically, so a numeric solver is employed to obtain values of ψ_m for all time $t \geq 0$.

4.4 Numerical Solution

Using code adapted from Medvedev (2005) a solution to equation (4.16) and equations (4.7) – (4.9) can be obtained. For the purposes of finding a suitable membrane potential for use in the governing equations listed in section 2.6, external stimuli of $I_0 = 10\mu A/cm^2$ will be applied for a period of $2ms$ every $t = 5ms$. Using the parameters in Table 4.1, a "train" of action potentials can be generated and the membrane voltage over time is shown in Figure 4.3.

The membrane potential in Figure 4.3 is the standard form of an action potential (Kandel et al., 2012). The membrane voltage, ψ_m can be used directly in the governing equations for the triphasic model to determine chemo-mechanical properties of the tissue such as ionic concentrations or deformation behaviors.

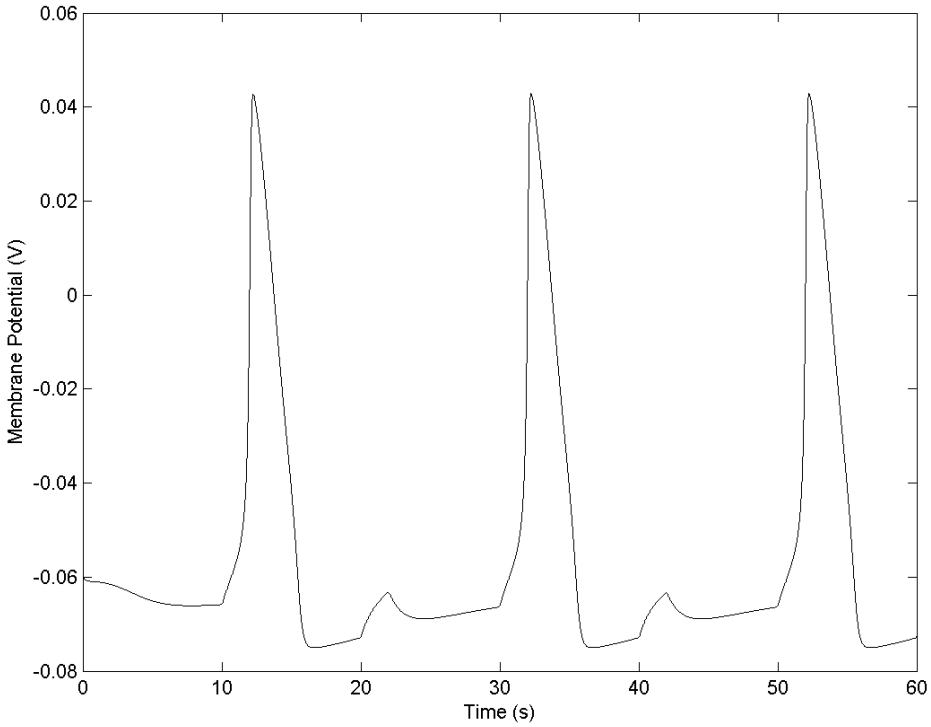

Figure 4.3. Membrane voltage obtained by solving the equations of the Hodgkin-Huxley model. The code used to obtain this particular solution was adapted from Medvedev (2005).

Chapter 5
Linearization of the Governing Equations

5.1 Balance of Momentum

The governing equations for the balance of mass and balance of momentum can be simplified to a *one-dimensional* case so that the balance laws, combined with the Hodgkin-Huxley model for the membrane potential, can be linearized. Creating one-dimensional versions of the governing equations given in section 2.6 are necessary in order to obtain an easily implementable solution for the concentrations of the ions and membrane displacement. The infinitesimal strain tensor, $\boldsymbol{\varepsilon}$, can be defined in Cartesian coordinates as:

$$\boldsymbol{\varepsilon} = \sum_{i,j=1}^{3} \varepsilon_{ij} \boldsymbol{e_i} \otimes \boldsymbol{e_j}, \tag{5.1}$$

where x corresponds to index 1, y corresponds to index 2, and z corresponds to index 3. Also, $\boldsymbol{e_i}$ defines a unit vector in the i^{th}-direction. In one-dimension, the components of the strain tensor can be represented in matrix form as:

$$[\boldsymbol{\varepsilon}] = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \varepsilon_{zz} \end{pmatrix}, \tag{5.2}$$

so the dilatation, $\mathrm{tr}(\boldsymbol{\varepsilon}) = \sum_{i=1}^{3} \varepsilon_{ii}$, is equal to the normal strain in the z-direction:

$$e = \mathrm{tr}(\boldsymbol{\varepsilon}) = \varepsilon_{zz}. \tag{5.3}$$

An equivalent expression of the one-dimensional strain tensor can be defined as:

$$\boldsymbol{\varepsilon} = \mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}, \tag{5.4}$$

which leads to the conclusion that:

$$\boldsymbol{\varepsilon} = e\boldsymbol{I}. \tag{5.5}$$

The chemical expansion stress, T_c, will be ignored (Gu et al., 1999; Lu et al., 2010; Sun et al., 1999). Equation (2.35) reduces to:

$$\left(H_a \frac{\partial}{\partial z}\left(e - p\right)\right)\boldsymbol{e_z} = \mathbf{0}, \tag{5.6}$$

$$H_a = \lambda_s + 2\mu_s. \tag{5.7}$$

The constant H_a was originally defined in Lu et al. (2010). Taking the divergence of equation (5.6) results in:

$$H_a \frac{\partial^2 e}{\partial z^2} - \frac{\partial^2 p}{\partial z^2} = 0, \tag{5.8}$$

which matches the result obtained by Lu et al. (2010).

5.2 Balance of Mass for the Fluid Phase

The balance of mass for the fluid phase is given in equation (2.36). There is an assumption made by Lu et al. (2010) that the osmotic coefficients for all of the ion species are: $\Phi_\alpha = 1 \;\forall\; \alpha = 1,\ 2,\ \ldots,\ n$. This assumption results in a one-dimensional form of the balance of mass:

$$\frac{\partial v^s}{\partial z} - k_0 \left\{ \frac{\partial^2 p}{\partial z^2} + \Xi_w \frac{\partial^2 e}{\partial z^2} - \omega \mathrm{F_c} \left(\frac{\partial c^F}{\partial z}\frac{\partial \psi}{\partial z} + c^F \frac{\partial^2 \psi}{\partial z^2} \right) \right\} = 0. \tag{5.9}$$

The hydrostatic fluid pressure, p, can be eliminated using equation (5.6):

$$\frac{\partial v^s}{\partial z} - k_0 \left(H_a + \Xi_w\right) \frac{\partial^2 e}{\partial z^2} + \omega \mathrm{F_c} k_0 \frac{\partial}{\partial z}\left(c^F \frac{\partial \psi}{\partial z}\right) = 0. \tag{5.10}$$

The quantity $\partial v^s/\partial z$ is related to the dilatation by equations (2.7) and (2.12) as:

$$\frac{\partial \phi^s}{\partial t} + \operatorname{div}\left(\phi^s \boldsymbol{v^s}\right) = \frac{\partial \phi^s}{\partial t} + \operatorname{grad} \phi^s \cdot \boldsymbol{v^s} + \phi^s \operatorname{div} \boldsymbol{v^s} = 0. \tag{5.11}$$

Second order terms such as $\operatorname{grad}\left(\phi^s\right) \cdot \boldsymbol{v^s}$ can be omitted in the linearization process (Lu et al., 2010) which results in:

$$\operatorname{div} \boldsymbol{v^s} = -\frac{1}{\phi^s} \frac{\partial \phi^s}{\partial t}. \tag{5.12}$$

Combining equation (2.12) with equation (5.12) gives:

$$\operatorname{div} \boldsymbol{v^s} = \frac{1}{1-e} \frac{\partial e}{\partial t}, \tag{5.13}$$

where the nonlinear term $1/\left(1-e\right)$ can be represented as the first two terms of a Maclaurin series:

$$\frac{1}{1-e} \cong 1 + e \text{ for } e \ll 1,$$

applying this approximation to equation (5.14) yields:

$$\operatorname{div} \boldsymbol{v^s} \cong \frac{\partial e}{\partial t} + e \frac{\partial e}{\partial t}. \tag{5.14}$$

By eliminating the second order term $e\partial e/\partial t$, a linear expression for the divergence of the solid phase velocity can be obtained as:

$$\operatorname{div} \boldsymbol{v^s} \cong \frac{\partial e}{\partial t}. \tag{5.15}$$

In one dimension, $\operatorname{div} \boldsymbol{w} = \partial\left(w_z\right)/\partial z$. So an expression for the one-dimensional divergence of the solid phase velocity can be obtained as:

$$\frac{\partial v^s}{\partial z} = \frac{\partial e}{\partial t}. \tag{5.16}$$

Using equation (5.16), the governing equation for the balance of mass of the mixture can be stated as:

$$(H_a + \Xi_w)\frac{\partial^2 e}{\partial z^2} + \frac{1}{k_0}\frac{\partial e}{\partial t} - \omega \mathrm{F_c}\frac{\partial}{\partial z}\left(c^F \frac{\partial \psi}{\partial z}\right) = 0. \tag{5.17}$$

5.3 Balance of Mass for the Ionic Phase

Equation (2.9) can be expanded as:

$$\phi^w \frac{\partial c_\alpha}{\partial t} + c_\alpha \frac{\partial \phi^w}{\partial t} + c_\alpha \operatorname{div} \phi^w \boldsymbol{v^\alpha} + \phi^w \boldsymbol{v^\alpha} \cdot \operatorname{grad} c_\alpha = 0. \tag{5.18}$$

which can be combined with equation (2.8) and multiplied with c_α to result in:

$$c_\alpha \frac{\partial \phi^w}{\partial t} + c_\alpha \operatorname{div} \phi^w \boldsymbol{v^w} = 0, \tag{5.19}$$

to form an equivalent form of equation (2.9) as:

$$\phi^w \frac{\partial c_\alpha}{\partial t} + \operatorname{div}\left(c_\alpha \phi^w \boldsymbol{v^\alpha} - c_\alpha \phi^w \boldsymbol{v^w}\right) + \phi^w \boldsymbol{v^w} \cdot \operatorname{grad} c_\alpha = 0. \tag{5.20}$$

By ignoring the higher order term, $\phi^w \boldsymbol{v^w} \cdot \operatorname{grad} c_\alpha$, and employing equation (2.15) and equation (2.16), the governing equation of the balance of mass of the ionic phase becomes:

$$\phi^w \frac{\partial c_\alpha}{\partial t} + \operatorname{div}\left(\boldsymbol{J_\alpha} - c_\alpha \boldsymbol{J_w}\right) = 0. \tag{5.21}$$

Combining this relation with equation (2.33) and noting that D_α is assumed to be constant results in:

$$\phi^w \frac{\partial c_\alpha}{\partial t} - D_\alpha \operatorname{div}\left(\phi^w \operatorname{grad}(c_\alpha) + \phi^w c_\alpha \left(\frac{z_\alpha \mathrm{F_c}}{RT}\right) \operatorname{grad}(\psi)\right) = 0. \tag{5.22}$$

This is the full form of the combined governing equations for the fluid and ionic balance of masses. The one-dimensional form of equation (5.22) is:

$$\phi^w \frac{\partial c_\alpha}{\partial t} - D_\alpha \frac{\partial}{\partial z}\left\{\phi^w \frac{\partial c_\alpha}{\partial z} + \phi^w c_\alpha \left(\frac{z_\alpha \mathrm{F_c}}{RT}\right) \frac{\partial \psi}{\partial z}\right\} = 0, \tag{5.23}$$

which can be further simplified by separating the portions that depend on ϕ^w and

the portions that depend on $\partial\phi^w/\partial z$ as:

$$\phi^w\left\{\frac{1}{D_\alpha}\frac{\partial c_\alpha}{\partial t}-\frac{\partial^2 c_\alpha}{\partial z^2}-\frac{z_\alpha \mathrm{F_c}}{RT}\frac{\partial}{\partial z}\left(c_\alpha\frac{\partial\psi}{\partial z}\right)\right\}$$
$$-\frac{\partial\phi^w}{\partial z}\left\{\frac{\partial\phi^w}{\partial z}\frac{\partial c_\alpha}{\partial z}+\frac{z_\alpha \mathrm{F_c}}{RT}c_\alpha\frac{\partial\psi}{\partial z}\right\}=0. \tag{5.24}$$

Since the variations in $\partial\phi^w/\partial z$ depend on the infinitesimal variations of the dilatation of the tissue, the terms associated with $\partial\phi^w/\partial z$ can be ignored in this linearization procedure. This means that the one dimensional governing equation equation (5.22) can be simplified to:

$$\frac{\partial^2 c_\alpha}{\partial z^2}-\frac{1}{D_\alpha}\frac{\partial c_\alpha}{\partial t}+\frac{z_\alpha \mathrm{F_c}}{RT}\frac{\partial}{\partial z}\left(c_\alpha\frac{\partial\psi}{\partial z}\right)=0. \tag{5.25}$$

It should be noted that a term associated with $c_\alpha\partial\psi/\partial z$ has not been eliminated which creates a nonlinear term in z. This inconsistency in the linearization will be eliminated because $\partial\psi/\partial z$ is only a function of t when combined with the membrane potential of the neuron calculated in chapter 4. The fact that the $c_\alpha\partial\psi/\partial z$ term is linear in z is only a result of the particular way in which ψ was calculated. It should also be noted that equation (5.25) is the same equation obtained by Weiss (1996)[1] using a different approach in deriving it.

[1]The equation is found on page 471

Chapter 6
Brain Chemo-Mechanics

6.1 Solution Domain

The domain of the problem will be defined over $z \in [0, \ell]$ which is shown in Figure 6.1. This chosen domain covers the intracellular space of the axon which is a sub-domain of the entire positive real z-axis, but the values of quantities in the extracellular space must be considered in deriving the values of parameters used in this analysis. The membrane that divides the intracellular from the extracellular sides of the neuron is located between $\ell < z < \ell + h$ with a thickness of h. It should be noted that the chosen problem domain does not cover ion concentrations inside the space $\ell < z < \ell + h$ which make up the membrane, but the displacement of the membrane will be modeled using the concentration of the ions at $z = \ell$.

The area shown in Figure 6.1 can be split into three separate domains titled: Intracellular, Membrane, and Extracellular. These domains can be defined as:

- Intracellular: $\{z \in \mathbb{R} \mid 0 \leq z \leq \ell\}$
- Membrane: $\{z \in \mathbb{R} \mid \ell < z \leq \ell + h\}$
- Extracellular: $\{z \in \mathbb{R} \mid z > \ell\}$

Note that the Intracellular and Membrane domains are the domains which will be modeled by the equations developed in chapters 2 and 5. The electric potential ψ need to be defined over: $\{z \in \mathbb{R} \mid z \geq 0\}$. However, only the membrane potential, ψ_m, is known for $t \geq 0$ through solving the Hodgkin-Huxley equations in section 4.4 (Dayan and Abbott, 2001; Hodgkin and Huxley, 1952a), not ψ. So ψ will need

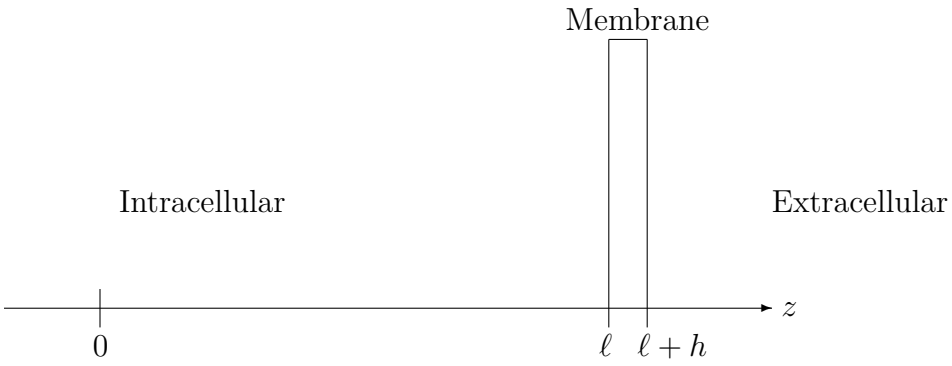

Figure 6.1. Simple drawing of the Intracellular, Membrane, and Extracellular domains on the z-axis (note that the boundary-value problem will be solved only for the Intracellular domain)

to be defined in terms of $\psi_{in},\ \psi_{ex},$ and ψ_m. The value of ψ takes on the following form:

$$\psi = \begin{cases} \psi_{in}; & 0 \leq z < \ell \\ \psi_m; & \ell \leq z < \ell + h \\ \psi_{ex}; & z \geq \ell + h \end{cases} . \tag{6.1}$$

Applying equation (4.4) to equation (6.1) results in:

$$\psi = \begin{cases} \psi_m + \psi_{ex}; & 0 \leq z < \ell \\ \psi_m; & \ell \leq z < \ell + h \\ \psi_{ex}; & z \geq \ell + h \end{cases} . \tag{6.2}$$

It will be assumed that the potential of the extracellular space varies minimally with respect to position: $\partial\psi_{ex}/\partial z \ll 1$. Taking the first derivative with respect to z of equation (6.2) results in:

$$\frac{\partial \psi}{\partial z} \cong \begin{cases} \frac{\partial \psi}{\partial z}\,|_{z=\ell}\ ; & 0 \leq z < \ell + h \\ 0 & z \geq \ell + h \end{cases} . \tag{6.3}$$

The quantity $\partial\psi/\partial z\,|_{z=\ell}$ can be approximated using the definition of the first

derivative about $z = \ell$ and equation (4.4) as:

$$\begin{aligned}\frac{\partial \psi_m}{\partial z} &= \lim_{h\to 0}\left\{\frac{\psi(\ell+h)-\psi(\ell)}{\ell+h-\ell}\right\},\\ \frac{\partial \psi_m}{\partial z} &= \lim_{h\to 0}\left\{\frac{\psi_{ex}-\psi_{in}}{h}\right\},\\ \frac{\partial \psi_m}{\partial z} &= -\lim_{h\to 0}\left\{\frac{\psi_m}{h}\right\},\end{aligned} \tag{6.4}$$

if h is sufficiently small ($h \ll 1$) then:

$$\frac{\partial \psi_m}{\partial z} \cong -\frac{\psi_m}{h}. \tag{6.5}$$

Using this approximation, the derivative of the electric potential with respect to z can be represented as:

$$\frac{\partial \psi}{\partial z} \cong \begin{cases} -\frac{\psi_m}{h}; & 0 \le z < \ell + h \\ 0 & z \ge \ell + h \end{cases}. \tag{6.6}$$

Thus equation (5.25) can be represented in the Intracellular Domain as:

$$\frac{\partial^2 c_\alpha}{\partial z^2} - \frac{1}{D_\alpha}\frac{\partial c_\alpha}{\partial t} - \left(\frac{z_\alpha \mathrm{F_c}}{RT}\frac{\psi_m}{h}\right)\frac{\partial c_\alpha}{\partial z} = 0. \tag{6.7}$$

Note that $\partial^2\psi/\partial z^2 = 0; \quad 0 \le z \le \ell$ since $\psi_m = \psi_m(t)$ according to equation (4.16). Therefore all terms in equation (6.7) are linear with respect to z.

