Parameter Reduction of the Hodgkin-Huxley Model of Action Potential Propagation

Tyler A. Bahr

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Mark K. Transtrum, Advisor

Department of Physics and Astronomy

Brigham Young University

April 2018

Copyright © 2018 Tyler A. Bahr

All Rights Reserved

ABSTRACT

Parameter Reduction of the Hodgkin-Huxley Model
of Action Potential Propagation

Tyler A. Bahr
Department of Physics and Astronomy, BYU
Bachelor of Science

In 1952 Hodgkin and Huxley formulated the fundamental biophysical model of how neurons integrate input and fire electric spikes. With 25 parameters and 4 dynamical variables, the model is quite complex. Using information theory, we analyze the model complexity and demonstrate that it is unnecessarily complex for many neural modeling tasks. Using the manifold boundary approximation method of model reduction, we perform a series of parameter reductions on the original 25-parameter model and create a series of spiking Hodgin-Huxley models, each with decreasing parameter number. We analyze the physical meaning of some key approximations uncovered by our systematic reduction methods, which are "blind" to the real physical processes the model is intended to capture. We then evaluate the behavior of the most greatly reduced 14-parameter model under different experimental conditions, including networks of neurons. We also discuss new questions that have arisen as a result of our work.

Keywords: Hodgkin-Huxley, action potential, model reduction, neuron model, manifold boundary approximation method

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Mark Transtrum, whose excellent mentoring gave me the resources and the skills to be able to complete this project. I also wish to thank the BYU ORCA office, the BYU department of Physics and Astronomy, and the Jessie Allen Werner Fund for their generous contributions that made this research possible.

Contents

Chapter 1

Introduction

1.1 A General Introduction to Modeling

1.1.1 Mathematical Models

All scientific study is based upon the idea that our physical world is governed by an unwavering set of rules. Early philosophers realized that learning these rules would make it possible to predict outcomes and events, ultimately giving man power over nature. Early physicists like Newton discovered that fundamental phenomena in our world, such as energy or momentum, could be explained using simple algebraic equations. While Newton's findings were revolutionary, modern physicists have discovered that Newtonian mechanics are actually excellent approximations of a more accurate (and complicated) theory involving relativity and quantum mechanics. Newton's equations are quite useful nonetheless for situations where large objects are not moving near light speed. While they might not be completely correct, his equations are still hailed for their practical utility.

Newton's equations (and their relativistic counterparts) are both examples of mathematical models. Any equation or set of equations that describes a physical process can be considered a model. The field of modeling is concerned with providing mathematical descriptions of real-world phenomena. Descriptions of a process in the real world are not necessarily confined to a single model, however, as we see in the case of energy and momentum. Multiple models can describe a single process. A dichotomy emerges with some models leaning toward describing the process in detail, and others toward capturing its overall behavior. Detailed models can be very cumbersome because they aim to explain even the most minuscule aspects of a process. These models are usually described as "mechanistic." Phenomenological models, on the other hand, aim to only capture the general patterns of a system's behavior. Therefore, a key aspect of modeling lies in choosing the complexity and accuracy desired in a model.

1.1.2 Parameters and Model Complexity

A parameter is a numerical value that describes one specific physical quantity involved in a model. Any specific set of parameter values corresponds to one possible configuration of the physical system being modeled. The equation for the period of a pendulum is a simple example of a model and its parameters:

$$T = \sqrt{\frac{g}{L}}. \tag{1.1}$$

This equation alone could describe the period of any pendulum. String length L is a parameter in this model, because it will take on distinct values for different pendulums. Gravitational acceleration is a paramater as well, because it could vary depending on the location of the experiment. The mathematical form of the model remains the same for all pendulums, however, no matter the values chosen for the parameters.

One way of measuring the complexity is to simply count the total number of parameters in a model. Generally, more parameters indicate a more cumbersome mathematical structure. Complex models tend to require more computational power to make predictions.

1.1.3 Mathematical Formalism for Analyzing Model Complexity

The main idea of mathematical modeling is to find some function that can reproduce observed behavior within a certain margin. Because there is often a trade-off between accuracy and complexity, our mathematical formalism is based on measures which quantify how well a model fits experimental data. This section summarizes a geometric approach to modeling described elsewhere [1].

Nonlinear least squares is a common method for calibrating a model to data and quantifying its accuracy. Given an independent variable t sampled at a set of points $\{t_m\}$ with observed behavior $\{y_m\}$, a model is a parameterized function $f(t_m, \theta)$. We seek the values of the parameters, θ, such that the model predictions $\{f(t_m, \theta)\}$ are as close as possible to the observations. We introduce the residuals:

$$r_m = y_m - f(t_m, \theta). \tag{1.2}$$

Weights $\{\sigma_m\}$ can be assigned to individual data points. In this case, the residuals become:

$$r_m = \frac{y_m - f(t_m, \theta)}{\sigma_m}. \tag{1.3}$$

The sum of squares of residuals is a metric that quantifies deviation of the model $\{f(t_m, \theta)\}$ from the observed behavior $\{y_m\}$, a quantity known as the cost:

$$C(\theta) = \frac{1}{2}\sum_m r_m(\theta). \tag{1.4}$$

With these definitions in place, we now move on to discuss a geometric interpretation of models in an abstract but general way. For the remainder of this discussion, we use for example a toy model with two parameters used to fit three data points (Figure 1.1a). The model has the form

$$y(t, \theta) = e^{-t\theta_1} + e^{-t\theta_2}. \tag{1.5}$$

The geometric interpretation of mathematical models we present here involves the concept of two multi-dimensional spaces. One space, parameter space (Figure 1.1b), is the set of all possible values that the model parameters can take. It has dimensionality equal to the number of parameters in the model, $N = dim\{\theta\}$. A point in parameter space represents one particular set of parameter values. The second space, data space, represents the possible observations of the model. It has dimensionality equal to the number of observations, $M = dim\{y_m\}$. A single point in data space is one possible set of observations.

The set of all possible predictions generated by the model from all possible parameter values forms a manifold, or multi-dimensional set of points, that is a subset of data space (Figure 1.1c). The dimensionality of this set is equal to N, the number of structurally identifiable parameters in the model and the dimensionality of parameter space. Points in data space outside of the model manifold correspond to observations that cannot be fit by the model for any values of parameters. Typically, model manifolds have boundaries or edges, as in Figure 1.1c. The edges of the manifold are the limits that the predictions approach as parameters approach infinitely large or small values.

The shape of the model manifold is a consequence of the mathematical form of the model. Geometric analysis of this manifold, as well as examination of how data space and parameter space are related, can give valuable insights into the complexity of a model. For example, consider the Jacobian matrix which contains derivatives of the residuals with respect to the parameters:

$$J_{m\mu} = \frac{\partial r_m}{\partial \theta_\mu} \equiv \partial_\mu r_m. \tag{1.6}$$

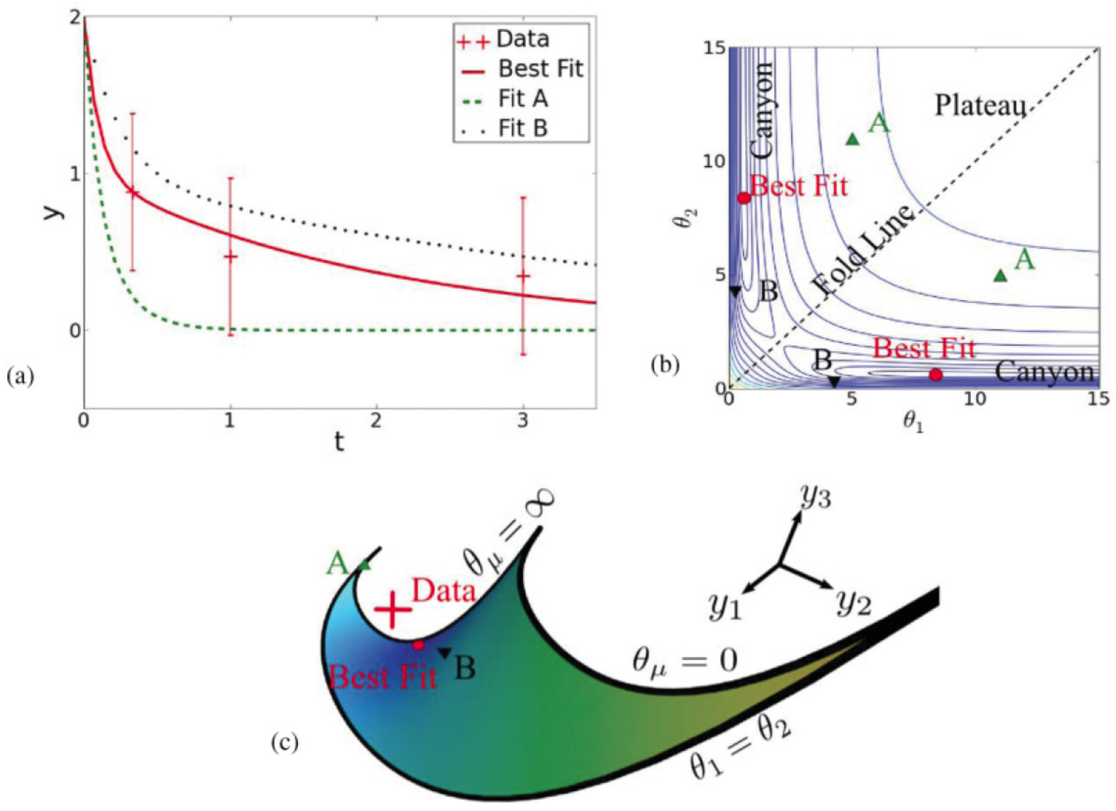

Figure 1.1 A geometric interpretation of mathematical modeling. **(a) Experimental data and model predictions.** Fit A has rate constants that decay too quickly, resulting in a poor fit; B is an improvement over fit A, although the rates are too slow; the best fit minimizes the computational cost. **(b) Parameter Space.** Contours of constant cost are shown. Note the "plateau" in the region of large rates where the model is essentially independent of parameter changes. Note also the long, narrow canyon at lower rates, characteristic of a sloppy model. Points A and B represent two different sets of parameter values for θ_1 and θ_2. **(c) Model predictions in data space.** The experimenal data are represented by a single point. The set of all possible fitting parameters induces a manifold of predictions in data space. The best fit is the point on the manifold nearest to the data. The plateau in (b) here is the small region around the short cusp near the corner. [1]

Each index m corresponds to one data point and each index μ corresponds to one of the parameters. The Jacobian matrix has size $M \times N$. Each entry in the Jacobian matrix is the partial derivative of one residual with respect to one parameter. This matrix connects changes in parameter space with changes in data space. For example, a small change in parameters $\Delta\theta$ representing a short path in parameter space could be multiplied by this matrix to give a short path in data space, indicating the rate of change in data space caused by a change in parameter values:

$$\Delta r_m = \sum_{\mu} J_{m\mu}\, \Delta\theta_{\mu}\ . \tag{1.7}$$

The total change in the residuals is given by

$$\sum_m {\Delta r_m}^2 = \sum_{\mu,\nu,m} \Delta\theta_\nu\ J_{m\nu}\ J_{m\mu}\ \Delta\theta_\mu. \tag{1.8}$$

$$= \sum_{\mu,\nu,m} \Delta\theta_\nu\ (J^T J)_{\mu\nu}\ \Delta\theta_\mu. \tag{1.9}$$

It is therefore convenient to define the matrix

$$I = J^T J, \tag{1.10}$$

which is an $N \times N$ square matrix usually known as the Fisher Information Matrix (FIM). The Fisher information matrix is a statistical object that measures the sensitivity of a model's predictions to variations in parameters. The eigenvalues of this matrix are of great significance for analyzing parameter identifiability and sloppiness, two important concepts which will be discussed in the next two sections.