6.2 Ionic Concentration Boundary/Initial Conditions

Equation (6.7) needs two boundary conditions and one initial condition to be solved. Since the domain only includes the intracellular portion of the neuron, then only the concentrations of ions located inside the neuron need to be considered at $t = 0$. The concentration of the ions in the Intracellular domain at $t = 0$ are:

$$c_\alpha(z, 0) = (c_0)_\alpha, \tag{6.8}$$

for $\alpha = 1,\ 2,\ \ldots,\ n$ where $(c_0)_\alpha$ is constant. At distances sufficiently far enough from the membrane, the concentration of ions should not vary over time. The

movement of ions inside the tissue should occur mainly near the membrane. So at $z = 0$, it will be assumed that the concentration of ions will remain constant at:

$$c_\alpha(0,t) = (c_0)_\alpha . \tag{6.9}$$

Note that at $z = \ell$, the movement of ions is significant. The boundary conditions at this point may be "jump" conditions since the concentrations of ions on either side of the membrane may be significantly different. As a first approximation for this analysis, it will be assumed that the net flux of each ionic species, $\alpha = 1,\ 2,\ \ldots,\ n$, through the membrane is equal to zero:

$$\frac{\partial c_\alpha}{\partial z}(\ell,t) + \frac{\partial c_\alpha}{\partial z}(\ell + h, t) = 0.$$

For membranes that are sufficiently thin ($h \ll 1$):

$$\frac{\partial c_\alpha}{\partial z}(\ell,t) \cong \frac{\partial c_\alpha}{\partial z}(\ell + h, t) .$$

Therefore the boundary condition at $z = \ell$ is defined as:

$$\frac{\partial c_\alpha}{\partial z}(\ell,t) = 0. \tag{6.10}$$

This boundary condition is at odds with the statement that the movement of ions at $z = \ell$ is significant. The reason that this boundary condition is being considered is so that a numerical solution can be easily implemented without the need to develop a model of ion movement at the membrane.

6.3 Membrane Displacement Conditions

The membrane motion is governed by equation (5.17). In the theory of linear elasticity, the displacement is related to the infinitesimal strain by (Gurtin et al., 2010):

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left(\operatorname{grad}\boldsymbol{u} + (\operatorname{grad}\boldsymbol{u})^{\mathrm{T}}\right). \tag{6.11}$$

In one dimension, the only change in displacement that is significant is the displacement in the z-direction. So the components of the displacement gradient can be represented in matrix form as:

$$[\operatorname{grad} \boldsymbol{u}] = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{\partial u_z}{\partial z} \end{pmatrix}. \tag{6.12}$$

Using equation (5.3), a link between the displacement gradient and the dilatation can be established:

$$e = \varepsilon_{zz} = \frac{\partial u_z}{\partial z}. \tag{6.13}$$

Combining this result with equation (5.17) gives:

$$(H_a + \Xi_w) \frac{\partial^2}{\partial z^2} \left(\frac{\partial u_z}{\partial z} \right) + \frac{1}{k_0} \frac{\partial}{\partial t} \left(\frac{\partial u_z}{\partial z} \right) - \omega \mathrm{F_c} \frac{\partial}{\partial z} \left(c^F \frac{\partial \psi}{\partial z} \right) = 0. \tag{6.14}$$

The third order derivative in displacement will be ignored since the membrane is thin and its displacement is assumed to be small ($\operatorname{grad} \boldsymbol{u} \ll 1$). Equation (5.17) can now be approximated as:

$$\frac{1}{k_0} \frac{\partial}{\partial t} \left(\frac{\partial u_z}{\partial z} \right) - \omega \mathrm{F_c} \frac{\partial}{\partial z} \left(c^F \frac{\partial \psi}{\partial z} \right) = 0. \tag{6.15}$$

The approximation of $\partial \psi / \partial z$ in equation (6.6) at $z = \ell$ can be applied to equation (5.17) which results in:

$$\frac{1}{k_0} \frac{\partial}{\partial t} \left(\frac{\partial u_z}{\partial z} \right) + \omega \mathrm{F_c} \frac{\psi_m}{h} \frac{\partial c^F}{\partial z} = 0. \tag{6.16}$$

Assuming that the spatial and time domains of $u_z(z,t)$ are smooth, then:

$$\frac{\partial^2 u_z}{\partial z \partial t} = \frac{\partial^2 u_z}{\partial t \partial z}.$$

Factoring out $\partial / \partial z$ from equation (6.16) and multiplying through by k_0 results in:

$$\frac{\partial}{\partial z} \left(\frac{\partial u_z}{\partial t} + \omega \mathrm{F_c} k_0 \frac{\psi_m}{h} c^F \right) = 0. \tag{6.17}$$

Since $\partial(\cdot)/\partial z = 0$, then the quantity in parenthesis is purely a function of time. Therefore, $\partial u_z / \partial t = du_z / dt$ and equation (6.17) can be represented as:

$$\frac{du_z}{dt} + \omega \mathrm{F_c} k_0 \frac{\psi_m}{h} c^F = \Theta(t). \tag{6.18}$$

The quantity c^F can be calculated from the electroneutrality condition, equation (2.10), using the concentrations of all ions at $z = \ell$. Since c^F and ψ_m are known for $t \geq 0$, the displacement of the membrane in the z-direction at $z = \ell$ can be readily solved after stating an initial condition and form for $\Theta(t)$. The initial condition will be assumed to be:

$$u_z(0) = 0, \tag{6.19}$$

as the membrane should have no change from its original value at $t = 0$. Equation (6.18) can be rewritten as:

$$\frac{du_z}{dt} = \Theta(t) - \omega \mathrm{F_c} k_0 \frac{\psi_m}{h} c^F. \tag{6.20}$$

For simplicity in notation, let:

$$g(t) = -\omega \mathrm{F_c} k_0 \frac{\psi_m}{h} c^F. \tag{6.21}$$

Previous research by Tasaki and Iwasa (1981); Yao et al. (2003) have shown that the membrane displacement and electric potential of the neuron are loosely related to one another. This means that the membrane displacement can be assumed to have some periodic form like the electric potential. So u_z can be represented as some Fourier series, and the value of $\Theta(t)$ needs to be chosen to reflect some periodic behavior. Using a Fourier series expansion of $g(t)$, equation (6.20) can be written as:

$$\begin{aligned} \frac{du_z}{dt} &= \Theta(t) + g(t), \\ \frac{du_z}{dt} &= \Theta(t) + a_0 + \sum_{n=1}^{\infty} \left(a_n \cos(\omega t) + b_n \sin(\omega t)\right). \end{aligned} \tag{6.22}$$

Integrating this result gives an expression for $u_z(t)$ as:

$$u_z(t) = \int_0^t \Theta\left(\tilde{t}\right) d\tilde{t} + a_0 t + \frac{1}{\omega} \sum_{n=1}^{\infty} \left(a_n \sin(\omega t) - b_n \cos(\omega t)\right). \tag{6.23}$$

This displacement, u_z, can be represented as its own unique Fourier series:

$$u_z(t) = \hat{a}_0 + \sum_{n=1}^{\infty} \left(\hat{a}_n \cos(\omega t) + \hat{b}_n \sin(\omega t) \right). \tag{6.24}$$

The quantity $\Theta(t)$ must be able to resolve equation (6.23) into equation (6.24). A possible representation of $\Theta(t)$ can be:

$$\begin{aligned} \Theta(t) &= \Gamma + mg(t), \\ \Theta(t) &= \Gamma + m \left(\sum_{n=1}^{\infty} a_n \cos(\omega t) + b_n \sin(\omega t) \right), \end{aligned} \tag{6.25}$$

where Γ and m are both constants. Equations (6.23) and (6.24) can be equated using the definition of $\Theta(t)$ given in equation (6.25):

$$\begin{aligned} \int_0^t \Theta\left(\tilde{t}\right) d\tilde{t} + a_0 t &= \hat{a}_0, \\ \Gamma t + m a_0 t + a_0 t + (m+1) \int_0^t \left(\sum_{n=1}^{\infty} a_n \cos\left(\omega \tilde{t}\right) + b_n \sin\left(\omega \tilde{t}\right) \right) d\tilde{t} &= \\ \hat{a}_0 + \sum_{n=1}^{\infty} \left(\hat{a}_n \cos(\omega t) + \hat{b}_n \sin(\omega t) \right), & \\ \Gamma t + m a_0 t + a_0 t + (m+1) \sum_{n=1}^{\infty} \left(\frac{a_n}{\omega} \sin(\omega t) - \frac{b_n}{\omega} \sin(\omega t) - \frac{b_n}{\omega} \right) &= \\ \hat{a}_0 + \sum_{n=1}^{\infty} \left(\hat{a}_n \cos(\omega t) + \hat{b}_n \sin(\omega t) \right). & \end{aligned}$$

In order to eliminate the linear term, $a_0 t$, in equation (6.23), the value of Γ needs to solve the following equation for all time t:

$$\begin{aligned} \Gamma t + (m+1) a_0 t - (m+1) \sum_{n=0}^{\infty} \frac{b_n}{\omega} &= \hat{a}_0, \\ (\Gamma + (m+1) a_0) t - (m+1) \sum_{n=0}^{\infty} \frac{b_n}{\omega} &= \hat{a}_0. \end{aligned}$$

For the following relation to be valid for all time t:

$$\Gamma = -(m+1) a_0. \tag{6.26}$$

Combining this relation with equation (6.25) results in one possible definition of $\Theta(t)$ as:

$$\Theta(t) = -(m+1)\,a_0 + mg(t). \tag{6.27}$$

The constant m will be used to scale the value of $u_z(t)$ in order for the calculated value from equation (6.20) to make physical sense. Since m controls the "amplitude" of the displacement, its value was chosen to bring the membrane displacement on the same scale as the one reported by Tasaki and Iwasa (1981). It should be noted that m is *not* a physical parameter, but a mathematical tool since $\Theta(t)$ is an unknown function.

6.4 Numerical Methods

6.4.1 Ionic Concentrations

The Crank-Nicholson method by Smith (1986) was again employed to solve equation (6.7). This method was modified to account for the $\partial c_\alpha/\partial z$ term. A center finite difference of the first-order (Iskandarani, 2010) was employed to discretize $\partial c_\alpha/\partial z$ in order to write equation (6.7) as:

$$\begin{aligned}\frac{c_i^{n+1}-c_i^n}{\Delta t} =& \frac{D}{2(\Delta z)^2}\left(c_{i+1}^{n+1} - 2c_i^{n+1} + c_{i-1}^{n+1} + c_{i+1}^n - 2c_i^n + c_{i-1}^n\right) - \\ & \frac{z\mathrm{F_c}D\psi_m^{n+1}}{4(\Delta z)RTh}\left(c_{i+1}^{n+1} - c_{i-1}^{n+1}\right) - \frac{z\mathrm{F_c}D\psi_m^n}{4(\Delta z)RTh}\left(c_{i+1}^n - c_{i-1}^n\right).\end{aligned} \tag{6.28}$$

Note that the α subscript was dropped for simplicity in the notation. Also note that z represents the valence of the ionic species and Δz represents the discrete spatial step. The ionic concentration was represented in discreet form as $c_i^n = c_\alpha$ at step z_i in space and step t_n in time. The system in equation (6.28) can be combined into the following:

$$-\theta^{n+1}c_{i+1}^{n+1} + (1+2\lambda)\,c_i^{n+1} - \varphi^{n+1}c_{i-1}^{n+1} = \theta^n c_{i+1}^n + (1-2\lambda)\,c_i^n + \varphi^n c_{i-1}^n, \tag{6.29}$$

where the coefficients θ^n,λ, and φ^n are defined as:

$$\lambda = \frac{D\Delta t}{2\left(\Delta z\right)^2}, \tag{6.30}$$

$$\theta^n = \frac{\Delta t}{2\Delta z}\left(\frac{D}{\Delta z} - \frac{z\mathrm{F_c}D\psi_m^n}{2RTh}\right), \tag{6.31}$$

$$\varphi^n = \frac{\Delta t}{2\Delta z}\left(\frac{D}{\Delta z} + \frac{z\mathrm{F_c}D\psi_m^n}{2RTh}\right). \tag{6.32}$$

Equation (6.29) can be written as a matrix vector system:

$$\left[\boldsymbol{J^{n+1}}\right]\left\{\boldsymbol{c_{mod}^{n+1}}\right\} = \left[\boldsymbol{K^n}\right]\left\{\boldsymbol{c_{mod}^{n}}\right\} + \begin{pmatrix} \varphi^n c_1^n + \varphi^{n+1}c_1^{n+1} \\ 0 \\ \vdots \\ 0 \\ \theta^n c_{N_z}^n \end{pmatrix}, \tag{6.33}$$

where the matrices $[\boldsymbol{J^{n+1}}]$ and $[\boldsymbol{K^n}]$ are tridiagonal matrices similar to the matrices defined in equations (3.25) and (3.26):

$$\left[\boldsymbol{J^{n+1}}\right] = \begin{pmatrix} 1+2\lambda & -\theta^{n+1} & 0 & 0 & \dots & 0 & 0 \\ -\varphi^{n+1} & 1+2\lambda & -\theta^{n+1} & 0 & \dots & 0 & 0 \\ 0 & -\varphi^{n+1} & \ddots & \ddots & & \vdots & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & -\theta^{n+1} & 0 \\ 0 & \dots & & 0 & -\varphi^{n+1} & 1+2\lambda & -\theta^{n+1} \\ 0 & \dots & & 0 & 0 & -\varphi^{n+1} & 1+2\lambda \end{pmatrix}, \tag{6.34}$$

$$\left[\boldsymbol{K^n}\right] = \begin{pmatrix} 1-2\lambda & \theta^n & 0 & 0 & \dots & 0 & 0 \\ \varphi^n & 1-2\lambda & \theta^n & 0 & \dots & 0 & 0 \\ 0 & \varphi^n & \ddots & \ddots & & \vdots & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & \theta^n & 0 \\ 0 & \dots & & 0 & \varphi^n & 1-2\lambda & \theta^n \\ 0 & \dots & & 0 & 0 & \varphi^n & 1-2\lambda \end{pmatrix}. \tag{6.35}$$

The column vectors c_{mod}^{n} and c_{mod}^{n+1} are defined similar to equation (3.24) as:

$$\left\{\boldsymbol{c}_{\boldsymbol{mod}}^{\boldsymbol{n+1}}\right\} = \begin{pmatrix} c_2^{n+1} \\ c_3^{n+1} \\ \vdots \\ c_{N_z-2}^{n+1} \\ c_{N_z-1}^{n+1} \end{pmatrix} \text{ and } \left\{\boldsymbol{c}_{\boldsymbol{mod}}^{\boldsymbol{n}}\right\} = \begin{pmatrix} c_2^{n} \\ c_3^{n} \\ \vdots \\ c_{N_z-2}^{n} \\ c_{N_z-1}^{n} \end{pmatrix}. \tag{6.36}$$

To account for the Neumann boundary condition in equation (6.10), equation (3.29) can be employed to calculate the value of $c_{N_z}^{n+1}$. The matrix in equation (6.34) must be modified in a similar manner as equation (3.31) to account for the boundary condition in equation (6.10) (Harder, 2012):

$$\begin{aligned} J_{N_z,N_z-1}^{n+1} &= -\varphi^{n+1} + \frac{1}{3}\theta^{n+1}, \\ J_{N_z,N_z}^{n+1} &= 1 + 2\lambda - \frac{4}{3}\theta^{n+1}. \end{aligned} \tag{6.37}$$

Equation (6.33) was solved numerically via Crank-Nicholson using the same algorithm in chapter 3.

6.4.2 Membrane Displacement

The displacement, u_z, was calculated from equation (6.20) using the MATLAB built-in function, *ode15s*, to solve the stiff ordinary differential equation (The MathWorks Inc., 2013). Before solving equation (6.20) numerically, the coefficients a_0 and m need to be specified. The coefficient m was chosen as:

$$m = \left(10^{-5}\right) h - 1, \tag{6.38}$$

which rescales the membrane displacement. The coefficient a_0 was calculated as (Kumaresan, 2012):

$$a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} g\left(t\right) dt, \tag{6.39}$$

where T defines the period of integration and t_0 account for the integration beginning at a time $t_0 \neq 0$. Because of the nature of $g\left(t\right)$, a_0 needed to be calculated numerically. In order to attempt to make a_0 as accurate as possible, many separate

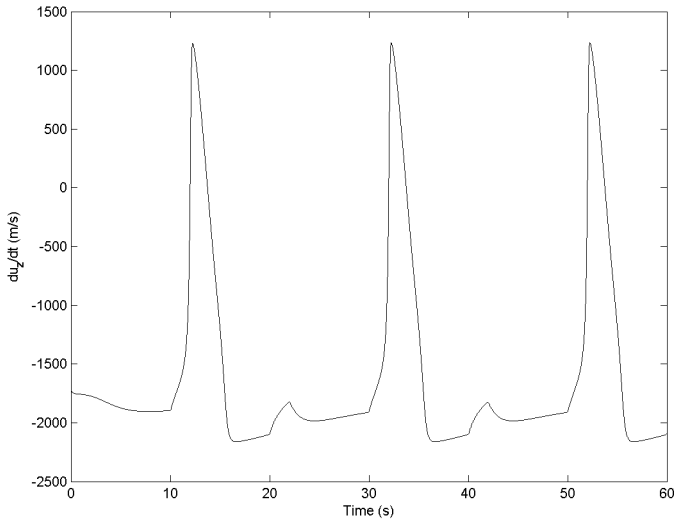

Figure 6.2. Graph of du_z/dt, which is the rate of change of the membrane displacement with respect to time

values of a_0 were calculated over different periods. The periods, T^k, where k designates the $1^{st}, 2^{nd}, 3^{rd}, \ldots$ local maxima, in which $g(t) > 0$. These local maxima can be seen in Figure 6.2, and code was implemented to calculate the precise locations of these local maxima. The code is given in Appendix A. The time t_0^k define the location of the k^{th} local maxima and T^k defines the difference between t_0^{k+1} and t_0^k or more directly: $T^k = t_0^{k+1} - t_0^k$.

Each value of a_0^k was calculated using equation (6.39) for each t_0^k and T^k using the *trapz* numerical integration function in MATLAB (The MathWorks Inc., 2013) and averaged over the number of local maxima minus one. The minus one accounts for the fact that no more values of a_0^k are calculated when the final local maxima is reached. So the practical value of a_0 that is used in numerically solving equation (6.20) can be calculated as:

$$a_0 = \frac{1}{N_{max} - 1} \sum_{k=1}^{N_{max}-1} \left(\int_{t_0^k}^{t_0^k + T^k} g(t)\, dt \right), \tag{6.40}$$

where N_{max} is the number of local maxima and $k = 1, 2, \ldots N_{max}$.