1.1.4 Parameter Identifiability

Parameter identifiability considers the question: is it possible to infer the values of all model parameters from data? Often, only combinations of parameters are identifiable from experimental data. When parameters can be varied in certain combinations that give exactly the same model behavior, they are known as structurally unidentifiable parameters. Structurally unidentifiable parameters correspond to the null space of the FIM. [1]

When a combination of parameters can be varied in a manner that gives approximately the same model behavior, the parameters are practically unidentifiable. We can find practically unidentifiable parameter combinations using the eigenvectors of the FIM. An eigenvector with a small eigenvalue indicates the presence of a practically unidentifiable combination of parameters. On the other hand, eigenvectors of the FIM with large eigenvalues indicate identifiable parameters whose effect on model behavior is quite clear.

Since they are numerical approximations, it is not straightforward to remove practically unidentifiable combinations from a model. We use a method known as the Manifold Boundary Approximation Method (MBAM) to identify and remove practically unidentifiable parameters from models [1]. This method will be discussed in more detail in Chapter 2.

1.1.5 Sloppy models

"Sloppy" models are models with many practically unidentifiable parameter combinations. These models are often characterized by cumbersome equations and can sometimes have even hundreds (or more) parameters. Sloppy models are common in systems biology and physics [1] and we have begun to think that they are also common in the field of neuroscience.

Recall from section 1.1.3 that each dimension in parameter space is associated with one parameter. Thus, moving through parameter space in one dimension corresponds to varying the value of one parameter. General paths in parameter space may not align with the parameter space axes. Such paths simultaneously vary multiple parameters together.

There are certain paths in parameter space that can be traversed without significantly affecting model predictions. The directions of those paths are called "sloppy" directions. On the other hand, paths that result in large changes in model predictions are known as "stiff" directions. Take for example the parameter space and cost surface shown in Figure 1.1b. The sloppy direction is parallel to the canyon. The cost remains more or less constant when moving through parameter space in that direction, indicating that the predictions do not change. The stiff direction is up the canyon wall. The cost changes rapidly in this direction, indicating significant changes in predictions.

Sloppy directions can sometimes exist in a single parameter dimension, but in many sloppy models they often exist in complex multi-dimensional directions involving combinations of parameters. Eigenvectors of the FIM with small eigenvalues align with the sloppy directions of the model. Models with many small eigenvalues are considered sloppy models (Figure 1.2).

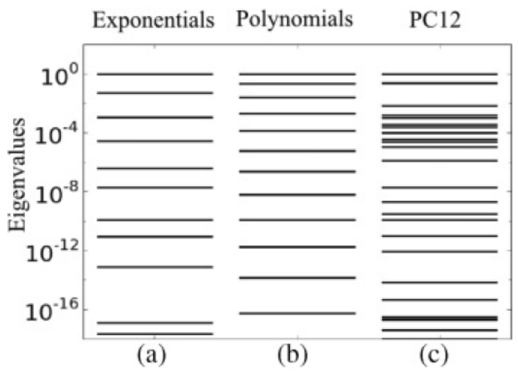

Figure 1.2 Eigenvalues of FIMs for three sloppy models. (a) A model formed by summing six exponential terms with rates and amplitudes. (b) A polynomial model. (c) A systems biology model of epidermal growth factor signaling in rat pheochromocytoma (PC12) cells. [1]

With the help of statistical objects like the FIM and other computational tools, we find sloppy directions in data space and use them to identify the corresponding combinations of parameters that do not affect a model's behavior. This in turn allows for the reduction (simplification) of models, one of the major focuses of this thesis.

1.2 Complex Modeling in Neuroscience

1.2.1 The Role of Complexity in Neural Models

The brain is a vast network composed of around 85 billion of neurons with over 100 trillion synaptic connections between them all [2]. The computational cost of modeling the entire brain is impossibly large using computational tools available today. For example, in 2014 Japanese researchers used the K computer (at that time the fourth most powerful in the world) to run a simulation using the NEST spiking neural network tool [2]. Their network included a combination of detailed and simplified neuron models, ranging from full Hodgkin-Huxley-type models down to basic integrate-and-fire models. Despite its 1.4 million GB of RAM and 705,024 processor cores, it still took the K computer 40 minutes of computation time to simulate a single second of neural activity. Although the model network included nearly 2 billion model neurons and 10 trillion connections, it represented only one percent of the neuronal network that makes up the entire human brain. From this example, it is clear that complexity is a major obstacle in the field of neuroscience. Unfortunately, many computational neuroscientists have resigned to putting their computationally expensive projects on hold and simply waiting for computer developers to build faster computers. However, there are other approaches. It might be possible to reduce the computational cost of neural simulations by removing unnecessary information from the neuron models themselves. In physics, models that contain few, relevant degrees of freedom are called "effective theories."

1.2.2 Parameters and Fundamental Theories

In 1952 Lord Alan Hodgkin and Andrew Huxley formulated a biophysical model to describe how neuron cell membranes exhibit voltage spiking [3]. In a neuron's membrane, ion channels act as non-ohmic (voltage-dependent) resistors that allow current to pass only above a voltage threshold. The passage of electic current through these channels further increases the voltage across the cell membrane for a short time until the channels are inactivated and reset. While conceptually simple, this process is explained mathematically using a set of four coupled differential equations with a whopping total of 25 parameters.

In contrast to the rather cumbersome model created by Hodgkin and Huxley, fundamental theories in the fields of physics and mathematics are usually simple models that depend on just a few parameters. Just 5 years after Hodgkin and Huxley's work, the physicists Bardeen, Cooper, and Schrieffer (BCS) proposed their model of superconductivity for which they won a Nobel prize. In stark contrast to the neural theory proposed by Hodgkin and Huxley, the BCS theory depended on only one parameter: the critical temperature [4]. As far as models go, the BCS theory is quite attractive because critical temperature is a parameter that is easy to measure and it is directly related to superconductor phenomenology. For instance, it would be easy to take experimental data from a superconductor sample and use it to deduce the value of the critical temperature for that material. The BCS model is known as a minimal mechanistic model. It is the simplest way to describe the behavior of superconductors that captures the essential mechanism at play.

Alan Hodgkin and Andrew Huxley (HH) also won a Nobel prize for their work. Their model has been used widely by neuroscientists despite it being a very cumbersome model with many parameters. Many of those parameters are quantities that are very difficult to measure and their values are not easily connected to the phenomenology. This raises a couple of key questions: Is the HH model truly a minimal mechanistic model? And are all of the parameters really necessary? Perhaps a simpler mathematical model could give the same predictions. We show that the behavior of the HH model is controlled by a few combinations of parameters, allowing for the possibility of simplification while still preserving its predictive power.

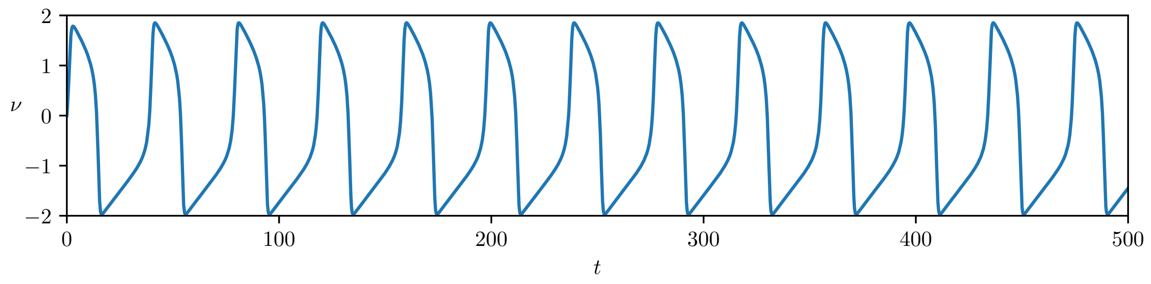

Figure 1.3 The Fitzhugh-Nagumo Model: Voltage response to a constant input current. Note that the time scale, parameters, and variables are all dimensionless. Parameter values: $a = 0.7$, $b = 0.8$, $\tau = 12.5$, $I = 0.5$.

1.2.3 Types of Neural Models

Others before us have realized that the complexity of the HH model is not necessary for most neurological modeling tasks. About ten years after the publication of the HH model, Fitzhugh and Nagumo (FN) proposed a simpler model of nerve excitation using a relaxation oscillator involving only two differential equations in dynamical variables ν and ω [5, 6]:

$$\frac{d\nu}{dt} = \nu - \frac{\nu^3}{3} - \omega + I \tag{1.11}$$

$$\tau\frac{d\omega}{dt} = \nu + a - b\omega. \tag{1.12}$$

The FN model is attractive because it exhibits spiking behavior above a threshold (Figure 1.3), and it involves only three parameters, τ, a, and b, with I representing the input current. The physical significance of a and b is unclear, however τ is known to be a time constant which determines the onset of spiking. While the FN model is useful for its spiking behavior, it fails to capture the bursting characteristic of many HH-type models. Bursting is a phenomenon in which groups of multiple spikes are rapidly fired in succession, with intervals of silence between the groups of spikes. In 1984 Hindmarsh and Rose (HR) developed another simplified neuron model to capture the bursting behavior of neurons (Figure 1.4) [7]. Their model involved three differential equations, eight parameters with blurry physical interpretations at best, and three dimensionless dynamical variables, x, y, and z [7]:

$$\frac{dx}{dt} = y - ax^3 + bx^2 - z + I \tag{1.13}$$

$$\frac{dy}{dt} = c - dx^2 - y \tag{1.14}$$

$$\tau\frac{dz}{dt} = s(x - xR) - z. \tag{1.15}$$

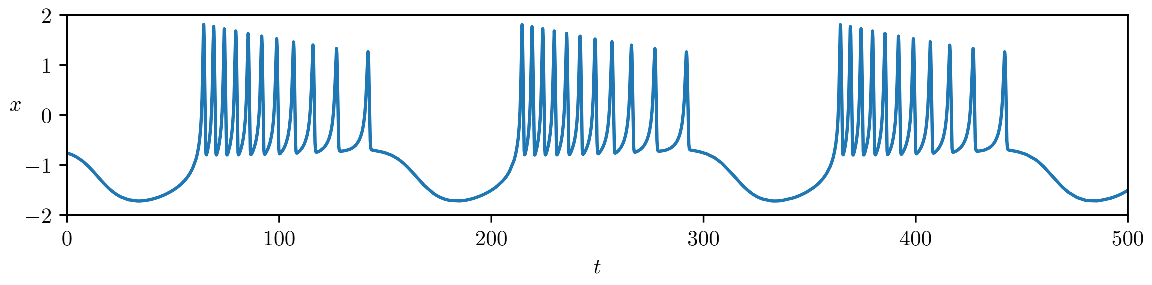

Figure 1.4 The Hindmarsh-Rose Model: Response to a constant input current. Note that the time scale, parameters, and variables are dimensionless. Parameter values: $a = 1$, $b = 2.7$, $c = 1$, $d = 5$, $\tau = 100$,$s = 4$, $xR = -1.6$, $I = 4$.

Both of these models are incredibly popular in neural computing, however they have a few substantial weaknesses. These include inaccurate spike shape, dimensionless parameters, and the necessity of experimental data for optimizing model fit. These models cannot be tweaked to simulate changes in experimental conditions, and the parameter values must be empirically determined. In fact, the parameters cannot be directly connected to real physiological quantities. Because these models were built from the ground up, the physical significance of the parameters in these models is unknown.

1.2.4 Using the Hodgkin-Huxley Model to Explore Parameter Reduction in Neuroscience

To summarize, the advantage of the Hodgkin-Huxley model is its versatility and accuracy in describing different experimental conditions. Its disadvantage is its complexity. Simplified neuron models are popular because they capture the behavior of the model with fewer parameters, making computation less expensive. However, they come with certain drawbacks, like loss of critical parameter information. We propose that this parameter information might be preserved if simplified models are generated from the top-down, rather than from the bottom-up. Model reduction might be the missing link between simplified neuron models and the real physiological quantities that govern their behavior.