Parameters used in the Numerical Analysis			
Domain Parameters			
Length of the Domain:	$\ell =$	10^{-6}	$[m]$
Initial Time:	$t_0 =$	0	$[s]$
Final Time:	$t_f =$	60	$[s]$
Temperature:[1]	$T =$	25	$[^\circ\mathrm{C}]$
Universal Gas Constant:[4]	$R =$	8.314472	$[J/(molK)]$
Faraday's Constant:[4]	$\mathrm{F_c} =$	96485.3383	$[C/mol]$
Spatial Grid Size:	$N_z =$	100	$[-]$
Time Grid Size:	$N_t =$	10^5	$[-]$
Spatial Step:	$\Delta z =$	10^{-8}	$[m]$
Time Step:	$\Delta t =$	6×10^{-4}	$[s]$
Membrane Thickness:[1]	$h =$	10^{-8}	$[m]$
Hydraulic Permeability:[3]	$k_0 =$	7.5×10^{-12}	$[m^4/(Ns)]$

Ionic Parameters			
Ion Name:	Valence:	Diffusivity:[1] $[m^2/s]$	Initial Concentration:[2] $[mol/m^3]$
Sodium:	$+1$	1.33×10^{-9}	50
Potassium:	$+1$	1.96×10^{-9}	400
Chlorine:	-1	2.03×10^{-9}	52

Table 6.1. Values of the parameters in equations (6.7) and (6.20) that would typically be found in the brain. Parameters denoted by [1] were taken from Weiss (1996); parameters denoted by [2] were taken from Kandel et al. (2012); parameters denoted by [3] were taken from Basser (1992); parameters denoted by [4] were taken from Lide (2007); all other parameters were defined by the author.

6.5 Numerical Results

In order for the results to make physical sense, the values of various parameters were chosen so that they would be close to what one might find in the brain. In addition to the values described in Table 6.1, the values employed in Table 4.1 were used to calculate ψ_m. The concentrations of the K^+, Na^+, and Cl^- ions along with the concentration of the fixed charge density of the tissue are described in Figures 6.3–6.7.

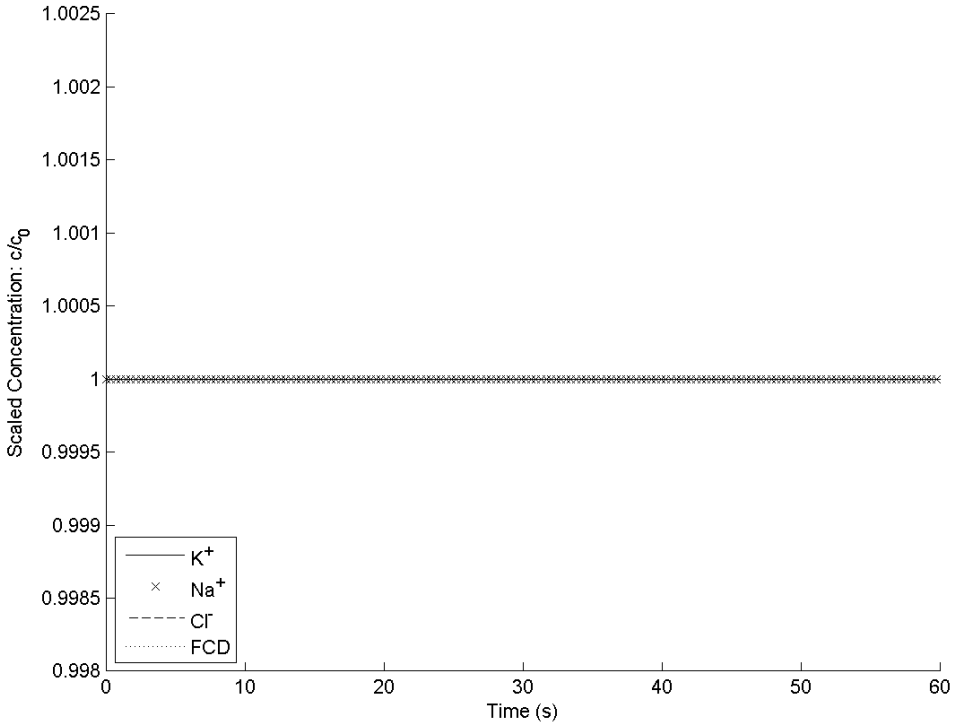

Figure 6.3. Graph of the concentrations of potassium, sodium, chlorine, and the FCD of the axon at $z = 0.15\mu m$ from $t = 0$ to $t = 60$ seconds.

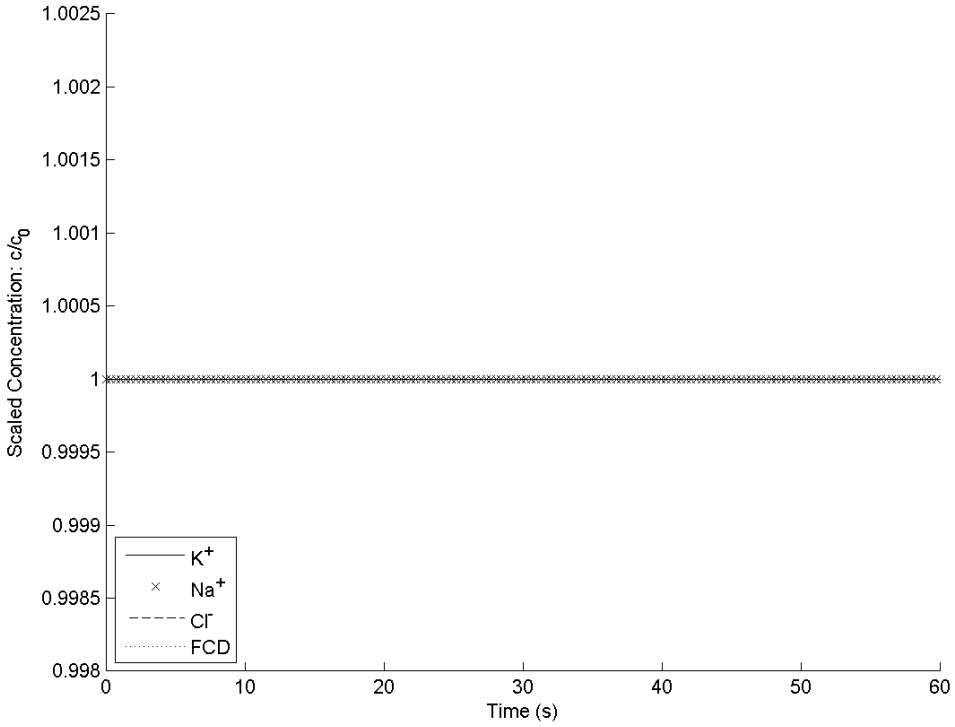

Figure 6.4. Graph of the concentrations of potassium, sodium, chlorine, and the FCD of the axon at $z = 0.40\mu m$ from $t = 0$ to $t = 60$ seconds.

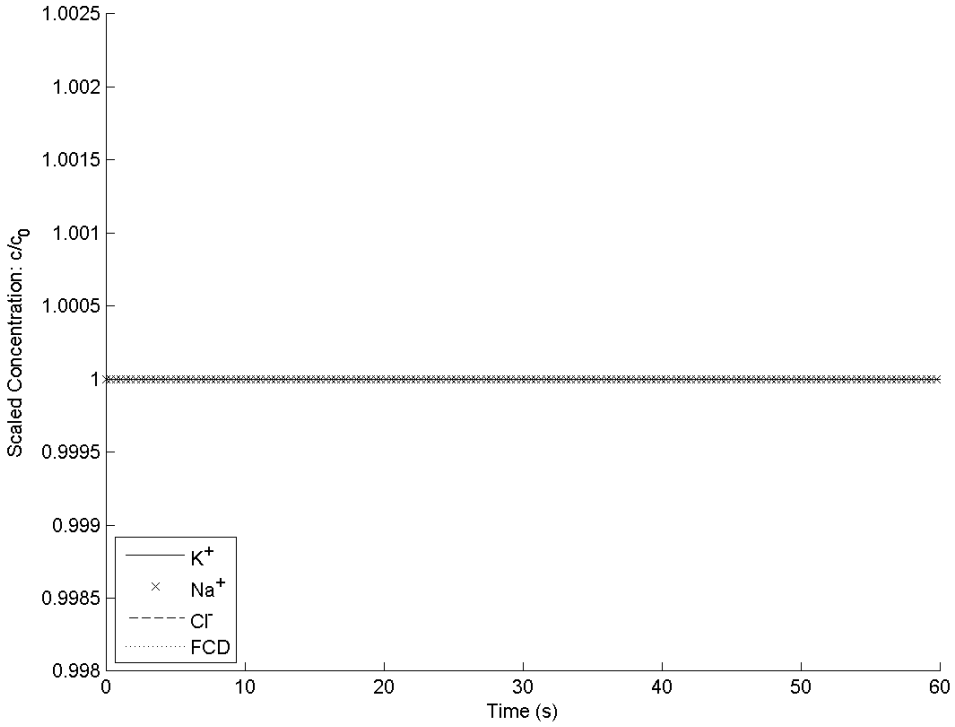

Figure 6.5. Graph of the concentrations of potassium, sodium, chlorine, and the FCD of the axon at $z = 0.50\mu m$ from $t = 0$ to $t = 60$ seconds.

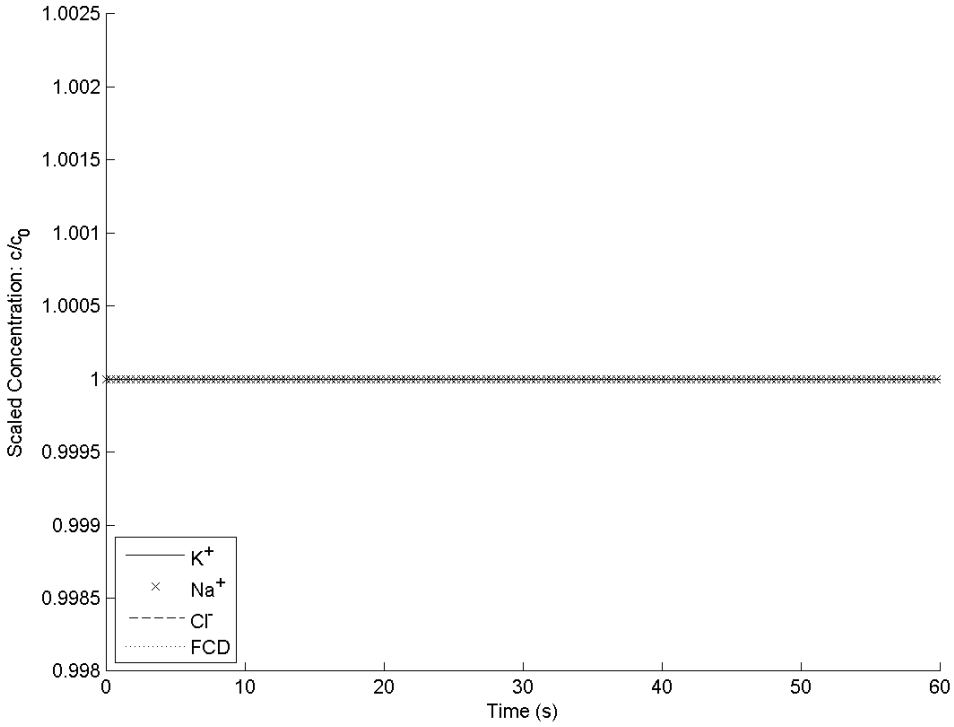

Figure 6.6. Graph of the concentrations of potassium, sodium, chlorine, and the FCD of the axon at $z = 0.75\mu m$ from $t = 0$ to $t = 60$ seconds.

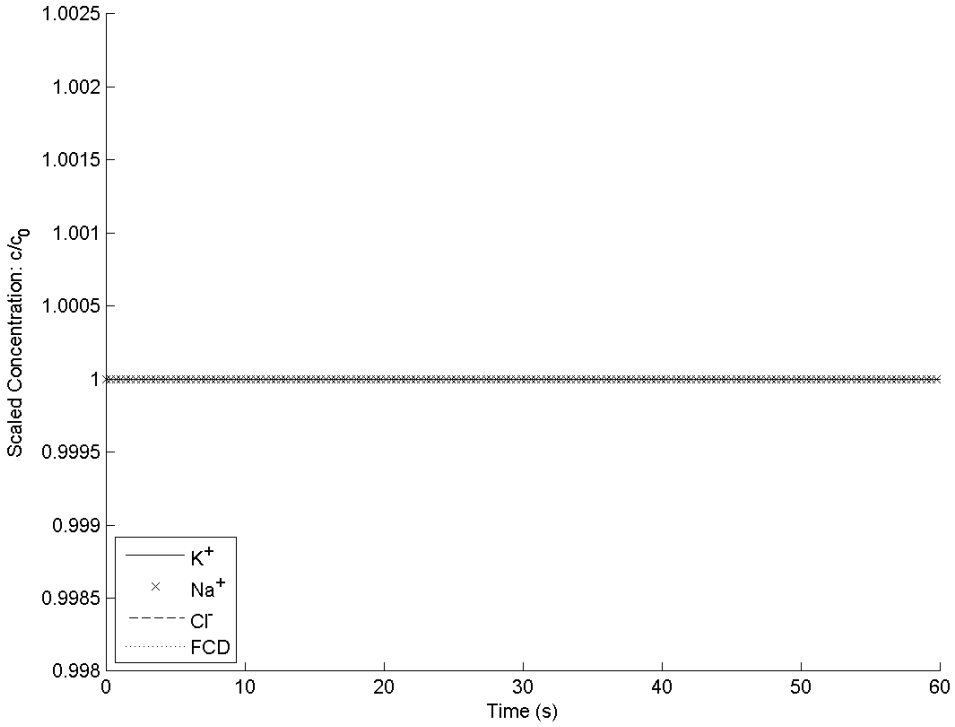

Figure 6.7. Graph of the concentrations of potassium, sodium, chlorine, and the FCD of the axon at $z = 1\mu m = \ell$ from $t = 0$ to $t = 60$ seconds.

The ionic concentration values shown in Figures 6.3–6.7 are scaled according to each ion's initial concentration (and the FCD with the initial FCD concentration at $t = 0$) given in Table 6.1 as:

$$\begin{aligned} c_{scaled} &= \frac{c_\alpha}{(c_0)_\alpha}, \\ c^F_{scaled} &= \frac{c^F}{c^F_0}, \end{aligned} \tag{6.41}$$

for $\alpha = K^+,\ Na^+,\ Cl^-$. All of the lines overlap with one another.

The membrane displacement, u_z, was calculated using the ionic concentration values described at $z = \ell$ which are displayed in Figure 6.7 along with the parameters described in Table 6.1. The values of the membrane displacement are shown in Figure 6.8.

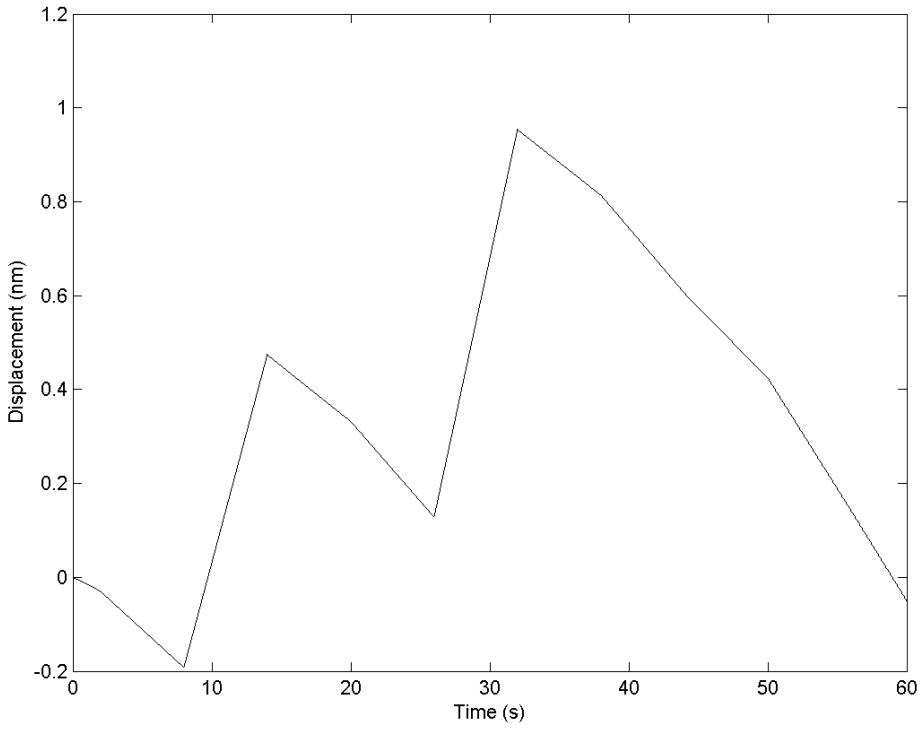

Figure 6.8. Displacement of the axon's membrane, located at $z = \ell$, from $t = 0$ to $t = 60$ seconds. The membrane displacement is scaled in nanometers (nm).

Chapter 7
Discussion

7.1 Concentration

The results displayed in Figures 6.3–6.7 should be an accurate numerical solution to equation (6.7). The code employed in solving equation (3.7) (Lu et al., 2010) was only slightly modified to account for the $\partial c_\alpha/\partial z$ term in equation (6.7). Since the numerical accuracy of the solution to equation (3.7) was verified in chapter 3, then the obtained solutions to equation (6.7) can be expected to accurately describe the changing concentrations of each ionic species.

What can be observed from Figures 6.3–6.7 is that the concentrations of the ions do not change with time at all. Since the FCD can be calculated as a linear combination of all of the ionic concentrations in the mixture according to equation (2.10), the behavior of the FCD over the domain should follow that of the ions themselves. This behavior is, in fact, observed in Figures 6.3–6.7 since the FCD does not change with time at all. The behavior of the FCD can be explained by equation (2.10). Since there are no changes in the concentrations of K^+, Na^+, or Cl^- over time, there is no reason that the FCD of the neuron would change according to equation (2.10). In fact, at any value of z in the Intracellular space, there is no change in the concentration of either K^+, Na^+, or Cl^-. Therefore, it can be said that the concentration: $c_\alpha(z,t)$ remains constant for any value of z or t. This result is expected since the model assumes that the concentration of all of the ions remains constant at $z = 0$. At $z = \ell$, the boundary condition implies that $c_{\ell,t} = \eta(t)$ where $\eta(t)$ is a function of time only. Since the boundary condition described in equation (6.10) specifically states that the ion flux across

the membrane is zero for all ions, no ions flow in or out of the Intracellular space of the neuron. Therefore, the concentration of all of the ions stays constant at the value of their respective initial concentrations which are $(c_0)_\alpha$ for $\alpha = K^+,\ Na^+,\ Cl^-$. This means that the steady-state value of the concentration of all of the ions is $(c_0)_\alpha$ which is already reached at $t = 0$. Therefore, there is no reason at all for the concentrations of any ion to change with time.

The boundary condition described in equation (6.9) and the initial condition described in equation (6.8) are approximations that make physical sense. However, the boundary condition at $z = \ell$ described in equation (6.10) is not physically permissible. Ions need to move through the membrane in order for action potentials to occur (Hodgkin and Huxley, 1952a; Kandel et al., 2012). An indirect measure of how many ions flow across the membrane is the total conductance of all of the ion channels of a specific ion: g_α for $\alpha = K^+,\ Na^+,\ Cl^-$. A large total conductance of an ion correlates to a high amount of that ion species flowing though the membrane (Kandel et al., 2012). Since the membrane has (virtually) zero conductance, it can be reasonably assumed that ion transport only occurs through the various ion channels and pumps in the membrane (Kandel et al., 2012). When those channels (and pumps) are not functioning, it can be reasonably assumed that no ions are flowing across the membrane (Kandel et al., 2012), hence: $\partial c_\alpha / \partial z = 0$.

For this analysis, the lack of ion flux through the membrane will be accepted since the scope of this thesis was to develop a numerical solver for further study of brain biomechanics. For accurate modeling of the flux across the membrane, the true rate of ion flux across the membrane needs to be calculated with respect to time for all ions inside and outside of the neurons. Since no ions are allowed to leave the intracellular space then the ionic concentrations should not change with time, which is evident in Figures 6.3–6.7. The importance of the numerical analysis is to show the inherent relationship of membrane potential and ionic concentration, and Figures 6.3–6.7 shows this relationship. The next logical step in modeling true brain behavior is to perform an analysis with the same parameters in Table 6.1, only with a different boundary condition at $z = \ell$ that accurately models the physics of an action potential. For instance, one might model the boundary condition at $z = \ell$ as:

$$\frac{\partial c_\alpha(\ell, t)}{\partial z} = \zeta_\alpha g_\alpha(t), \tag{7.1}$$

where $g_\alpha(t)$ is the total conductance of all ion channels of ion α at time t and ζ_α is

a proportionality constant associated with ion species α. Ion channel conductances can be readily calculated using equations (4.6) – (4.15), and ζ_α may be chosen in a manner similar to m so that it properly scales g_α to make physical sense of $\partial c_\alpha / \partial z$ at $z = \ell$. Further investigation will be required into finding the most optimal model of ion flux through the membrane during an action potential.

7.2 Displacement

The membrane displacement for $0 \leq t \leq 60$ s is shown in Figure 6.8. The membrane can become positively or negatively displaced where a positive displacement is defined as a "stretching" of the membrane in the positive z direction (swelling) following the definition of the z-axis in Figure 6.1. The membrane displacement does not appear to have a direct correlation to the action potential since the displacement begins to become more positive before the action potential occurs. In fact, the membrane displacement has a negative rate of change with respect to time when the last action potential occurs. This behavior is in stark contrast to the highly correlated behavior between du_z/dt and ψ_m shown in Figure 6.2. A reason for the non-correlated behavior may rest with the method by which $\Theta(t)$ was determined. $\Theta(t)$ was determined by the argument that it needed to prevent any dominant linear behavior in u_z while making sure that the scale of u_z was correct.

The value of $\Theta(t)$ in equation (6.25) is essentially a hypothesis which is used to make equation (6.20) able to accurately model physical changes in the membrane displacement with respect to time. The behavior of the membrane displacement shown in Figure 6.8 can be considered accurate for the electric potential that is being considered since the method of integration is based on a MATLAB built in function (The MathWorks Inc., 2013). Because the numerical methods used to calculate u_z yield no significant errors, it can be said that the membrane displacement displayed in Figure 6.8 is expected to be accurate according to the proposed model and developed numerical method.

Tasaki and Iwasa (1981); Yao et al. (2003) have experimentally measured membrane displacement for different biological materials. Yao et al. (2003) used an optical lever to measure the membrane displacement of Lobster nerve bundles, and Tasaki and Iwasa (1981) measured the membrane displacement of the squid

giant axon using a device that measured rapid pressure changes. There appeared to be no significant correlation between the membrane displacement and electric potential of the tissue in either Tasaki and Iwasa (1981) or Yao et al. (2003). The results in Tasaki and Iwasa (1981); Yao et al. (2003) show that a non-correlation between membrane displacement and electric potential can exist, so it can be said that the membrane displacement in Figure 6.8 is accurate for the parameters given in Table 6.1.

The membrane displacement is related to the ionic concentrations though the FCD and thus related to the concentrations of all of the ions in the neuron through equation (2.10). The membrane displacement results in Figure 6.8 do not truly represent membrane behavior during an action potential since the concentrations of the ions do not change as they should during an action potential. A much better description of the membrane motion during an action potential can be achieved by modeling the ionic concentrations with the boundary condition in equation (7.1). If the solution to equation (6.7) properly accounts for the flux of ions through the membrane in an action potential, then by extension, the solution to equation (6.20) should accurately model the displacement of the membrane during an action potential.

7.3 Conclusion

The numerical solutions to the governing equations, equations (6.7) and (6.14), are accurate for the boundary and initial conditions stated in equations (6.8) – (6.10), equation (6.19), and equation (6.25). Suggested modifications include modeling the ion flux through the membrane using a boundary condition at $z = \ell$ such as the one proposed in equation (7.1).

Equations (6.7) and (6.20) resulted from a linearizion procedure that neglected the effects of certain terms. These terms could have a tremendous impact on the state of the mixture, but were ignored since they were determined to be non-linear and the scope of this thesis was to investigate the effects of governing equations linear in z. Future work should include investigating how the non-linear terms in the governing equations affect the concentration and membrane displacement.

Accurately modeling of normal brain activity must be achieved before this model can be used in cases of abnormal brain activity such as CSD or Traumatic

Brain Injury (Drapaca and Fritz, 2012; Leao, 1944). If this model is accurate in modeling normal brain activity such as the propagation of action potentials, it should be able to produce results similar to Bennett et al. (2008); Chang et al. (2013) for CSD. The developed linear model in this thesis is only a "proof of concept" that the equations of the triphasic model in chapter 2 can accuratly model phenomena in the brain. However, further work on this model will serve to make the model more adaptable to modeling both healthy and diseased tissue in the brain. Accurate modeling of brain activity will yield a better understanding of how the mechanics of brain tissue affect the movement of ions and electrical activity associated with conditions such as CSD and hopefully lead to a better understanding into how the mechanics, chemistry, and electrical activity influence one another in the brain.

Appendix A
MATLAB Code for Chapters 4 and 6

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  FULL DRIVER MATLAB EXECUTABLE FILE FOR SOLVING FOR THE CHEMO-
%  MECHANICS OF THE BRAIN TISSUE SPECIFIED IN THE DOMAIN FILE
%  WITH IONIC DATA SPECIFIED IN THE INDIVIDUAL ION PARAMETER FILES,
%  BOTH OF WHICH ARE DETERMINED BY THE USER
%
%  AUTHOR: BRADFORD JOSEPH LAPSANSKY
%  EMAIL: bradjlap@comcast.net
%
%
clc
clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
fprintf('Begin Driver\n');
tic
DIRECTORY = pwd;
s=0;

%Determine if you want plots: ifPlot = 1 for Plot Creation
ifPlot = 1;

%Points on the nz grid on which you want to plot
plotPoints = [1, 15, 40, 50, 75, 100];
```

```
26
27 %ID Number for the Saved Files:
28 ID = '003';
29
30 %Filenames
31 %For Concentration
32 % domainFile = 'INTRACELLULAR_INPUT_DATA';  %Domain
33 domainFile = strcat('INTRACELLULAR_INPUT_DATA_',ID);  %Domain
34 sodiumFile = 'SODIUM_INPUT';
35 potassiumFile = 'POTASSIUM_INPUT';
36 chlorineFile = 'CHLORINE_INPUT';
37
38 %Check if the Domain Exists; if not, create it:
39 cd('Files')
40 domainExist = exist(strcat(domainFile,'.mat'),'file');
41
42 if domainExist == 2
43     fprintf('\nDomain Exists\n');
44 else
45     cd(DIRECTORY);
46     %Name the Type of Domain
47     DOMAIN_DATA_DRIVER()
48     cd('Files')
49 end
50
51
52 %For Displacement
53 %Names of the files that will be used
54 NA_NAME = strcat('SODIUM_INTRACELLULAR_CONCENTRATION_RESULTS_', ...
       ID);
55 K_NAME ...
       =strcat('POTASSIUM_INTRACELLULAR_CONCENTRATION_RESULTS_', ID);
56 CL_NAME=strcat('CHLORINE_INTRACELLULAR_CONCENTRATION_RESULTS_', ...
       ID);
57
58 %Check the Existence of the Chosen Filenames
59 NaExist = exist(strcat(NA_NAME,'.mat'),'file');
60 KExist = exist(strcat(K_NAME,'.mat'),'file');
61 ClExist = exist(strcat(CL_NAME,'.mat'),'file');
62 cd(DIRECTORY);
63
```

```
64 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
65 %%
66 %Concentration Data
67 s=s+1;
68 fprintf('Sodium Concentration Calculation, Driver Step %1.0f',s);
69 fprintf('\n----------------------------------------------------------\n');
70
71 %Sodium
72 if NaExist == 2 && domainExist == 2
73     fprintf('     Sodium Previously Calculated\n');
74 else
75     CONCENTRATION(sodiumFile,domainFile, ifPlot, plotPoints, ID);
76 end
77
78 s=s+1;
79 fprintf('Potassium Concentration Calculation, Driver Step ...
       %1.0f',s);
80 fprintf('\n----------------------------------------------------------\n');
81 %
82 %Potassium
83 if KExist ==2 && domainExist == 2
84     fprintf('     Potassium Previously Calculated\n');
85 else
86     CONCENTRATION(potassiumFile,domainFile, ifPlot, plotPoints, ...
          ID);
87 end
88 %
89 s=s+1;
90 fprintf('Chlorine Concentration Calculation, Driver Step %1.0f',s);
91 fprintf('\n----------------------------------------------------------\n');
92
93 if ClExist ==2 && domainExist == 2
94     fprintf('     Chlorine Previously Calculated\n');
95 else
96    CONCENTRATION(chlorineFile,domainFile, ifPlot, plotPoints, ID);
97 end
98
99 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
100 %%
101 %  Calculate Displacements:
102 fprintf('\n----------------------------------------------------------');
```

```
fprintf('\nDisplacement Calculation, Driver Step %1.0f', s);
fprintf('\n-----------------------------------------------------------\n');
%Calculate Displacement:
DISPLACEMENT(NA_NAME,K_NAME,CL_NAME,domainFile,ifPlot,ID);

s=s+1;
fprintf('\n-----------------------------------------------------------\n');
fprintf('Plot Concentrations on One Graph, Driver Step %1.0f', s);

%Plot All 3 Ionic Concentrations on One Common Graph:
PLOT_COMBINED(K_NAME, NA_NAME, CL_NAME, domainFile,...
    plotPoints, ID);

fprintf('\n-----------------------------------------------------------\n');
fprintf('End Driver; Time: %6.2f\n', toc);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Domain Information - This program writes a file that contains ...
    all of the
%                           information about the domain and ...
    physical
%                           parameters of the space:
%
%Author: Bradford Lapsansky
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function []=DOMAIN_DATA_DRIVER()
%Creates the Domain of the BVP:
fprintf('Begin Domain Creation\n');
s = 0;  %Begin Step Counter:
tic %Start Timer

DIRECTORY = pwd;
%Make a Files folder if there is none already
fileExist = exist('Files','file');
if fileExist ~= 7
    mkdir('Files')
```

```
23 end
24 cd('Files');
25
26 %Name the file for the current domain:
27 ID = '003';
28 FILENAME = strcat('INTRACELLULAR_INPUT_DATA_',ID);  %Domain
29 DOMAIN = 'INTRACELLULAR';
30
31 %Open a txt file to save all of the data:
32 ntxt = strcat(FILENAME,'_VALUES.txt');
33 writefile = fopen(ntxt,'wt');
34
35 %Location of the Membrane/Length of Space [m]:
36 L=10^-6; %1 micro meter;
37 fprintf(writefile,'Scalar Parameters for the Domain: \n');
38 fprintf(writefile,'-----------------------------------\n');
39 fprintf(writefile,'Length of the Domain, L: %dm\n',L);
40
41 %Temperature of the Tissue [K]:
42 T = 293;
43 fprintf(writefile,'Temperature, T: %dK\n',T);
44
45 %Assign a "Stimulation Frequency/Intensity" for the HH Equations:
46 intensity = 10;
47 duration = 2;
48 delaytime = 10;
49
50 fprintf(writefile,'Hodgkin-Huxley Parameters: \n');
51 fprintf(writefile,'\tIntensity: %d\n',intensity);
52 fprintf(writefile,'\tDuration: %d\n',duration);
53 fprintf(writefile,'\tDelay Time: %d\n',delaytime);
54
55 %Choose the Grid Size in the Spatial Domain:
56 nz = 100;
57 fprintf(writefile,'Spatial Grid Size, nz: %d\n', nz);
58
59 %Initial Time - Keep at 0s ALWAYS
60 t0 = 0;
61 fprintf(writefile,'Initial Time, t0: %ds\n', t0);
62
63 %Choose an Ending Time for the Simulation [s]:
```

```
64 tf = 60;
65 fprintf(writefile,'End Time, tf: %ds\n', tf);
66
67 %Choose a CFL Condition such that
68 %     CFL > D_a FOR ALL of the ions:
69 CFL = 10^-8;
70 fprintf(writefile,'CFL Condition: %d\n', CFL);
71
72 %%
73 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
74
75
76 %Define the Spatial Step dz:
77 dz = L/nz;
78 fprintf(writefile,'Spatial Step, dz: %dm\n', dz);
79
80 %Define a cutoff for number of timesteps:
81 test = 10^5;
82 fprintf(writefile,'Cutoff Number, test: %d\n',test);
83
84 %Define the TimeStep dt by the CFL condition
85 dt = dz^2/CFL;
86
87 %Define the Grid Size nt from dt
88 nt = ceil((tf - t0)/dt);
89 fprintf(writefile,'Time Grid Size, nt: %d\n',nt);
90
91 if nt < test
92     %FIX dt TO CONFORM TO THE ROUNDED VALUE OF nt
93     dt = (tf - t0) / nt;
94 else
95     %Define the number of spatial steps if nt>cutoff value
96     nt = test;
97     dt = tf/nt;
98 end
99 fprintf(writefile,'Time Step, dt: %ds\n',dt);
100
101 %Create an Array of Time Steps
102 time = 0:(tf-0)/(nt-1):tf;
103 time = transpose(time);  %Make time a column vector
104
```

```
105 %Choose the size of the membrane
106 h = 10^-8;
107 fprintf(writefile,'Membrane Thickness, h: %dm\n',h);
108
109 %Hydraulic Permeability:
110 k0 = 7.5*10^-12; %White_Mater from Basser
111 fprintf(writefile,'Hydraulic Permiability, k0: %dm^4/(Ns)\n',k0);
112
113 s=s+1;
114 fprintf('Step %1.0f, Assigned Parameters, Time Elapsed: ...
      %7.4fs\n',...
115     s, toc);
116 %%
117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
118 %Perform Stimulation to Obtain HH Results
119 v = HH_STIMULATE(intensity, duration, delaytime, dt, tf);
120 %Convert v from [mV] to [V]
121 v = 10^-3*v;
122
123 s=s+1;
124 fprintf('Step %1.0f, Performed HH Stimulation, Time Elapsed: ...
      %7.4fs\n',...
125     s, toc);
126 %%
127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
128 %Create the Correct Voltage Distribution over the Domain
129 dv = zeros(nz, nt);
130 for i=1:nz
131     for j=1:nt
132         dv(i,j) = -v(j)/h;
133     end
134 end
135
136 s=s+1;
137 fprintf('Step %1.0f, Calculated dv/dz, Time Elapsed: %7.4fs\n',...
138     s, toc);
139
140 %%
141 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
142 save(FILENAME, 'L', 'T','DOMAIN','intensity',...
143     'duration','delaytime','nz','tf','CFL', 'dz', 'dt',...
```