In this work, we apply a model reduction method called the Manifold Boundary Approximation Method (MBAM) to the HH model. MBAM identifies identifiable parameter combinations which generally end up being products, quotients, or other similar (and usually simple) mathematical combinations of the original parameters. The availability of reduced neuron models with these properties would in principle give neuroscientists the ability to tweak their models and make predictions for different systems without empirically fitting data, that is, enable predictive modeling. This might save time, money, and other valuable resources.

While this work focuses on the HH model, it is more than a simple exploration of whether the the model of nerve excitation can be simplified. It is a preliminary investigation of the geometry of models in neuroscience, which are often based on some sort of neuron model. Ultimately, we hope that this project might serve as an indicator of how the field of neuroscience could benefit from information theory and systematic model reduction.

Chapter 2

Materials and Methods

2.1 The HH model

2.1.1 Structure of The Hodgkin-Huxley model

The Hodgkin-Huxley model describes how neuronal cell membranes exhibit voltage spiking behavior. In a neuron's membrane, ion channels act as non-ohmic (voltage-dependent) resistors that allow current to pass after a threshold is reached, which in turn leads to increased voltage across the cell membrane until biophysical interactions between domains of the channels leads to their inactivation and subsequent reset. This process is explained by a set of differential equations with one variable representing the voltage and other variables related to the gating properties of the ion channels. The utility of the Hodgkin-Huxley model lies in its ability to predict voltage as a function of time; knowing the values of the gating variables is usually of lesser importance.

The original Hodgkin-Huxley (HH) model is composed of four nonlinear coupled differential equations that describe four dynamical variables as a function of time: voltage V, the potassium ion channel gating constant n, the sodium channel activation constant m, and the sodium channel inactivation constant h. Other HH-type models are similar in form to this model, with additional equations describing the kinetics of other ion channels like calcium, chloride, etc.

In the original HH model, the equation for voltage V has the form

$$\frac{dV}{dt} = \frac{1}{C_m}(I_{app} - I_K - I_{Na} - I_L), \tag{2.1}$$

where

$$I_K = n^4 G_K (V - E_K) \tag{2.2}$$

$$I_{Na} = m^3 h G_{Na} (V - E_{Na}) \tag{2.3}$$

$$I_L = G_L (V - E_L) \; . \tag{2.4}$$

I_K, I_{Na}, and I_L represent the current passing through potassium, sodium, and leak channels, respectively. The physical significance of each of these parameters and variables in these equations is explained in Appendix C. The other three differential equations in the Hodgkin-Huxley model describe the probabilities of ion channels gates being in the open state, and are as follows:

$$\frac{dn}{dt} = \frac{n - n_\infty}{\tau_n} \tag{2.5}$$

$$\frac{dm}{dt} = \frac{m - m_\infty}{\tau_m} \tag{2.6}$$

$$\frac{dh}{dt} = \frac{h - h_\infty}{\tau_h}. \tag{2.7}$$

For an iterator i representing the three variables n, m and h, i_∞ and τ_i are given by

$$i_\infty = \frac{A_i}{A_i + B_i} \tag{2.8}$$

$$\tau_i = \frac{1}{A_i + B_i} \; . \tag{2.9}$$

Finally, A_i and B_i are both functions of V that have similar structures but vary slightly for each i.

$$A_n = \frac{\alpha_n (V - V_{\alpha_n})}{1 - e^{\frac{-(V - V_{\alpha_n})}{K_{\alpha_n}}}} \tag{2.10}$$

$$A_m = \frac{\alpha_m (V - V_{\alpha_m})}{1 - e^{\frac{-(V - V_{\alpha_m})}{K_{\alpha_m}}}} \tag{2.11}$$

$$A_h = \alpha_h \, e^{\frac{-(V - V_{\alpha_h})}{K_{\alpha_h}}} \tag{2.12}$$

$$B_n = \beta_n \, e^{\frac{-(V - V_{\beta_n})}{K_{\beta_n}}} \tag{2.13}$$

$$B_m = \beta_m \, e^{\frac{-(V - V_{\beta_m})}{K_{\beta_m}}} \tag{2.14}$$

$$B_h = \frac{\beta_h}{1 + e^{\frac{-(V - V_{\beta_h})}{K_{\beta_h}}}}. \tag{2.15}$$

2.1.2 Structural Identifiability in the HH Model

Recall from section 1.1.4 that structurally unidentifiable parameters can varied in certain combinations such that model behavior is unchanged. There is a prime example of structural identifiability in equation 2.14. It has one independent dynamical variable V and three parameters, β_m, V_{β_m}, and K_{β_m}. Using some algebra, the exponential expression can be split into a product of two terms,

$$B_m = \beta_m e^{\frac{-V_{\beta_m}}{K_{\beta_m}}} e^{\frac{V}{K_{\beta_m}}}. \tag{2.16}$$

The prefactor of the exponential is composed entirely of parameters. There are infinitely many values of those parameters that would give the same value for this expression. By the same token, it would be impossible to differentiate between the effects of those parameters from the data. This means that we can define a new parameter to take its place, which we will call $\tilde{\beta_m}$,

$$\tilde{\beta_m} = \beta_m e^{\frac{-V_{\beta_m}}{K_{\beta_m}}}. \tag{2.17}$$

This has the following result on the original function:

$$B_m = \beta_m e^{\frac{-V_{\beta_m}}{K_{\beta_m}}} e^{\frac{V}{K_{\beta_m}}} = \tilde{\beta_m} e^{\frac{V}{K_{\beta_m}}}. \tag{2.18}$$

Our new equation for B_m is mathematically identical to the original, except it depends on only 2 parameters now instead of 3. This process illustrates the fundamental idea behind model reduction using parameter combinations.