```
144     'nt', 'time','t0','h', 'dv','k0');
145
146 %Change back to working Directory:
147 cd(DIRECTORY);
148 fclose('all');
149 fprintf('Step %1.0f, Wrote File, Time Elapsed: %7.4fs\n',...
150     s, toc);
151 fprintf('\nEnd Domain Creation\n');
152
153 end
```

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %       SOLVER FOR THE LINEAR TRIPHASIC MODEL IN TERMS OF
3  %       CONCENTRATIONS AND DILATATION OF THE MIXTURE FOR BRAIN
4  %
5  %AUTHOR: BRADFORD JOSEPH LAPSANSKY
6  %CREATION DATE: 2/12/2014
7  %
8  %
9  %
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 function [] = CONCENTRATION(nameION, nameDOMAIN, ifPlots, ...
       points, ID)
12 fprintf('\nBegin Concentration\n');
13 tic; %Start Timer
14 s = 0;  %Begin Step Counter:
15
16 %Make a Files folder if there is none already
17 fileExist = exist('Files','file');
18 if fileExist ~= 7
19     mkdir('Files')
20 end
21 addpath('Files');
22
23 %Load Data Files
24 DIRECTORY = pwd;
25 ion = load(nameION);
26 domain = load(nameDOMAIN);
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28 %%
```

```
29
30 %Obtain Necessary Quantities from the Files:
31 %Ion File
32 Da = ion.Da;   %Diffusivity
33 za = ion.za;   %Valence
34 c0 = ion.c0;   %BC at z=0
35 ION = ion.ION; %Name of the Ion
36 ABBRV = ion.ABBRV; %Abbreviation of the Ion Name
37
38 %Domain File:
39 T = domain.T;  %Tissue Temp
40 DOMAIN = domain.DOMAIN; %Domain Name
41
42 %Grid Parameters
43 nz = domain.nz;     %Grid Size
44 dz = domain.dz;     %Δ z
45 dt = domain.dt;     %Δ t
46 nt = domain.nt;     %# of timesteps
47 time = domain.time; %Array of Timesteps
48 dv = domain.dv;     %dv/dz at all space and time
49
50 s=s+1;
51 fprintf('Step %1.0f, Imported Files, Time Elapsed: %7.4fs\n', ...
       s, toc);
52 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53 %%
54 %Define Universal Constants
55 R = 8.314472; %Universal Gas Constant [J/(mol—K)]:
56 Fc = 96485.3383;  %Faraday [C/mol]  (CRC Handbook)
57
58 %%
59 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60 %Loop to find the remaining solutions:
61 %Create Constant Combinations
62 lambda = Da*dt/(2*dz^2);
63 Zeta = za*Fc*Da*dt/(4*dz*R*T)*dv;
64
65 %Form Phi and Theta:
66 theta = zeros(nz,nt);
67 phi=zeros(nz,nt);
68 ones = zeros(nz,1);
```

```
69
70 for i=1:nz
71     for j=1:nt
72         theta(i,j) = lambda - Zeta(i,j);
73         phi(i,j) = lambda + Zeta(i,j);
74     end
75     ones(i) = 1;
76 end
77
78 s=s+1;
79 fprintf('Step %1.0f, Begin Building Solution, Time Elapsed: ...
       %7.4fs\n',...
80     s, toc);
81
82 %Build Solution Function
83 ca = zeros(nz, nt);
84 [ca] = BUILD_SOLUTION(ca, dz, nz, nt, c0, lambda, theta, phi, ...
       ones);
85
86 s=s+1;
87 fprintf('Step %1.0f, Finished Building Solution, Time Elapsed: ...
       %7.4fs\n',...
88     s, toc);
89 %%
90 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
91 %Plot the and save data at various values of z:
92 %
93 %Cut the number of timesteps in order to make the graphs
94 %    look neater
95 if ifPlots == 1
96     PLOT_CONCENTRATION(ca, time, dz, nz, points, ION, ID);
97
98     s=s+1;
99     fprintf('Step %1.0f, Plotted Solution, Time Elapsed: ...
           %7.4fs\n',...
100        s, toc);
101 end
102 %%
103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104 %Save Solution Data:
105 %Save Concentration Data for the Ion.
```

```
106 %Change to the Files Folder
107 cd('Files');
108
109 %Save Files
110 FILENAME = strcat(ION, '_',DOMAIN,'_CONCENTRATION_RESULTS_', ID);
111 save(FILENAME, 'ca', 'za','ABBRV','c0');
112
113 %Change the Directory Back:
114 cd(DIRECTORY);
115
116 s=s+1;
117 fprintf('Step %1.0f, Saved Data, Time Elapsed: %7.4fs\n', s, toc);
118
119 fprintf('End Concentration\n');
120 end
121
122 function [u] = BUILD_SOLUTION(u, dz, nz, nt, c0, lambda, theta, ...
       phi, ones)
123 %BUILD_SOLUTION:  CREATES THE FULL GRID FOR THE INPUTTED ARGUMENTS
124 %%
125 %Create the Initial Solution:
126 u(:,1) = BUILD_INITIAL_SOLUTION(nz, dz, c0);
127
128 %Create the Matrix of Known Values: B
129
130 %FORM THE LOOP THAT ULTIMATELY BUILDS THE GRID:
131 minusLambda = (1-2*lambda)*ones;
132 plusLambda = (1+2*lambda)*ones;
133 B_temp = zeros(nz,1);  %Initialize B_temp
134
135 for n = 1:nt-1
136     %Form the Diagonals of the Matrix B:
137     B(:,1) = phi(:,n);
138     B(:,2) = minusLambda;
139     B(:,3) = theta(:,n);
140     B_temp= TDMULT(B,u(:,n));
141
142
143     %Form the Diagonals of A
144     Adiag(:,1) = -phi(:,n+1);
145     Adiag(:,2) = plusLambda;
```

```
146     Adiag(:,3) = -theta(:,n+1);
147
148     %Form the Tri-Diagonal Matrix A:
149     A = TRI_DIAG_MATVAR(Adiag,nz);
150
151     %MARCH AHEAD ONE TIME STEP:
152     %  EQUATION TO SOLVE IS u(n+1) = [A]^-1*[B]*u(n) USING LU ...
           DECOMP.
153     u(:,n+1) = SPEC_SOLVER(A, B_temp, nz, c0, theta, phi,n);
154
155     %CURRENT ENDPOINTS OF u(n+1) ARE GARBAGE, FIX
156     %  THE SOLUTION TO TAKE INTO ACCOUNT THE BOUNDARY CONDITIONS:
157     u(:,n+1) = COND_BOUNDARY(1, 1, u(:,n+1), dz,c0,0);  %LOWER
158     u(:,n+1) = COND_BOUNDARY(3, 3, u(:,n+1), dz,0,0);  %UPPER
159 end %for
160
161 end
162
163 function [uOut] = SPEC_SOLVER(A, Bu, nz, c0, theta, phi,n)
164 %SPEC_SOLVER: THIS WILL SOLVE THE CRANK NICHOLSON METHOD USING ...
       THE VECTOR
165 % u WITHOUT THE u(1) or u(nz) TERMS IN ORDER TO CORRECTLY USE ...
       THE NEUMANN
166 %B.C.
167
168 %COPY THE 2 THROUGH nz-1 TERMS INTO A TEMP VARIABLE TO SOLVE USING
169 %INVERSION:
170 matA = zeros(nz-2);
171
172 %FORM A TRUNCATED VERSION OF THE MATRIX OF u(n) VALUES IN ORDER
173 %TO PROPERLY USE THE CN-METHOD IN SPEC_SOLVER:
174 count = 0;
175 temp = zeros(nz-2,1);
176 for i=2:nz-1
177     for j=2:nz-1
178         matA(i-1,j-1) = A(i,j);  %Smaller Matrix (nz-1)x(nz-1)
179     end
180     temp(i-1,1) = Bu(i,1);  %Cutoff first and last values of {u}
181     count = count +1;
182 end %for
183
```

```
184 %Correct for the Dir. Boundary Conditions at the End:
185 temp(1,1) = temp(1,1) + phi(1,n+1)*c0;
186 % temp(nz-2,nz-2) = temp(nz-2,nz-2) + theta(nz,n)*cl;
187
188 %MODIFY THE LAST ENTRY OF matA TO SOLVE THE NEUMANN CONDITION ...
       AS SPECIFIED
189 %  BY DOUGLASS HARDER
190 matA(count,count) = matA(count,count)-4/3*theta(nz, n+1);
191 matA(count,count-1) = matA(count, count-1) +1/3*theta(nz,n+1);
192
193 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
194 %%
195 %Solve for the temp matrix at step n+1:
196 temp2 =matA\temp;
197
198 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
199 %%
200 %BUILD THE uOut VECTOR WITH THE FIRST AND LAST ENTRIES BEING ...
       GARBAGE:
201 uOut = zeros(nz,1);
202 for i=1:nz-2
203     uOut(i+1,1) = temp2(i,1);
204 end
205
206
207 %DECLARE THE LAST TWO ENTIRES TO BE GARBAGE, WHICH THEY ARE AT ...
       THIS MOMENT
208 uOut(1,1) = NaN;
209 uOut(nz,1) = NaN;
210
211 end
212
213 function [MAT] = TRI_DIAG_MATVAR(entry,N)
214 %TRI_DIAG:  CREATES A TRIDIAGONAL MATRIX OF SIZE "N" FOR THE ...
       GIVEN INPUTS
215
216 %INITIALIZE A MATRIX OF ZEROS OF SIZE N
217 MAT = zeros(N);
218
219 %PULL INFORMATION FROM THE ENTRIES
220
```

```
221 for i = 2:N-1
222     MAT(i,i) = entry(i,2);     %MAIN DIAG
223     MAT(i,i-1) = entry(i,1);   %LOWER DIAG
224     MAT(i,i+1) = entry(i,3);   %UPPER DIAG
225 end  %for
226
227 %FILL IN THE REMAINING SPOTS OF THE MATRIX:
228 %MAIN DIAG SPOTS:
229 MAT(1,1) = entry(1,2);
230 MAT(N,N) = entry(N,2);
231
232 %LOWER DIAG SPOTS:
233 MAT(N,N-1) = entry(N,1);
234
235 %UPPER DIAG SPOTS:
236 MAT(1,2) = entry(1,3);
237
238 end
239
240 function [v] = TDMULT(diag, u)
241 %T_DIAG_MULT: PROVIDES A QUICK WAY TO MULTIPLY A TRI-DIAG ...
       MATRIX WITH
242 %             A CONSTANT ARGUMENT ON THE DIAGONAL
243
244
245 %OBTAIN THE LENGTH OF THE VECTOR BEING MULTIPLIED:
246 n = length(u);
247 v = zeros(n,1);
248
249 %SET THE FIRST AND LAST TERMS OF THE MULTIPLICATIONS:
250 v(1,1) = diag(1,2)*u(1,1) + diag(1,3) * u(2,1);
251 v(n,1) = diag(n,1)*u(n-1,1) + diag(n,2)*u(n,1);
252
253 %%
254 %LOOP TO OBTAIN THE REST OF THE MULTIPLICATION:
255 for i = 2:(n-1)
256     v(i,1) = diag(i,1)*u(i-1,1) + ...
           diag(i,2)*u(i,1)+diag(i,3)*u(i+1,1);
257 end
258
259 end
```

```
260
261 function [u] = BUILD_INITIAL_SOLUTION(nz, dz, c0)
262 %BUILD INITIAL SOLUTION: CREATES THE INITIAL MATRIX u THAT ...
       COMPRISES THE
263 %ENTIRE GRID OF POINTS THAT WILL BE MADE USING THE CN METHOD.  ...
       THE FIRST
264 %ENTRY INTO THIS MATRIX WILL CONSIST OF THE SYSTEM STATE AT t = ...
       t0.  USING
265 %THE INITIAL CONDITIONS OF THE PROBLEM
266
267 %%CREATE THE MATRIX u USING THE CHOSEN SPACING:
268 u = zeros(nz,1);
269
270 %FORM THE MATRIX AT TIME t=t0 USING THE INITIAL CONDITIONS:
271 for i = 1:nz
272     u(i,1) = COND_INITIAL(c0);
273 end %for
274
275 %Fix the first and last boundary conditions, I WILL NOT apply ...
       treatment to
276 %    the center boundary condition since the motion of the ions ...
       did not
277 %    begin yet and the condition comes into play when the ions move
278 u(:,1) = COND_BOUNDARY(1,1,u(:,1),dz, c0,0);
279 u(:,1) = COND_BOUNDARY(3,0,u(:,1),dz, 0,0);
280 end
281
282 function [init] = COND_INITIAL(c0)
283 %COND_INITIAL:  THIS FUNCTION SERVES AS THE INITIAL CONDITION ...
       FOR e, AND
284 %gamma FOR THE PARAMETERS (Z, t0), IN MOST CASES THOUGH, t0 = ...
       0, BUT t0
285 %WILL BE USED IN CASE A TIME OTHER THAN ZERO IS USED FOR AN INITIAL
286 %CONDITION.
287 init = 100*c0;
288 end
289
290 function [u] = COND_BOUNDARY(IS_LOWER, IS_DIRICHLET, u, dz, ...
       importDir,...
291     importNeumann)
292 %COND_BOUNDARY: CALCULATES THE BOUNDARY CONDITION VALUE FOR THE ...
```

```
    GRID.  BOTH
293 %               BOUNDARY CONDITIONS ARE LOCATED HERE AND ARE ...
    CORRECTLY
294 %               OUTPUT TO THE CODE USING A FLAG FOR UPPER OR ...
    LOWER B.C.
295 %
296 %              IS_LOWER: IS THE B.C. THE LOWER ONE, IF SO THEN ...
    IS_LOWER IS
297 %                           FLAG
298 %              IS_DIRICHLET: IS THE B.C.  A DIRICHLET ONE, IF ...
    SO THEN
299 %                              IS_DIRICHLET = TRUE
300
301 %BOUNDARY CONDITION SPECIFICS:
302 %    LOWER B.C: DIRICHLET
303 %    UPPER B.C: NEUMANN
304 %
305 %LOWER B.C. VALUE/ u(z0,t) = ?
306 u_z0_t = importDir;  %DIR. COND
307 h_z0 = importNeumann;    %NEU  COND.
308
309 %UPPER B.C. VALUE/ u_x(zL,t) = ?
310 u_zL_t = importDir;  %DIR. COND
311 h_zL = importNeumann;    %NEU. COND
312
313
314 %%LOWER BOUNDARY CONDITION u(z0,t):
315 %THIS BOUNDARY CONDITION WILL ACTIVATE IF IS_LOWER = FLAG == 1:
316 FLAG = 1;
317
318 if IS_LOWER == FLAG
319     %LOWER BOUNDARY CONDITION:
320     if IS_DIRICHLET == FLAG
321         u(1,1) = u_z0_t;
322     else
323         u(1,1) = 4/3*u(2,1) - 1/3*u(3,1); %+ 2*dz*h_z0
324     end %if DIRICHLET
325 else
326     %%UPPER BOUNDARY CONDITION:
327     gridLen = length(u);
328     if IS_DIRICHLET == FLAG
```

```
329         u(gridLen,1) = u_zL_t;
330     else
331         u(gridLen,1) = 4/3*u(gridLen-1, ...
                1)-1/3*u(gridLen-2,1)+2*dz*h_zL;
332     end %if DIRICHLET
333 end  %if LOWER
334 
335 end
336 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
337 
338 
339 %Plot Concentrations:
340 function [] = PLOT_CONCENTRATION(c, time, dz, nz, points, ION, ID)
341 %PLOTS: Plots the concentration over time at specified points
342 backDir = pwd;    %Save Current Directory
343 
344 %Check to see if the folder's name exists:
345 figExist = exist('Figures', 'file');
346 if figExist ~= 7
347     mkdir('Figures');
348 end
349 cd('Figures');  %Change Directory to Figures
350 
351 figExistIon = exist(ION,'file');
352 if figExistIon ~= 7
353     mkdir(ION);
354 end
355 cd(ION);
356 
357 %Fix the c vector to plot properly
358 c=transpose(c);
359 
360 for k=1:length(points)
361     %Test if the passed point is in the array:
362     if points(k) > nz || points(k) < 1
363         %Continue to next point in the vector
364         continue
365     end
366 
367     dblLocation = dz*double(points(k))*10^6;  %MicroMeters
368     points(k) = uint64(points(k));  %Convert point to an ...
```

```
           integer value
369
370        %Create a Title
371        strTitle = strcat(ION,': Concentration versus time at z= ',...
372            num2str(dblLocation),'micro meters');
373        %Plot the Function
374        FIGURE = figure('Visible','off');
375        plot(time, c(:,points(k)),'k');
376        xlabel('Time (s)');
377        strLabelY = strcat(ION,' Concentration (mol/m^3)');
378        ylabel(strLabelY);
379
380        %%
381    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
382        %Create Filename:
383        ION_UCASE = upper(ION);
384        FILENAME = strcat(ION_UCASE,'_AT_Z',num2str(points(k)),'_',ID);
385
386
387        %Change Directories
388        %This block saves the MATLAB figure in .fig and .pdf in a ...
           folder called
389        %   "Figures".
390        %
391
392        filenameExist = exist(FILENAME,'file');
393        if filenameExist ~= 7
394            mkdir(FILENAME);  %Create New Folder
395        end
396        cd(FILENAME)
397        %Save the MATLAB Figure as a .fig file:
398        saveas(FIGURE, FILENAME,'fig');
399
400        %Save the same MATLAB Figure as a .pdf file"
401        saveas(FIGURE, FILENAME, 'pdf');
402
403        %Save the same MATLAB Figure as a .png file"
404        saveas(FIGURE, FILENAME, 'png');
405
406        txtFile = fopen(FILENAME,'wt');
407        strTxtInfo = strcat('Info. for ',ION,' Concentration\n');
```

```
408     fprintf(txtFile,strTxtInfo);
409     fprintf(txtFile,strTitle);
410     fclose('all');
411     cd ..
412 end
413
414 %Change back to current directory
415 cd(backDir);
416 end
```

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %Ion Specific File
3  %Author: Bradford Lapsansky
4  %
5  %This file contains all of the necessary information to solve for
6  %    the changing concentrations of ions over time.
7  %
8  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9  clc
10 clear all
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 fprintf('Begin Program\n');
13 DIRECTORY = pwd;
14 %Make a Files folder if there is none already
15 fileExist = exist('Files','file');
16 if fileExist ≠ 7
17     mkdir('Files')
18 end
19 cd('Files');
20
21 %------------------------------Ion ...
       Names-----------------------------------------
22 ION ='SODIUM';
23 ABBRV = 'Na';
24
25 % ION ='POTASSIUM';
26 % ABBRV = 'K';
27
28 % ION ='CHLORINE';
29 % ABBRV = 'Cl';
```

```
30
31 %
32 %
33 %
34 %%
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36 %Define the Diffusion Coefficient of the Ion [m^2/s]:
37 Da = 13.3*10^-10;  %Sodium
38 % Da = 1.96*10^-9;   %Potassium
39 % Da = 2.03*10^-9;     %Chlorine
40
41 % Define the Valence of the Ion:
42 za = 1;   %Sodium/Potassium
43 % za = -1;  %Chlorine
44
45 %Initial Quantities of the Ion at t=0 in [mol/m^3]
46 %From Kandel et al. (2012) in mM data page 129
47 c0 = 50;   %Sodium
48 % c0 = 400;  %Potassium
49 % c0 = 52;    %Chlorine
50
51 %Create the Filename:
52 FILENAME = strcat(ION,'_INPUT');
53 save(FILENAME,'Da','za','c0','ION','ABBRV');
54
55 cd(DIRECTORY);  %Change back to the working directory
56
57 %Display Closing Information
58 fprintf('     Ion Name: ');
59 fprintf(ION);
60 fprintf('\nWrote File, End Program\n');
```

```
1 function [] = PLOT_COMBINED(kName, naName, clName, domainName, ...
      points, ID)
2 %PLOTS: Plots the concentration over time at specified points ...
      for all
3 %       Ions and the FCD
4 backDir = pwd;    %Save Current Directory
5
6 %Load Concentration Files:
```

```
cd('Files');
na = load(naName);
k = load(kName);
cl = load(clName);
domain = load(domainName);
cd(backDir);

%Pick Necessary Info. from the various files:
%Domain:
dz = domain.dz;
nz = domain.nz;
time = domain.time;

%Concentrations:
cna = na.ca;    %Sodium
ck = k.ca;      %Potassium
ccl = cl.ca;    %Chlorine

%Initial Concentrations:
cna0 = na.c0;
ck0 = k.c0;
ccl0 = cl.c0;

%Scaled Concentration Values by c0:
%    c_ION/c0_ION:
sna = cna./cna0;
sk = ck./ck0;
scl = ccl./ccl0;

%Create a Folder for the Figures:
figExist = exist('Figures', 'file');
if figExist ≠ 7
    mkdir('Figures');
end
cd('Figures');  %Change Directory to Figures

figExistIon = exist('Combined','file');
if figExistIon ≠ 7
    mkdir('Combined');
end
cd('Combined');
```

```
48
49
50  for k=1:length(points)
51      %Test if the passed point is in the array:
52      if points(k) > nz || points(k) < 1
53          %Continue to next point in the vector
54          continue
55      end
56      %Location in the Domain of the Plot:
57      dblLocation = dz*double(points(k))*10^6;  %MicroMeters
58      points(k) = uint64(points(k));  %Convert point to an ...
            integer value
59
60      %Limit the Number of Sodium Points so that it doesn't ...
            overlap with
61      %   the Potassium Plot:
62      skip = 600;
63      n = 0;
64      for i=1:skip:length(time);
65          n=n+1;
66          limNa(points(k),n) = sna(points(k),i);
67          limTime(n) = time(i);
68      end
69
70      %Create a Title
71      strTitle = strcat('All Ionic Concentrations versus time at ...
            z= ',...
72          num2str(dblLocation),'micro meters');
73
74
75      %Plot the Function
76      FIGURE = figure('Visible','off');
77      hold on
78      %     Note the Limited Points of Sodium/Na that are being ...
            plotted
79      plot(time, sk(points(k),:),'k', limTime, limNa(points(k),:),...
80          'xk',time, scl(points(k),:),'--k', time, ...
                scF(points(k),:), ':k');
81      axis([0, 60, 0.998, 1.0025]);
82      xlabel('Time (s)');
83      legend('K^+','Na^+', 'Cl^-','FCD','Location','Southwest');
```

```
84      ylabel('Scaled Concentration: c/c_0');
85      hold off
86
87      %%
88      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89      %Create Filename:
90      FILENAME = strcat('COMBINED_AT_Z',num2str(points(k)),'_',ID);
91
92
93      %Change Directories
94      %This block saves the MATLAB figure in .fig and .pdf in a ...
            folder called
95      %   "Figures".
96      filenameExist = exist(FILENAME,'file');
97      if filenameExist ~= 7
98          mkdir(FILENAME);  %Create New Folder
99      end
100     cd(FILENAME)
101     %Save the MATLAB Figure as a .fig file:
102     saveas(FIGURE, FILENAME,'fig');
103
104     %Save the same MATLAB Figure as a .pdf file"
105     saveas(FIGURE, FILENAME, 'pdf');
106
107     txtFile = fopen(FILENAME,'wt');
108     strTxtInfo = strcat('Info. for Combined Ionic ...
            Concentrations\n');
109     fprintf(txtFile,strTxtInfo);
110     fprintf(txtFile,strTitle);
111     fprintf(txtFile,'c0 for Na: %d\n', cna0);
112     fprintf(txtFile,'c0 for K: %d\n', ck0);
113     fprintf(txtFile,'c0 for Cl: %d\n', ccl0);
114     fclose('all');
115     cd ..
116 end
117
118 %Change back to current directory
119 cd(backDir);
120 end
```

Appendix B
MATLAB Code for Chapter 3

B.1 Numerical Solution

```
1  %NAME:  NUMERIC SOLUTION TO EQUATIONS 1 AND 2 IN LU ET. AL.
2  %AUTHOR: BRADFORD LAPSANSKY
3  %
4  %DESCRIPTION: THIS CODE WILL USE THE CRANK—NICHOLSON METHOD IN ...
       ORDER TO
5  %             PERFORM A NUMERICAL SOLUTION TO EQUATIONS 1 AND 2 ...
       IN LU ET.
6  %             AL.  THE PURPOSE OF THE NUMERICAL SOLVER IS TO ...
       ALLOW THE USER
7  %             TO IMPLEMENT ARBITRARY BOUNDARY CONDITIONS FOR ...
       THE SOLUTION
8  %             AND TO COMPARE THE SOLUTION DERIVED IN THIS ...
       PROBLEM TO THE
9  %             ALREADY DERIVED FOURIER SERIES SOLUTION TO THESE ...
       EQUATIONS.
10 %
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 %%IMPORT DATA AND FORM THE MATRICES [A], [M], AND [LAMBDA]
13 %
14 %LOAD INPUT PARAMETERS FROM A MATLAB FILE:
15 clc
16 clear all
17
18 WORKING_DIR = pwd;
```

```
19
20 fprintf('CURRENT DIRECTORY = ')
21 disp(WORKING_DIR);
22 ls DATA*    %DISPLAY CONTENTS OF CURRENT DIRECTORY
23 fprintf('\n————————————————————————————————————————————————\n');
24
25 % FILENAME = input('Please Enter a Filename for Input: ','s');
26 FILENAME = 'DATA_MALAKPOOR.mat';
27 fprintf('\n');
28
29 LOADED_DATA = load(FILENAME);
30 inp = LOADED_DATA.inp;                %DATA POINTS
31 AUTHOR = LOADED_DATA.LEAD_AUTHOR;     %AUTHOR OF DATA
32
33 %DISPLAY AUTHOR OF INPUT DATA ON THE SCREEN
34 fprintf('Author of Data: ');
35 disp(AUTHOR)
36 fprintf('\n\n')
37
38 %FORM THE MATRICES [A], [M], AND [LAMBDA]
39 [A, M, LAMBDA] = FORM_A(inp);
40
41 %Display the Values of [A]
42 fprintf('\nMatrix A: \n');
43 for i=1:length(A)
44     for j=1:length(A)
45         scale = ORDER_MAGNITUDE(A(i,j));
46         fprintf('a(%1.0f,%1.0f) = %4.2f x 10^(%3.0f)\n', i,...
47             j, A(i,j)/(10^scale), scale);
48     end
49 end
50 fprintf('\n\n');
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52 %%DETERMINE THE GRID SPACING FOR DELTA_Z AND DELTA_t
53 nz = 100;  %NUMBER OF SPACES TO CREATE THE DELTA_Z GRID
54
55 %STARTING AND ENDING TIME IN SECONDS
56 t0 = 0;
57 endT = 3600; %1 Hour
58
59 %VALUE OF r USED TO DETERMINE THE CONVERGENCE CONDITION:
```

```
r = 0.5;  %NOTE: r<1/2 FOR NON-OSCILLATING GRIDS

%FOR THE GRID SPACING
[dz, dt, nt, endT] = GRID_SPACING(inp, LAMBDA, nz, endT, t0, r);

fprintf('1) GRID SPACING COMPLETED \n');
fprintf('     dz = %7.6f', dz);
fprintf('\n     dt = %7.6f', dt);
fprintf('\n     nz = %6.0f', nz);
fprintf('\n     nt = %6.0f', nt);
fprintf('\n');
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%BUILD THE INITIAL SOLUTION TO THE PROBLEM:
%NOTE: varCoice = 1 for "f", varChoice = 2 for "g"
f0 = BUILD_INITIAL_SOLUTION(M, nz, nt, 1, inp, dz);
g0 = BUILD_INITIAL_SOLUTION(M, nz, nt, 2, inp, dz);
fprintf('2) INITIAL VALUES FOR f AND g ARE NOW CREATED \n');
%%
%CALCULATE THE LOWER-DIRICHLET BOUNDARY CONDITIONS FOR f AND g:
e_lDir = 0;
gamma_lDir = 0;
vect_fg_lDir = pinv(M)*[e_lDir;gamma_lDir];

f_lDir = vect_fg_lDir(1);
g_lDir = vect_fg_lDir(2);
%%

%BUILD THE FULL GRIDS FOR BOTH uf and ug
f = BUILD_SOLUTION(f0, dz, dt, nz, nt, 1, LAMBDA, f_lDir);
g = BUILD_SOLUTION(g0, dz, dt, nz, nt, 2, LAMBDA, g_lDir);
fprintf('3) THE GRIDS FOR f AND g ARE NOW CREATED \n');

%TRANSFORM (f,g) into (e,gamma) using the matrix [M]:
[e, gamma] = TRANSFORM_GRID(f, g, M, nz, nt);
fprintf('4) THE GRIDS FOR e AND gamma ARE NOW CREATED \n');
fprintf('  WITH A NUMBER OF TIME STEPS nt = %6.0f', nt);
fprintf('\n');

%%
%SAVE PROGRAM DATA:
```

```
101 %TO THE MOW_ERROR FOLDER:
102 %DETERMINE THE SPOT AT WHICH TO COMPARE THE TWO ARRAYS
103 MULTIPLE = 1;  %MUST BE 0≤ MULTIPLE ≤ 1
104 z_Reach = MULTIPLE*(inp(11)-inp(12));
105
106 FILENAME = strcat('RESULTS_NUMERIC_', AUTHOR);
107 save(FILENAME, 'e', 'gamma','dz','dt','nt','nz','z_Reach');
108
109 FILENAME2 = strcat('RESULTS_FROM_NUMERIC_',AUTHOR);
110 save(FILENAME2, 'dz', 'dt', 'nt', 'nz','z_Reach', 't0','endT');
111 endProgram=strcat('5) SAVE RESULTS AS: ', FILENAME,' AND END ...
      PROGRAM \n');
112
113 fprintf(endProgram);
```

```
1  function [A, M, Lambda] = FORM_A(input)
2  %THIS FUNCTION FORMS THE MATRIX A FROM A1, A2, A4, AND A5 AS ...
       SPECIFIED IN
3  %   LU ET. AL.
4  %
5  %THIS FUNCTION ALSO DEFINES THE PARAMETERS OF A1, A2, A4, AND A5
6
7  %%%%%%%THIS SECTION CONTAINS INPUT TO SYSTEM%%%%%%%%%%%%%%%%
8
9
10 %%%%%%%%%%%%%%%%FORMED QUANTITIES%%%%%%%%%%%%%%%%
11 %THESE QUANTITIES ARE FORMED FROM THE input QUANTITIES ABOVE
12 %THEY SHOULD NEVER! BE EDITED
13 %
14 %
15 %c0^k:
16 c0k = input(5) + input(6);
17
18 %VARIOUS D^ QUANTITIES:
19 Da = 0.5*(input(1)+input(2));    %(D^+ + D^-)/2
20 Db = input(3)*input(4);          %Ha/k
21 Dd = 0.5*(input(1)-input(2));    %(D^+ - D^-)/2
22 Dk = Da + c0k/input(7)*Dd;       %Da + (c0^k/c0^F)*Dd
23 DF = c0k/input(7)*Da + Dd;       %Dd + (c0^k/c0^F)*Da
24
```

```
%Quantities A1, A2, A4, and A5:
%A(1,1) = A1
%A(1,2) = -A2
%A(2,1) = -A5
%A(2,2) = A4

%For ease of editing, let "exTerms = R*T*c0^F*phi0w*k:
exTerms = input(8)*input(9)*input(7)*input(10)*input(4);

%A1
A(1,1) = Db - (Db - Da)*(1+DF/exTerms)^-1;

%-A2
A(1,2) = -(Db*Dd/DF)*(1+exTerms/DF)^-1;

%-A5
A(2,1) = Dk/Db*(A(1,1) - (Db + exTerms*Dd/DF));

%A4
A(2,2) =  Da + Dk/Db*(A(1,2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FORM THE EIGENVECTOR MATRIX OF A AND THE DIAGONAL MATRIX OF A's
%EIGENVALUES FOR USE IN THE PROBLEM SOLUTION

[M, Lambda] = eig(A);
end
```

```
function [order] = ORDER_MAGNITUDE(testNum)
%ORDER_MAGNITUDE: CALCULATES THE ORDER OF MAGNITUDE OF A NUMBER
%THAT IS PASSED TO THE FUNCTION

%TEST IF THE NUMBER IS >1 OR <1
if abs(testNum)>1
    flag = 1;
elseif abs(testNum) == 1
    flag = 0;
    order = 1;
else
    flag = -1;
```

```
13 end
14 %VARIABLE TO END THE TEST AT:
15 killTest = 99;
16 logTest = log10(testNum);
17
18 %PERFORM THE TEST
19 if flag ≠ 0
20     for k=0:flag:flag*killTest
21         if(logTest≥k && logTest<k+1)
22             order = k;
23             break
24         end
25     end
26 end
27 end
```

```
1  function [dZ, dt, nt, endT] = GRID_SPACING(inp, LAMBDA, ...
       nz, endT, t0, r)
2  %GRID_SPACING:  THIS FUNCTION RETURNS THE SPACING DELTA_Z AND ...
       DELTA_t SO
3  %               THAT THE SPACING SATISFIES THE STABILITY CONDITION:
4  %               ri = (LAMBDA(i,i)*DELTA_t)/(DELTA_Z)^2 ≤ 0.5
5
6  %%CHOOSE THE DELTA_X SPACING BASED ON THE
7  dZ = (inp(11)-inp(12))/nz;
8
9  %CREATE A DELTA_t BASED ON THE LARGER EIGENVALUE IN LAMBDA
10 l = 0;
11 for k = 1:length(LAMBDA)
12     if LAMBDA(k,k) - l > 0
13         l = LAMBDA(k,k);
14     end %if
15 end %for
16
17
18 %DETERMINE THE SPACING FOR DELTA_T USING THE VALUE CONDITION:
19 %[l*DELTA_t)/(DELTA_Z)^2 = H] ≤ 0.5
20 dt = dZ^2*r/l;
21
22 %%DETERMINE THE VALUE OF nt TO THE CLOSEST INTEGER WITH ...
```

```
    TRUNCATION USING
%THE ENDING TIME endT
nt = ceil((endT - t0)/dt);

%FIX dt TO CONFORM TO THE ROUNDED VALUE OF nt
dt = (endT - t0) / nt;
end
```

```
function [u] = BUILD_INITIAL_SOLUTION(M, nz, nt, varChoice, ...
    inp, dz)
%BUILD INITIAL SOLUTION: CREATES THE INITIAL MATRIX u THAT ...
    COMPRISES THE
%ENTIRE GRID OF POINTS THAT WILL BE MADE USING THE CN METHOD.  ...
    THE FIRST
%ENTRY INTO THIS MATRIX WILL CONSIST OF THE SYSTEM STATE AT t = ...
    t0.  USING
%THE INITIAL CONDITIONS OF THE PROBLEM

%%CREATE THE MATRIX u USING THE CHOSEN SPACING:
u = zeros(nz,1);


%DETERMINE THE t0 CONDITION FROM THE INITIAL CONDITIONS OF THE ...
    PROBLEM:
ut0 = COND_INITIAL(0,0,inp);

%PLACE THE B.C. IN TERMS OF (f;g) INSTEAD OF (e; gamma)
ut0 = pinv(M)*ut0;

%%FORM THE MATRIX AT TIME t=t0 USING THE INITIAL CONDITIONS:
for i = 1: nz
    u(i,1) = ut0(varChoice);
end %for

%%OVERRIDE THE INITIAL AND FINAL VALUES OF u(i,1) TO COINCIDE ...
    WITH THE
%STATED BOUNDARY CONDITIONS FOR THE PROBLEM:
%NOTE: COND_BOUNDARY(IS_LOWER, IS_DIRICHLET, uMod, dz)

u(:,1) = COND_BOUNDARY(1,1, u(:,1), dz); %LOWER
```

```
27 u(:,1) = COND_BOUNDARY(0,0, u(:,1), dz); %UPPER
28 end
29
30 function [v_t0] = COND_INITIAL(z, t0, inp)
31 %COND_INITIAL:  THIS FUNCTION SERVES AS THE INITIAL CONDITION ...
       FOR e, AND
32 %gamma FOR THE PARAMETERS (Z, t0), IN MOST CASES THOUGH, t0 = ...
       0, BUT t0
33 %WILL BE USED IN CASE A TIME OTHER THAN ZERO IS USED FOR AN INITIAL
34 %CONDITION.
35
36 %RETURN v0 AS A COLUMN VECTOR OF e AND gamma
37 e0 = 10^-4;
38 gamma0 = inp(8)*inp(9)/inp(3)*(inp(5)+inp(6));
39
40 v_t0 = [e0; gamma0];
41 end
42
43 function [u] = COND_BOUNDARY(IS_LOWER, IS_DIRICHLET, u, dz)
44 %COND_BOUNDARY: CALCULATES THE BOUNDARY CONDITION VALUE FOR THE ...
       GRID.  BOTH
45 %               BOUNDARY CONDITIONS ARE LOCATED HERE AND ARE ...
       CORRECTLY
46 %               OUTPUT TO THE CODE USING A FLAG FOR UPPER OR ...
       LOWER B.C.
47 %
48 %              IS_LOWER: IS THE B.C. THE LOWER ONE, IF SO THEN ...
       IS_LOWER IS
49 %                           FLAG
50 %              IS_DIRICHLET: IS THE B.C.  A DIRICHLET ONE, IF ...
       SO THEN
51 %                              IS_DIRICHLET = TRUE
52
53 %BOUNDARY CONDITION SPECIFICS:
54 %    LOWER B.C: DIRICHLET
55 %    UPPER B.C: NEUMANN
56 %
57 %LOWER B.C. VALUE/ u(z0,t) = ?
58 u_z0_t = 0;   %DIR. COND
59 h_z0 = 0;     %NEU  COND.
60
```

```
%UPPER B.C. VALUE/ u_x(zL,t) = ?
u_zL_t = 0;  %DIR. COND
h_zL = 0;    %NEU. COND


%%LOWER BOUNDARY CONDITION u(z0,t):
%THIS BOUNDARY CONDITION WILL ACTIVATE IF IS_LOWER = FLAG == 1:
FLAG = 1;

if IS_LOWER == FLAG
    %LOWER BOUNDARY CONDITION:
    if IS_DIRICHLET == FLAG
        u(1,1) = u_z0_t;
    else
        u(1,1) = 4/3*u(2,1) - 1/3*u(3,1); %+ 2*dz*h_z0
    end %if DIRICHLET
else
    %%UPPER BOUNDARY CONDITION:
    gridLen = length(u);
    if IS_DIRICHLET == FLAG
        u(gridLen,1) = u_zL_t;
    else
        u(gridLen,1) = 4/3*u(gridLen-1, ...
            1)-1/3*u(gridLen-2,1)+2*dz*h_zL;
    end %if DIRICHLET
end  %if LOWER
end
```

```
function [u] = BUILD_SOLUTION(u0, dz, dt, nz, nt, ...
    varChoice,LAMBDA, u_lDir)
%BUILD_SOLUTION:  CREATES THE FULL GRID FOR THE INPUTTED ARGUMENTS

%INITIALIZE THE GRID TO BE ALL VALUES OF ZERO
u = zeros(nz, nt);

%INITIALIZE THE FIRST COLUMN OF u TO BE u0:
u(:,1) = u0(:,1);

%FORM THE VALUE OF r USED BASED ON THE SELECTION INDICATED IN ...
    varChoice AND
```

```
%THE EIGENVALUE BASED ON varChoice:

%CALCULATE THE VALUE OF r USED IN THE PROBLEM:
r = LAMBDA(varChoice, varChoice) * dt / (dz)^2;

%FORM THE MATRICES [A], AND [B] ACCORDING TO THE CN CONVENTION
A = [-r,2*(1+r), -r];
B = [r,2*(1-r), r];

%%
%FORM THE LOOP THAT ULTIMATELY BUILDS THE GRID:
for n = 1:nt-1
    %FORM B*temp IN THE CN METHOD
    B_temp= TDMULT(B,u(:,n));

    %MARCH AHEAD ONE TIME STEP:
    %  EQUATION TO SOLVE IS u(n+1) = [A]^-1*[B]*u(n) USING LU ...
       DECOMP.
    u(:,n+1) = SPEC_SOLVER(A, B_temp, nz, r, u_lDir);

    %CURRENT ENDPOINTS OF u(n+1) ARE GARBAGE, FIX
    %  THE SOLUTION TO TAKE INTO ACCOUNT THE BOUNDARY CONDITIONS:
    u(:,n+1) = COND_BOUNDARY(1, 1, u(:,n+1), dz);  %LOWER
    u(:,n+1) = COND_BOUNDARY(0, 0, u(:,n+1), dz);  %UPPER
end %for
end

function [v] = TDMULT(diag, u)
%T_DIAG_MULT: PROVIDES A QUICK WAY TO MULTIPLY A TRI-DIAG ...
   MATRIX WITH
%             A CONSTANT ARGUMENT ON THE DIAGONAL


%OBTAIN THE LENGTH OF THE VECTOR BEING MULTIPLIED:
n = length(u);
v = zeros(n,1);

%SET THE FIRST AND LAST TERMS OF THE MULTIPLICATIONS:
v(1,1) = diag(2)*u(1,1) + diag(3) * u(2,1);
v(n,1) = diag(1)*u(n-1,1) + diag(2)*u(n,1);

```

```
%%
%LOOP TO OBTAIN THE REST OF THE MULTIPLICATION:
for i = 2:(n-1)
    v(i,1) = diag(1)*u(i-1,1) + diag(2)*u(i,1)+diag(3)*u(i+1,1);
end
end

function [uOut] = SPEC_SOLVER(A, Bu, nz, r, u_lDir)
%SPEC_SOLVER: THIS WILL SOLVE THE CRANK NICHOLSON METHOD USING ...
    THE VECTOR
% u WITHOUT THE u(1) or u(nz) TERMS IN ORDER TO CORRECTLY USE ...
    THE NEUMANN
%B.C.