The equations for B_m and B_n are of the same form. Likewise, they can each be simplified as follows:

$$B_m = \tilde{\beta}_m \, e^{\frac{-V}{K_{\beta_m}}} \, , \tag{2.19}$$

and

$$B_n = \tilde{\beta}_n \, e^{\frac{-V}{K_{\beta_n}}} \, . \tag{2.20}$$

Applying these three algebraic simplifications brings the total number of HH parameters down to 22.

2.1.3 Practical Identifiability in the HH Model

Recall from section 1.1.2 that the presence of unidentifiable parameters correlates with model sloppiness; sloppy models usually have many unidentifiable parameters. We used the FIM to estimate the relevant number of parameters in the Hodgkin-Huxley model responding to a constant input current. We find that about two-thirds of the parameters in the Hodgkin-Huxley model are unnecessary in this case (Figure 2.1).

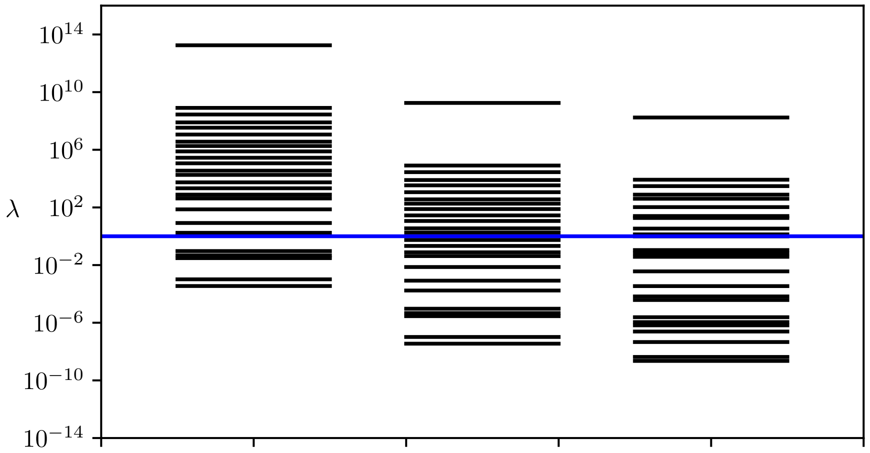

Figure 2.1 Eigenvalues of the Fisher Information Matrix for a spiking neuron. Eigenvectors with eigenvalue less than 1 correspond to irrelevant parameters. The blue line emphasizes the cutoff. Left: Model predictions for V, n, m, and h. Middle: Model predictions for V only. Right: Model predictions for V only at 1/10 the sampling frequency.

Before delving into the computational processes required to identify practically unidentifiable combinations, we discuss an example of an unidentifiable parameter in the HH model.

We identified a practically unidentifiable parameter, K_{α_m}, in the equation for A_m,

$$A_m = \frac{\alpha_m(V - V_{\alpha_m})}{1 - e^{\frac{-(V-V_{\alpha_m})}{K_{\alpha_m}}}}. \tag{2.21}$$

Our reductive approximation involved removing the parameter K_{α_m} and replacing the original function for A_m with a new function obtained by taking the limit as K_{α_m} approaches zero:

$$A_m \approx \alpha_m(V - V_{\alpha_m})\, \mathscr{H}(V - V_{\alpha_m}). \tag{2.22}$$

Although this new function does not require K_{α_m}, the behavior of the two functions over the experimentally relevant range of values is nearly identical (Figure 2.2). It would be difficult to notice the effect of changing K_{α_m} on the predictions of the HH model, especially when analyzing experimen-

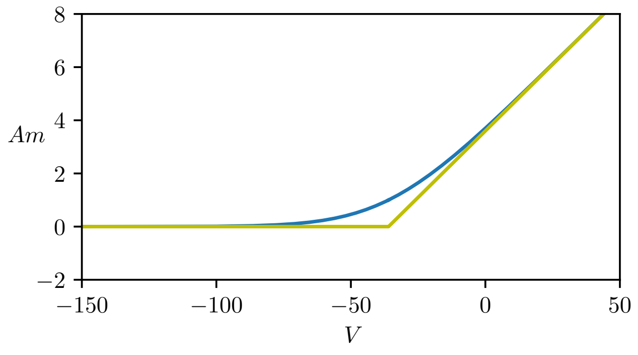

Figure 2.2 The original and reduced functions for A_m in blue and yellow, respectively. Voltage V has units of mV. A_m is a value with units of frequency in MHz.

tal data. In other words, the practically identifiable combination of parameters in the equation for A_m excludes K_{α_m}. In the next section we will discuss the computational methods used to identify this combination, and other practically unidentifiable parameter combinations in the HH model.

2.2 Model Reduction of HH Model using Manifold Boundaries

2.2.1 Outline of the Manifold Boundary Approximation Method

As described previously [8], model reduction by manifold boundaries uses the geometrical approach involving the model manifold. The manifold of many complex models is often very thin, like a ribbon. In this case, the thickness of the ribbon would usually be of little importance. For example, a three dimensional model manifold with one very thin dimension could be approximated as two-dimensional manifold instead. The reason certain dimensions of the model manifold are thin is because they represent sets of predictions that differ very little despite large changes in parameter values. These directions in data space correspond to sloppy directions in parameter space. Using the FIM to find sloppy directions, parameter values can be taken to infinity or zero to find the manifold's edge. The path to the manifold boundary (in data space) which results from taking a parameter to infinity or zero is known as a *geodesic*.

Reducing the dimensionality of the model manifold reduces the number of parameters in the model. This can be accomplished by finding the combination of parameters that defines the manifold's edge, and then by eliminating that dimension and re-defining the model with the new combination forming the new edge. Thus, reducing the dimensionality of the model manifold corresponds to a reduction in dimensionality in parameter space.

To summarize, the theoretical framework for model reduction by manifold boundaries involves the following steps:

1. Define the model and construct a data space to house the model manifold.

2. Compute the Jacobian matrix, then find the FIM and its eigenvalue decomposition to find sloppy directions in parameter space. In data space, follow the sloppy directions along a geodesic to find the manifold boundary.

3. Identify the combination of parameters that trace out the manifold boundary.

4. Reduce dimensionality of model manifold (reducing parameter number of the model) by implementing a new mathematical model reflecting the combination of parameters at the manifold boundary. Replace old parameters with their new combinations.

5. Evaluate whether the model satisfies the accuracy requirements.

6. Repeat steps 2-5.

Generally each iteration of model reduction decreases the number of parameters in the model by one.

2.2.2 Defining the Model and a Data Space

In order to create a data space, we need to define observation conditions. Designing this simulation involved selecting an appropriate input current, creating a time series, and choosing an appropriate sampling frequency. We then evaluated the model with physiologically relevant parameters and used this result as base-truth against which we would compare our reduced models.

We used the same parameter values that Hodgkin and Huxley used to fit their model to experimental data on squid axons. We chose a Gaussian pulse as the input current to capture key phenomenology. We observed that a Gaussian input beginning at zero, increasing until passing the spiking threshold, and then decreasing again explores both the spiking and non-spiking states of the Hodgkin-Huxley model. The time series we chose covered 200 milliseconds, giving enough time for about twenty spikes during the entire experiment if the input had been above threshold for the entire time series. We sampled at 1001 time points in order to get at least fifty time points per spike in order to accurately capture spike shape. Because each time point contained a value for each of the 4 dynamical variables, V, n, m, and h, the 1001 time points contained a total of 4004 data points.

The computational software developed by Dr. Transtrum for exploring model manifolds in data space requires a file describing observational conditions and a second file defining the mathematical structure of the model. *Expt.jl* (Appendix A) contains the observation information and the file *hh25.jl* defines and solves the original Hodgkin-Huxley model for these conditions. The *Expt.jl* file also includes a weights functionality in which certain residuals were scaled with a multiplier to emphasize their importance in the model. The weights were chosen so that the variation in each residual would reflect the percent error in each variable equally. The three dynamical variables n, m, and h each range between 0 and 1, so they were given the same weight value of 10. We assigned a weight of 0.1 to V since it ranges from -30 to +70, a range 100 times greater than the range for the other three variables.

One additional functionality to discuss here is the ability to actually specify the allowed values for a parameter using exponential transforms. For instance, in the Hodgkin-Huxley model there are a few parameters (like sodium and potassium channel conductances) that must necessarily be limited to positive values only. In our model definition files, we create a parameter space which contains log transforms of the original parameters rather than the parameters themselves. This has a couple of significant (and very useful) consequences. First, no point in parameter space can give a negative value for a conductance x if the parameter is log(x); a negative value for log(x) would simply give a very small x. This lets us impose limitations on parameters that are physically allowed to be positive numbers only. Similarly, parameters that can take on either positive or negative values are transformed with the inverse sinh function rather than the log function.

The second consequence of using a model with transformed parameters is that logarithmic transformations shrink distances in parameter space, making computation more feasible. This cuts down on run time and improves the numerical stability of the simulation.

2.2.3 Following Geodesics to the Manifold Boundary

Methods for solving geodesics are described elsewhere [1]. We implemented these methods in the script *Geodesic.jl*. The output of this script allowed us to infer the values that the parameters approach at the manifold boundary. Another script, *PlotGeodesics.jl*, takes data from *Geodsic.jl* and plots the derivatives of the parameter values along the path towards the manifold boundary. These scripts are included in Appendix C. We observed that at the manifold boundary, the parameter space geodesic velocity is primarily aligned with one or a few parameters, indicating that they are approaching infinity at the boundary.

2.2.4 Identifying Manifold Boundaries

Parameters that approach infinitely large or small values together at the manifold boundary are usually found in practically unidentifiable combinations. After reading the output from *Geodesic.jl* via *PlotGeodesics.jl*, the next step is to determine how the limits at the manifold boundary translate to a combination of parameters. This involves applying the limits the parameters approach in the equations from the model, however finding a finite function using these limits isn't always straightforward. Sometimes it takes extensive algebraic manipulation of the equations in question and physical intuition to identify the new function.

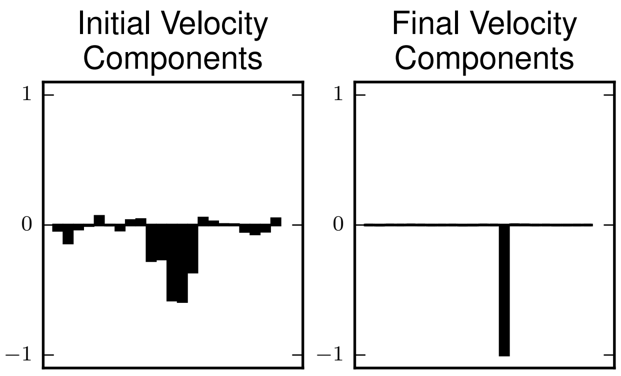

Figure 2.3 Initial and final geodesic velocities for 22 parameter model: This first geodesic finds that log $[K_{\alpha_m}]$ (the transformed analog of K_{α_m}) $\to -\infty$ at this particular boundary of the model manifold. Thus $K_{\alpha_m} \to 0$ in the real model.

For example, after running *Geodesic.jl* using *hh22.jl*, we found that $\log[K_{\alpha_m}]$ (the transformed analog of K_{α_m}) approaches negative infinity at that particular boundary of the model manifold (Figure 2.3). Thus the actual parameter K_{α_m} approaches an infinitesimally small value there. We therefore write $K_{\alpha_m} \to -1/\varepsilon$ and consider the limit $\varepsilon \to 0$.