%COPY THE 2 THROUGH nz-1 TERMS INTO A TEMP VARIABLE TO SOLVE USING
%INVERSION:
matA = zeros(nz-2,3);
count = 0;

%FORM A TRUNCATED VERSION OF THE MATRIX OF u(n) VALUES IN ORDER
%TO PROPERLY USE THE CN-METHOD IN SPEC_SOLVER:
temp = zeros(nz-2,1);
for i=2:nz-1
    temp(i-1,1) = Bu(i,1);
    %FORM THE TRI-DIAGONAL PORTIONS OF THE MATRIX SO THAT THE ...
        BOUNDARY
    %CONDITIONS ARE SET-UP FOR A NEUMANN CONDITION AT z=L:
    matA(i-1,1) = A(1);
    matA(i-1,2) = A(2);
    matA(i-1,3) = A(3);
    count = count +1;
end %for

temp(1,1) = temp(1,1) + r*u_lDir;
%MODIFY THE LAST ENTRY OF matA TO SOLVE THE NEUMANN CONDITION ...
    AS SPECIFIED
%  BY DOUGLASS HARDER
matA(count,1) = -2*r/3;
matA(count,2) = 2+2*r/3;

%%
```

```
87 %INVERT THE temp MATRIX USING THE THOMAS ALGORITHM:
88 temp2 = THOMASVAR(matA, temp);
89
90 %%
91 %BUILD THE uOut VECTOR WITH THE FIRST AND LAST ENTRIES BEING ...
      GARBAGE:
92 uOut = zeros(nz,1);
93 for i=1:nz-2
94     uOut(i+1,1) = temp2(i,1);
95 end
96
97 %DECLARE THE LAST TWO ENTIRES TO BE GARBAGE, WHICH THEY ARE
98 uOut(1,1) = NaN;
99 uOut(nz,1) = NaN;
100 end
101
102 function [invTri] = THOMASVAR(triMat, vect)
103 %THOMAS: USES THE THOMAS ALGORITHM TO SOLVE FOR THE VECTOR x IN THE
104 %        SYSTEM: [A]x = y WHERE [A] IS A TRI-DIAGONAL MATRIX:
105 %NOTE:
106 %triMat = [A]
107 %vect = {y}
108 %invTri = {x}
109
110 %CALCULATE THE SIZE OF THE SYSTEM:
111 nz = length(vect);
112 invTri = zeros(nz,1);  %INITIALIZE
113
114 %%
115 %OBTAIN VALUES FOR THE INTERMEDIATE VARIABLES:
116
117 a = triMat(:,1);
118 b = triMat(:,2);
119 c = triMat(:,3);
120
121 %INITIALIZE THE INTERMEDIATE VARIABLES:
122 cPr = zeros(nz,1);
123 dPr = zeros(nz,1);
124
125 %FOR i=1:
126 cPr(1) = c(1)/b(1);
```

```
127 dPr(1) = vect(1)/b(1);
128
129 for i=2:nz
130     cPr(i) = c(i)/(b(i)-cPr(i-1)*a(i));   %c' Values
131     dPr(i) = (vect(i)-dPr(i-1)*a(i))/(b(i)-cPr(i-1)*a(i)); %d' ...
            Values
132 end %for i
133
134 %%
135 %CALCULATE THE VALUES OF invTri USING BACK-SUBSTITUTION:
136 invTri(nz) = dPr(nz);
137
138 for i=nz-1:-1:1
139     invTri(i) = dPr(i) - cPr(i)*invTri(i+1);
140 end %For i
141 end %Function
```

```
1  function [e, gamma] = TRANSFORM_GRID(f,g,M,nz,nt)
2  %TRANSFORM_GRID: TRANSFORMS EACH COUPLED POINT [f(i,n);g(i,n)] ...
       INTO THEIR
3  %CORRESPONDING VALUES OF [e(i,n); gamma(i,n)] FOR ALL POINTS IN ...
       THE GRIDS:
4
5  %INITIALIZE THE GRID OF e AND gamma BASED ON THE SIZES OF nz ...
       and nt
6  e = zeros(nz, nt);
7  gamma = zeros(nz, nt);
8
9  for i=1:nz
10     for n = 1:nt
11
12         %DEFINE THE VECTOR OF (f,g)in TO BE vm_in
13         vm_in = [f(i,n); g(i,n)];
14
15         %CALCULATE THE TRANSFORMATION OF THESE POINTS THROUGH ...
               THE MATRIX
16         %[M] FOR ALL POINTS IN THE GRID TO FORM v_in = [e; ...
               gamma]in:
17         v_in(:,1) = M*vm_in;
18
```

```
        %POPULATE THE GRIDS OF e AND gamma:
        e(i,n) = v_in(1,1);
        gamma(i,n) = v_in(2,1);
    end %for n
end %for i
```

B.2 Analytic Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%A TRIPHASIC MODEL FOR BRAIN MATERIAL%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%ANALYTICAL SOLUTION%%%%%%%%%%%%%%%%
%CODE AUTHOR: BRADFORD JOSEPH LAPSANSKY
%CODE SPECIFICS: ANALYTIC SOLUTION TO THE SYSTEM OF EQUATIONS 1 ...
    AND 2 IN
%   "A LINEARIZED FORMULATION OF TRIPHASIC MIXTURE THEORY FOR ...
    ARTICULAR
%   CARTILAGE AND ITS APPLICATION TO INDENTATION ANALYSIS" BY ...
    LU ET. AL.
%   FOR A 1-DIMENSION IN SPACE AND TIME
%
%VERSION: 0.01
%RELEASE DATE:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%MATLAB SCREEN COMMANDS
clc
clear all

% cd('X:\Thesis\Thesis_Material\Code\Mow_Analytic');
WORKING_DIR = pwd;

fprintf('CURRENT DIRECTORY = ')
disp(WORKING_DIR);
ls DATA*    %DISPLAY CONTENTS OF CURRENT DIRECTORY
fprintf('\n-----------------------------------------------\n');

% %%%%%%%%%%%%%%%%%%%VARIOUS INPUT ARGUMENTS:%%%%%%%%%%%%%
% %%%%%% GUIDE TO INPUT VECTOR IS BELOW%%%%%%%%%%%%%%%%%%%%%%%
```

```
% inp(1) %D+ in m^2 s^-1
% inp(2) %D- in m^2 s^-1
% inp(3) %Ha in Pa
% inp(4) %k in m^4 N^-1 s^-1
% inp(5) %c0^+ in mol m^-3
% inp(6) %c0^- in mol m^-3
% inp(7) %c0^F in mol m^-3
% inp(8) %R in J mol^-1 K^-1
% inp(9) %T in K
% inp(10)%phi0w in [-]
% inp(11)%L - Height of Sample

%LOAD INPUT PARAMETERS FROM A MATLAB FILE:
%FILENAME = input('Please Enter a Filename for Input: ','s');
FILENAME = 'DATA_MALAKPOOR.mat';
fprintf('\n');
FILENAME2 = 'RESULTS_FROM_NUMERIC_MALAKPOOR.mat';
fprintf('\n')

%FROM THE FILE WITH THE EXPERIMENT PARAMETERS
LOADED_DATA = load(FILENAME);
inp = LOADED_DATA.inp;                %DATA POINTS
AUTHOR = LOADED_DATA.LEAD_AUTHOR;     %AUTHOR OF DATA

%FROM THE NUMERIC SOLUTION PARAMETERS
LOADED_DATA = load(FILENAME2);
nt = LOADED_DATA.nt;
dt = LOADED_DATA.dt;
z_Reach = LOADED_DATA.z_Reach;
endT = LOADED_DATA.endT;
startT = LOADED_DATA.t0;

fprintf('\nFile Loaded\n');

%DISPLAY AUTHOR OF INPUT DATA ON THE SCREEN
fprintf('Author of Data: ');
disp(AUTHOR)
fprintf('\n')

%%%%%%%%%%%%%%%%%%%%%%FORM NECESSARY MATRICES%%%%%%%%%%%%%%%
```

```
68 %FORM THE MATRIX [A] FROM A1, A2, A4, AND A5 IN ORDER TO TURN ...
       THE SYSTEM
69 %  INTO A TRADITIONAL SYSTEM OF EQUATIONS AND PERFORM THE
70 %  EIGEN-DECOMPOSITION OF THE MATRIX TO RETURN A MATRIX OF THE ...
       EIGENVALUES
71 %  OF [A] AND THE EIGENVECTORS OF [A]
72 %
73 %LET: [L] = diag(lambda1, lambda2)
74 %     [M] = EIGENVECTOR MATRIX
75
76 [A, M, L] = FORM_A(inp);   %[A], [M], [L]
77
78 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
79 %%%%%%%%%%%%%%%%FORM ANALYTICAL SOLUTION OF THE MATRIX%%%%%%%%%%%
80 %
81 %
82 %-------------------BOUNDARY CONDITIONS-----------------------------
83 %Let v = [e; gamma]
84 %DEFINE v0:
85 v0(1,1) = 10^-4;                                          %e0
86 v0(2,1) = inp(8)*inp(9)/inp(3)*(inp(5)+inp(6));    ...
       %gamma0=RT/Ha*(c0^k)
87
88 fprintf('\ne_0 = \n');
89 disp(v0(1));
90 fprintf('\n\gamma_0 = \n');
91 disp(v0(2));
92 fprintf('\n');
93 %%%%%%%%%%%%%%%%%%%%SOLVE THE DERIVED ANALYTICAL SYSTEM%%%%%%%%%%%%%
94 %%%%%%%%%%%%%%%%%%%%%%%%FOR AN ARRAY OF z AND t VALUES%%%%%%%%%%%
95 %
96 %OBTAIN THE zArr AND tArr VALUES FROM A WRITTEN FUNCTION:
97 %SET THE VALUES OF N, (NUMBER OF DISCREET POINTS IN SPACE AND ...
       TIME):
98 [zArr, tArr] = INIT_Z_T(z_Reach, nt, startT, endT);
99
100 %FIND THE ARRAY OF v VALUES
101 [vArr] = SOLVE_ARRAY(M, L, v0, inp(11), zArr, tArr);
102
103 %%%%%%%%%%%%%%%%%%%PLOT THE SOLUTIONS TO [e; gamma]%%%%%%%%%%%
104 FILENAME = strcat('RESULTS_ANALYTIC_', AUTHOR);
```

```
105 save(FILENAME, 'vArr','zArr','tArr', 'AUTHOR');
106 fprintf('\nEND PROGRAM \n');
```

```
1  function [zArr, tArr] = INIT_Z_T(len, N,tStart, tEnd)
2  %INIT_Z_T: THIS FUNCTION INITIALIZES THE VALUES OF z AND t FOR ...
       USE IN THE
3  %ANALYTICAL SOLUTION
4  %
5  %THE ARRAYS ARE SET BY PICKING RANGES [a,b] FOR POSITION AND ...
       TIME AND
6  %CREATING AN ARRAY BASED ON A DIVISION OF THE NUMBER OF POINTS, N:
7  %
8
9
10 %%%%%%%%%%%%%%%%%%%%SET TIME ARRAY%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 %SINCE THE BEGINNING TIME IS t = 0, t is in range [0, b]
12 %
13 %SET THE VALUE OF [at, bt] (in seconds)
14 at = tStart;
15 bt = tEnd; %2 Hours
16
17 tArr = at:(bt-at)/(N-1):bt;  %ARRAY OF t-Values
18
19 %%%%%%%%%%%%%%%%%SET POSITION ARRAY%%%%%%%%%%%%%%%%%%%%%%%%%%%
20 %POSITION ARRAY BASED ON THE LENGTH L
21 %
22 %SET THE VALUE OF [az, bz] (in meters)
23 %
24 %SET THE VALUES OF THE zArr TO ALL BE EQUAL TO len
25 for i = 1:length(tArr)
26     zArr(i) = len;
27 end
28 end
```

```
1  function [vArr] = SOLVE_ARRAY(M, L, v0, len, zArr, tArr)
2  %SOLVE_ARRAY: SOLVES THE ANALYTICAL SOLUTION FOR THE STATED ...
       B.C. FOR THE
3  %SYSTEM IN LU. ET. AL. FOR AN ARRAY OF Z AND t VALUES
```

```
4
5  %%%%CHECK TO SEE IF length(zArr) = length(tArr)
6  if length(zArr) == length(tArr)
7      ARR_LEN = length(zArr);    %Set ARR_Length = length(z)
8  else
9      vArr = 'error';
10 end %if
11
12 %%%%%%%%INITIALIZE THE VALUE OF v with respect to ARR_LENGTH
13 vArr = zeros(2,ARR_LEN);
14
15 %%%%%SOLVE FOR THE VARIOUS SOLUTIONS OF vArr
16 for i = 1:ARR_LEN
17     vArr(:,i) = SOLVE_SYS(M, L, v0, len, zArr(i), tArr(i));
18 end %for
19 end
20
21 function [v] = SOLVE_SYS(M,L,v0, len, z, t)
22 %SOLVE_SYS: Solves the Analytical Solution as Derived by ...
       Lapsansky from
23 %Equations 1 and 2 by Lu Et. Al.
24
25 %DEFINE NECESSARY CONSTANTS TO BE USED IN THE PROBLEM
26 Minv = pinv(M);          %[Minv] = [M]^-1
27
28 %STOPPING CONDITIONS
29 CLOSE_COND = 10 ^-50; %How Close sum(v) - sum(vOld) must be for ...
       series
30                       %to sufficiently converge
31 N_MAX = 10^6 ;         %How many iterations n should go increase ...
       to prevent
32                       %the program from crashing
33
34 %INITIALIZATION OF VALUES
35 n = 0;                 %Initialize n to zero
36 vOLD = [10; 10];       %Initial value of vOld
37 FLAG = 1;              %Initial value for the FLAG
38 R = zeros(2);          %Form an initial zero matrix for R
39 v = [0;0];             %Initialize the value of v for use in summing
40 %COLVE FOR v(z,t) USING A WHILE LOOP
41 while FLAG > CLOSE_COND
```

```
42      wn = 0.5*pi*(2*n+1)/len;      %w for specified value of n
43
44      %Calculate values of b(i) * t for use in [R]
45      for i = 1:2
46          R(i,i) = exp(-wn^2*L(i,i)*t);
47      end %for
48
49      %Calculate v for a specific z and t
50      v = v + (2*n+1)^-1*sin(wn*z)*M*R*Minv*v0;
51
52      %Form the FLAG variable
53      FLAG = norm(v- vOLD);
54
55      %Check to see in n>N_MAX
56      if n > N_MAX
57          break
58      end %if
59
60      %Increase the value of n
61      n=n+1;
62
63      %Set vOLD = v for the next loop iteration
64      vOLD = v;
65  end %while
66
67  %Multiply by the final 4/pi
68  v = v*4/pi;
69  end
```

B.3 Error Analysis

```
1  %TITLE: ERROR ANALYSIS FOR MOW'S EQUATIONS
2  %AUTHOR: BRADFORD JOSEPH LAPSANSKY
3  %PURPOSE: THE PURPOSE OF THIS PROGRAM IS TO POINTWISE COMPARE ...
       THE VALUES OF
4  %         e AND gamma IN BOTH THE ANALYTICAL AND NUMERIC ...
       SOLUTION AND TO
5  %         COMPUTE THE RELATIVE AND ABSOLUTE ERROR BETWEEN THESE TWO
```

```
6  %          SOLUTIONS IN ORDER TO DETERMINE THE ACCURACY OF THE ...
       NUMERIC
7  %          SOLVER.
8  %
9  %
10
11 clc
12 clear all
13 %%
14 %DISPLAY DIRECTORY INFORMATION:
15 fprintf('Current Directory = ');
16 disp(pwd);
17
18 %DISPLAY CURRENT RESULTS FILES IN CURRENT DIRECTORY
19 ls RESULTS_ANALYTIC*
20 fprintf('\n');
21
22 %IMPORT DATA:
23 fprintf('\n');
24 ls RESULTS_NUMERIC*
25 fprintf('\n\n');
26
27
28 clc
29
30 %%
31 %LOAD DATA
32 aFilename = 'RESULTS_ANALYTIC_MALAKPOOR.mat';
33 nFilename = 'RESULTS_NUMERIC_MALAKPOOR.mat';
34
35 aFile = load(aFilename);
36 nFile = load(nFilename);
37
38 %OBTAIN THE NAME OF THE AUTHOR OF THE ORIGINAL DATA SET:
39 AUTHOR = aFile.AUTHOR;
40 %AUTHOR = 'MALAKPOOR_TEST';
41
42 %SET VARIABLES OF THE ANALYTICAL SOLUTION USING A "_a" FLAG:
43 vArr_a= aFile.vArr;
44 zArr_a = aFile.zArr;
45 tArr_a = aFile.tArr;
```

```
46
47 %SEPARATE vArr_a INTO e AND gamma
48 vArr_a = transpose(vArr_a);
49 e_a = vArr_a(:,1);
50 g_a = vArr_a(:,2);
51
52 %SET VARIABLES OF THE ANALYTICAL SOLUTION USING A "_n" FLAG:
53 e_n = nFile.e;
54 g_n = nFile.gamma;
55 dz_n = nFile.dz;
56 dt_n = nFile.dt;
57 nt_n = nFile.nt;
58 nz_n = nFile.nz;
59
60 %SET THE POINT OF z IN WHICH YOU WANT TO COMPARE OVER
61 %THE DESIRED TIME DOMAIN
62 z_Reach = nFile.z_Reach;
63
64 %FIX zReach SO THAT IT IS A WHOLE NUMBER
65 zReach = z_Reach/dz_n;
66
67 %%
68 %CHECK TO MAKE SURE THAT THE TIMESTEPS IN BOTH SOLUTIONS IS ...
      EQUAL IN
69 %     ORDER TO PROPERLY, POINTWISE, COMPARE BOTH SOLUTIONS
70 tLen_a = length(tArr_a);
71 tLen_n = nt_n;
72
73 %%
74 if tLen_a ≠ tLen_n
75     fprintf('The discreet number of time points is not equal, \n');
76     fprintf('    this program will not run.');
77 else  %CALCULATE ERRORS IN e AND gamma
78     tLen = tLen_n;   %SHORTEN THE NAME OF tLen FOR CONVINCE
79
80     %CALCULATE THE SIZE OF e (ALSO EQUAL TO THE SIZE OF gamma)
81     sizeNum_n = size(e_n);
82     zLen_n = sizeNum_n(1);
83
84     %CALCULATE ERRORS IN e (DENOTED BY "_e" VARIABLE NAME ENDING):
```

```
85      [relV_e,relS_e,absV_e,absS_e]=CALCULATE_ERROR(e_a, e_n, ...
            tLen, zReach);
86
87      %CALCULATE ERRORS IN gamma (DENOTED BY "_g" VARIABLE NAME ...
            ENDING):
88      [relV_g,relS_g,absV_g,absS_g]=CALCULATE_ERROR(g_a, g_n, ...
            tLen, zReach);
89
90      %COMBINE THE e AND gamma REL AND ABS ERROR VECTORS INTO A ...
            MATRIX FOR
91      %     EASY PLOTTING
92      relV(:,1) = relV_e;  %COLUMN 1 IS e
93      relV(:,2) = relV_g;  %COLUMN 2 IS gamma
94
95      absV(:,1) = absV_e;  %COLUMN 1 IS e
96      absV(:,2) = absV_g;  %COLUMN 2 IS gamma
97
98
99      %DISPLAY THE ERROR SUMS ON THE SCREEN:
100     fprintf('Point of Interest on the Spatial Grid ...
            (z_measured/L): %4.3f \n', z_Reach/(dz_n*nz_n))
101     fprintf('\n\n----------SUM OF ERRORS IN ...
            "e"----------\n');
102     fprintf('RELATIVE: %10.6f', max(relV_e));
103     fprintf('\n');
104     fprintf('ABSOLUTE: %20.6f', max(absV_e));
105
106     fprintf('\n\n----------SUM OF ERRORS IN ...
            "gamma"----------\n');
107     fprintf('RELATIVE: %10.6f', max(relV_g));
108     fprintf('\n');
109     fprintf('ABSOLUTE: %20.6f', max(absV_g));
110     fprintf('\n\n')
111
112     fprintf('\n\n----------MAX ERRORS IN "e"----------\n');
113     fprintf('RELATIVE: %10.6f', max(relV_e));
114     fprintf('\n');
115     fprintf('ABSOLUTE: %20.6f', max(absV_e));
116     fprintf('\n\n')
117
```

```
118     fprintf('\n\n----------MAX ERRORS IN ...
            "gamma"----------\n');
119     fprintf('RELATIVE: %10.6f', max(relV_g));
120     fprintf('\n');
121     fprintf('ABSOLUTE: %20.6f', max(absV_g));
122     fprintf('\n\n')
123
124
125     %DISPLAY A TABLE OF POINTWISE RELATIVE ERRORS
126     FILENAME = strcat('ERROR_DATA_',AUTHOR);
127
128 %     excelFilename = strcat(FILENAME,'.xlsm');
129
130     %FILE WRITTEN WITH COLUMNS AS:
131     %
132     %zArr.....tArr.....rel_error_e.....rel_error_gamma
133 %     xlswrite(excelFilename, xlDisplay,strcat('Error_in_', ...
        AUTHOR),COLS);
134
135     %SAVE DATA AS A .mat FILE
136     save(FILENAME,'AUTHOR', 'zArr_a', 'tArr_a', ...
            'relV_e','relV_g','absV_e','absV_g', 'nt_n', 'dt_n');
137
138     %SEND FUNCTIONS TO BE PLOTTED
139     %NOTE: z_Reach ISTHE ACTUAL VALUE (NOT PART OF THE NUMERIC ...
            ARRAY)
140     %       THAT WE ARE COMPARING OUR ANSWERS ON.
141     FUNCTION_PLOT(e_a, e_n, g_a, g_n, tArr_a, zReach, 0, ...
            z_Reach, dt_n);
142 end  %IF
143
144 fprintf('\n\n END PROGRAM \n');
```

```
1 function ...
      [relV,relS,absV,absS]=CALCULATE_ERROR(res_a,res_n,tLen,zReach)
2 %CALCULATE_ERROR: THIS FUNCTION COMPARES THE VALUES IN BOTH e ...
      AND gamma
3 %                   FOR BOTH THE ANALYTICAL AND NUMERIC SOLUTION ...
      AND RETURNS
4 %                   AN ARRAY OF RELATIVE ERRORS IN e AND gamma ...
```

```
    ALONG WITH THE
%                 SUM OF THE RELATIVE ERRORS IN BOTH e AND gamma
%

%%
%CALCULATE THE RELATIVE ERROR BETWEEN THE ANALYTIC AND NUMERIC ...
    SOLUTION
%    AT z = zReach (z-VALUE AT WHICH THE FUNCTION IS BEING ...
    COMPARED)

%DECLARE THE ERROR VECTORS
relV = zeros(tLen,1);   %POINTWISE RELATIVE ERROR
absV = relV;            %POINTWISE ABSOLUTE ERROR
relS = 0;               %SUM OF REL. ERRORS
absS = 0;               %SUM OF ABSOLUTE ERRORS

for i = 1:tLen
    %POINTWISE ERRORS
    relV(i,1) = res_n(zReach, i) - res_a(i);
    absV(i,1) = abs(relV(i,1)/res_n(zReach,i));

    %SUM OF ERRORS:
    relS = relS + abs(relV(i,1));
    absS = absS + absV(i,1);
end %for

end
```

```
function []=FUNCTION_PLOT(e_a, e_n, g_a, g_n,tArrOrig, zReach,...
    figEnd, z_Reach, dt)
%FUNCTION_PLOT: PLOTS A COMPARISON OF THE ANALYSIS AND ...
    NUMERICAL SOLUTION
%              GRAPHS THAT APPEAR RIGHT OVER EACH OTHER:

%FORM COLUMN MATRICES OUT OF e, gamma, AND time DATUM:
e_n = transpose(e_n);
g_n = transpose(g_n);
tArr = transpose(tArrOrig);

%FORM MATRICES FROM THE INPUTS TO FIT THE PLOT ALGORITHM
```

```
12 ua(:,1) = e_a;
13 unFull(:,1) = e_n(:,zReach);
14 ua(:,2) = g_a;
15 unFull(:,2) = g_n(:,zReach);
16
17 %Limit the Display of the Numeric Solution
18 %    Determine the Order of Magnitude of a Time Step.
19 order = ORDER_MAGNITUDE(dt);
20 SKIP = 10^(2+abs(order));
21
22 i = 0;
23 for count=1:SKIP:length(tArr)
24     i=i+1;
25     for j=1:2
26         if count < length(tArr)
27             un(i, j) = unFull(count,j);
28         end
29     end
30     if count < length(tArr)
31         tArrSkip(i,1) = tArr(count);
32     else
33         break
34     end
35 end
36
37
38 for k=1:2
39     figure(k+figEnd)  %FIGURE NUMBER
40
41     %DETERMINE FIGURE TITLE
42     switch k
43         case 1
44             varName = 'Dilatation';
45         otherwise
46             varName = 'Gamma';
47     end %switch
48
49     %PLOT THE VARIOUS PLOTS:
50     %Analytic + Numeric
51     plot(tArr,ua(:,k),'k',tArrSkip,un(:,k),'xk');
52     xlabel('Time (s)');
```

```
53      strLabel = varName;
54      ylabel(strLabel);
55      legend('Analytic', 'Numeric');
56  end
57  end
```

```
1   %NAME: MAIN_PICK_ERROR
2   %AUTHOR: BRADFORD LAPSANSKY
3   %DESCRIPTION: ALLOWS THE USER TO PICK ERRORS FROM THE
4   % TRIPHASIC MODEL PROGRAM TO BUILD THE TABLE OF ERRORS
5   %
6   %
7   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8   clc
9   clear all
10  %%
11  %DISPLAY DIRECTORY INFORMATION:
12  fprintf('Current Directory = ');
13  disp(pwd);
14
15  %DISPLAY CURRENT RESULTS FILES IN CURRENT DIRECTORY
16  ls ERROR_DATA*
17  fprintf('\n');
18
19  %%
20  %LOAD DATA
21  FILENAME = 'ERROR_DATA_MALAKPOOR';
22  file = load(FILENAME); %Load Filename
23
24  %Load Saved Data
25  author = file.AUTHOR;
26  relVe = file.relV_e;
27  relVg = file.relV_g;
28  absVe = file.absV_e;
29  absVg = file.absV_g;
30  zarr = file.zArr_a;
31  tarr = file.tArr_a;
32  nt = file.nt_n;
33  dt = file.dt_n;
34
```

```
%%
%Chosen Times in an Array:
ctimes = [10, 20, 30, 100, 500, 1000, 1500, 2000, 3000, 3600];

%Find the Errors at These Time-Staps
format shortEng

%Perform an Interpolation Query for the correct time
rele = interp1(tarr,relVe, ctimes);
relg = interp1(tarr,relVg, ctimes);
abse = interp1(tarr,absVe, ctimes);
absg = interp1(tarr,absVg, ctimes);

xlDisplay(:,1) = ctimes';
xlDisplay(:,2) = rele';
xlDisplay(:,3) = relg';
xlDisplay(:,4) = abse';
xlDisplay(:,5) = absg';

%Save as an Excel File:
COLS = strcat('A2:E',num2str(length(ctimes)+1));
xlFilename = strcat('SELECTED_ERROR_DATA_',author,'.xlsm');
xlswrite(xlFilename, xlDisplay,'Selected_Error',COLS);

fprintf('\nEnd Program\n');
```

Bibliography

Hervé Abdi. The Eigen-Decomposition: Eigenvalues and Eigenvectors. In *Encyclopedia of measurement and statistics*, pages 304–308. 2007. URL http://ftp.utdallas.edu/~herve/Abdi-EVD2007-pretty.pdf.

P J Basser. Interstitial Pressure, Volume, and Flow during Infusion into Brain Tissue. *Microvascular research*, 44(2):143–65, September 1992. ISSN 0026-2862. URL http://www.ncbi.nlm.nih.gov/pubmed/1474925.

Max R Bennett, Les Farnell, and William G Gibson. A quantitative model of cortical spreading depression due to purinergic and gap-junction transmission in astrocyte networks. *Biophysical journal*, 95(12): 5648–60, December 2008. ISSN 1542-0086. doi: 10.1529/biophysj.108.137190. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2599846\&tool=pmcentrez\&rendertype=abstract.

Raymond Bowen. *Porous elasticity*. 2010. URL http://repository.tamu.edu/handle/1969.1/2500/browse?value=Bowen\%2C+Ray+M.\&type=author.

RM Bowen. Theory of Mixtures. In *Continuum Physics Volume III*. 1976. URL http://link.springer.com/chapter/10.1007/978-94-015-9327-4_2http://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:Continuum+Physics\#6.

RM Bowen. Incompressible porous media models by use of the theory of mixtures. *International Journal of Engineering Science*, 18:1129–1148, 1980. URL http://www.sciencedirect.com/science/article/pii/0020722580901147.

Joshua C Chang, Kevin C Brennan, Dongdong He, Huaxiong Huang, Robert M Miura, Phillip L Wilson, and Jonathan J Wylie. A mathematical model of the metabolic and perfusion effects on cortical spreading depression. *PloS one*, 8(8), January 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0070469. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3743836\&tool=pmcentrez\&rendertype=abstract.

Chapra, Stephen C. and Canale, Raymond P. *Numerical Methods for Engineers*. McGraw-Hill, New York, NY, sixth edition edition, 2010.

Peter Dayan and L.F. Abbott. *Theoretical Neuroscience Computational and Mathematical Modeling of Neural Systems*. The M.I.T Press, London, England, 2001.

Corina S Drapaca and Jason S Fritz. A Mechano-Electrochemical Model of Brain Neuro-Mechanics: Application to Normal Pressure Hydrocephalus. *International Journal of Numerical Analysis and Modeling, Series B*, 3(1):82–93, 2012.

Benjamin S Elkin, Mohammed A Shaik, and Barclay Morrison. Fixed negative charge and the Donnan effect: a description of the driving forces associated with brain tissue swelling and oedema. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 368 (1912):585–603, February 2010. ISSN 1364-503X. doi: 10.1098/rsta.2009.0223. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2944388\&tool=pmcentrez\&rendertype=abstract.

G. Bard Ermentrout and David H. Terman. The Hodgkin-Huxley Equations. In *Mathematical Foundations of Neuroscience*, volume 35 of *Interdisciplinary Applied Mathematics*. Springer New York, New York, NY, 2010. ISBN 978-0-387-87707-5. doi: 10.1007/978-0-387-87708-2. URL http://link.springer.com/10.1007/978-0-387-87708-2.

B Grafstein. Mechanism of Spreading Cortical Depression. *Journal of neurophysiology*, 19(2):154–171, 1956.

W Y Gu, W M Lai, and V C Mow. A mixture theory for charged-hydrated soft tissues containing multi-electrolytes: passive transport and swelling behaviors. *Journal of biomechanical engineering*, 120(2):169–80, April 1998. ISSN 0148-0731. URL http://www.ncbi.nlm.nih.gov/pubmed/10412377.

WY Gu, WM Lai, and VC Mow. Transport of multi-electrolytes in charged hydrated biological soft tissues. *Transport in Porous Media*, 34:143–157, 1999. URL http://link.springer.com/article/10.1023/A:1006561408186.

Morton Gurtin, Eliot Fried, and Lallit Anand. *The Mechanics and Thermodynamics of Continua*. Cambridge University Press, New York, NY, 2010.

Douglas Wilhelm Harder. The Crank-Nicolson Method and Insulated Boundaries, 2012. URL https://ece.uwaterloo.ca/~math212/Laboratories/03/3.CrankNicolson.pptx?

AL Hodgkin and AF Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. *The Journal of physiology*, 116:449–472, 1952a. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392213/.

AL Hodgkin and AF Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117:500–544, 1952b. URL http://www.ncbi.nlm.nih.gov/pmc/articles/pmc1392413/.

Irwin, J. David and Nelms, R. Mark. *Basic Engineering Circuit Analysis*. John Wiley & Sons, tenth edition edition, 2011.

Mohamed Iskandarani. Chapter 11 Finite Difference Approximation of Derivatives, 2010.

Eric R. Kandel, James H. Schwartz, Thomas M. Jessell, Steven A. Siegelbaum, and A. J Hudspeth. *Principles of Neural Science*. 2012.

Ramdas Kumaresan. Fourier Series, 2012.

W M Lai, J S Hou, and V C Mow. A triphasic theory for the swelling and deformation behaviors of articular cartilage. *Journal of biomechanical engineering*, 113 (3):245–58, August 1991. ISSN 0148-0731. URL http://www.ncbi.nlm.nih.gov/pubmed/1921350.

A.A.P. Leao. Spreading depression of activity in the cerebral cortex. *Journal of Neurophysiology*, 7(6):359–390, 1944. URL http://psycnet.apa.org/psycinfo/1945-01398-001.

David R. Lide. *CRC Handbook of Chemistry and Physics, 88th Edition (CRC Handbook of Chemistry & Physics)*. CRC Press, 88 edition, June 2007. ISBN 0849304881. URL http://www.worldcat.org/isbn/0849304881.

Xin L Lu, Leo Q Wan, X Edward Guo, and Van C Mow. A linearized formulation of triphasic mixture theory for articular cartilage, and its application to indentation analysis. *Journal of Biomechanics*, 43(4):673–9, March 2010. ISSN 1873-2380. doi: 10.1016/j.jbiomech.2009.10.026. URL http://www.ncbi.nlm.nih.gov/pubmed/19896670.

K. Malakpoor, E.F. Kaasschieter, and J.M. Huyghe. An analytical solution of incompressible charged porous media. *Zamm*, 86(9):667–681, September 2006. ISSN 0044-2267. doi: 10.1002/zamm.200510269. URL http://doi.wiley.com/10.1002/zamm.200510269.

Georgi Medvedev. MATH 680: Introduction to Computational Neuroscience, 2005. URL http://www.math.drexel.edu/~medvedev/classes/2005/math680/.

T P Obrenovitch and E Zilkha. High extracellular potassium, and not extracellular glutamate, is required for the propagation of spreading depression. *Journal of neurophysiology*, 73(5):2107–14, May 1995. ISSN 0022-3077. URL http://www.ncbi.nlm.nih.gov/pubmed/7623102.

G.D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, 3rd edition edition, 1986.

D N Sun, W Y Gu, X E Guo, W M Lai, and V C Mow. A Mixed Finite Element Formulation of the Triphasic Mehano-Electrochemical Theory for Charged, Hydrated Biological Soft Tissue. *International Journal for Numerical Methods in Engineering*, 1402(January 1998):97–119, 1999.

I Tasaki and K Iwasa. Further Studies of Rapid Mechanical Changes in Squid Giant Axon Associated with Action Potential Production. *The Japanese journal of physiology*, 32:505–518, 1981. URL `http://europepmc.org/abstract/MED/7176207`.

The MathWorks Inc. MATLAB and Statistics Toolbox Release 2013b, 2013.

Thomas F. Weiss. *Cellular Biophysics Volume 1 Transport*. The M.I.T Press, London, England, 1996.

Xin-Cheng Yao, David M Rector, and John S George. Optical lever recording of displacements from activated lobster nerve bundles and Nitella internodes. *Applied optics*, 42(16):2972–8, June 2003. ISSN 0003-6935. URL `http://www.ncbi.nlm.nih.gov/pubmed/12790447`.

Academic Vita
Bradford Joseph Lapsansky

E-Mail:	bradjlap@comcast.net
Address:	16 Hilldale Ave. Plains, PA 18705

Education

- B.S. (Honors) in Engineering Science, *Student Marshall*
  Minor in Engineering Mechanics
  The Pennsylvania State University (University Park, PA), May 2014
  Scholar in the Schreyer Honors College

Presentations

- B. J. Lapsansky and C. S. Drapaca. A Model of Brain Neuro-Mechanics. Abstract accepted to the SIAM Annual Meeting. Chicago, IL. July 2014.

Professional Experience

- PPL Susquehanna LLC - Berwick, PA
  In-Service Inspection Cooperative Associate (May 2013 – Aug. 2013)
  - Created a Program that Collected and Summarized Inspection Results for the PPL Susquehanna Nuclear Plant
  - Assisted with an Institute of Nuclear Power Operations Jet Pump Review
- Pride Mobility Products - Duryea, PA
  Manufacturing Engineering Intern (May 2012 – Aug. 2012)
  - Designed and Built an Electronics Testing Fixture for the "Maxima" Power Scooter which Saved the Company more than $1400
  - Designed and Built a Battery Storage Cart for the "Maxima" Scooter to Reduce Ergonomic Issues and Decrease Wasted Time
  - Analyzed and Made Suggestions for the Proper Placement of the anti-tip brackets on the "Litestream" Manual Wheelchair

Honors and Awards

- S.M.A.R.T. Scholarship – Awarded Aug. 2013
- Tau Beta Pi Record Scholarship – Awarded Jul. 2013
- Robert and Myrtle Vierck Scholarship – Awarded Jul. 2013
- Evan Pugh Scholar Award – Awarded April 2013
- Sam Y. and Myrna R. Zamrik Scholarship – Awarded Jul. 2012

Association Memberships/Activities

- Tau Beta Pi (Pennsylvania *Beta* Chapter)
  Member: Dec. 2012 – Present
- Penn State Society of Engineering Science
  Member: Aug. 2012 – May 2014
  Treasurer: Aug. 2013 – May 2014
  - Managed the Society's Website
  - Managed Club Funding
- Penn State Wilkes-Barre Honor Society
  Member: Aug. 2010 – May. 2011
- Penn State Wilkes-Barre Blue and White Society
  Member: Aug. 2010 – Dec. 2011
  President: Aug. 2011 – Dec. 2011
  Vice-President: Jan. 2011 – Aug. 2011
  - Helped Plan and Organize a "Blue and White" Tailgate
  - Initiated Preparations on the "Blue and White Ball"

Skills

- C++
- Advanced Excel
- VBA
- MATLAB
- LaTeX