To understand what this means, we started by rearranging the equation for A_m from the HH model until we arrived at the following expression:

$$A_m = \alpha_m(V - V_{\alpha_m}) \, \frac{1}{1 - e^{\frac{-(V-V_{\alpha_m})}{K_{\alpha_m}}}}. \tag{2.23}$$

Implementing the limit from our geodesic results in the fractional term on the right hand side of the equation being simplified to two possible expressions depending on the value of V:

$$\lim_{\varepsilon \to 0} \frac{1}{1 - e^{-1/\varepsilon}} = 1, \;\; V > V_{\alpha_m} \tag{2.24}$$

or

$$\lim_{\varepsilon \to 0} \frac{1}{1 - e^{1/\varepsilon}} = 0, \;\; V < V_{\alpha_m}. \tag{2.25}$$

These two expressions taken together constitute a shifted Heaviside step function. Thus the equation for A_m could be simplified to

$$A_m \approx \alpha_m(V - V_{\alpha_m}) \mathscr{H}(V - V_{\alpha_m}) \tag{2.26}$$

with practically no effect on model behavior.

This is only the first of eleven reductive approximations we were able to make using information from the geometric structure of the model manifold. Appendix B presents in great detail the steps for identifying the rest of the parameter combinations we identified from the other geodesic paths we studied.

2.2.5 Re-Defining the Model and Optimization

After re-working the structure of the model, the parameter space must be re-defined to reflect the reduced dimensionality of the model manifold. We created simple parameter transform files to import a list of parameters from before the reduction from a text file, and create a new text file with the new list of parameters, omitting the ones that were removed.

Next, we create a new model definition file defining the mathematical structure of the reduced model. We followed a nomenclature pattern in which each model definition file was named according to the number of parameters, i.e., *hh21.jl*, *hh20.jl*, *hh19.jl*, etc. These files are given in Appendix D.

After transforming the parameter vector and creating the new model file, the final step in the reduction process was to find parameter values for the new model. The file *Fit.jl* finds the set of parameter values for the new model that best fits the original model's predictions. It also computes the total least-squares cost of the approximation. A text file is produced as output with a comma-separated list of the parameter values that constitute the optimum fit.

We created a file called *PlotFit.jl* to visualize the predictions of the finalized reduced models compared to the original model predictions.

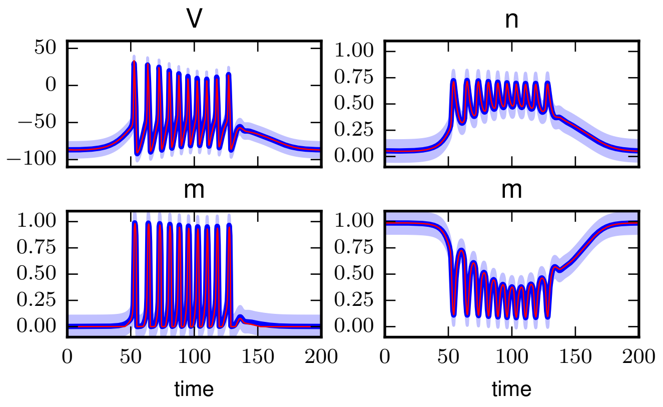

Figure 2.4 21 Parameter Model Fit

Figure (Figure 2.4) shows the fit from our 21-parameter HH model. The remaining fits of the other reduced models are contained in Appendix B.

Chapter 3

Results

3.1 A Brief Overview of the Results

Using the iterative MBAM process of model reduction, we reduced the parameter number of the Hodgkin-Huxley model until we obtained a reduced model that no longer satisfied the 95 percent accuracy requirement. We identified a total of 3 structurally unidentifiable combinations and 8 practically identifiable combinations, for a total reduction of 11 parameters. The transition from 14 to 13 parameters resulted in significant loss of predictive accuracy, so we concluded that the 14-parameter model was the most accurate and minimal approximation. Our reduction process also decreased the dynamical order of the system by one, with our reduced model only involving 3 coupled differential equations rather than 4.

After implementing all of the reductive approximations shown in appendix C, the Hodgkin-Huxley model has the form:

$$C_m \frac{dV}{dt} = (I_{app} - G_{{}_K} n^4 (V - E_{{}_K}) - GE_{Na}\, m_\infty^3 h - GE_L) \tag{3.1}$$

where

$$\frac{dn}{dt} = \frac{n - n_\infty}{\tau_n} \tag{3.2}$$
$$\frac{dh}{dt} = \frac{h - h_\infty}{\tau_h} \tag{3.3}$$

$$n_\infty = \frac{A_n}{A_n + \tilde{\beta}_n} \tag{3.4}$$
$$m_\infty = \frac{1}{1 + \beta_m \alpha V_m e^{\frac{-V}{K_{\beta_m}}}} \tag{3.5}$$
$$h_\infty = \frac{\tilde{\alpha}_h}{\tilde{\alpha}_h + B_h} \tag{3.6}$$

$$\tau_n = \frac{\tilde{\alpha}_h}{\tilde{\alpha}_h + B_h} \tag{3.7}$$
$$\tau_h = \frac{\tilde{\alpha}_h}{\tilde{\alpha}_h + B_h} \tag{3.8}$$

$$A_n = \alpha_n (V - V_{\alpha_n}) \mathscr{H}(V - V_{\alpha_n}) \tag{3.9}$$

$$B_h = \frac{\beta_h}{1 + e^{\frac{-(V - V_{\beta_h})}{K_{\beta_h}}}}. \tag{3.10}$$

In this section we discuss some key insights gained during the reduction process.The mathematical details of each round of model reduction are included in Appendix C. The final section of this chapter summarizes our work to evaluate the behavior of the reduced model.

3.2 Insights Gained from the Model Reduction

As discussed previously, one key aspect of MBAM model reduction is that it reveals what physical parameters truly govern the behavior of a physical model. The 11 reductive approximations either removed the parameters with little effect on the model, or identified the combinations of parameters that controlled model behavior. In this section, we discuss notable examples of insights gained about the HH model from just a few of the parameter combinations we discovered. Each iteration of model reduction is outlined in detail in Appendix A.

For example, the first reduction step (Appendix B.1) approximates a sigmoid curve with a step function. The resulting step function is centered around a voltage V_{α_m}, indicating that this voltage must represent the threshold for sodium channel activation gates.

Another step (Appendix B.4) in the reduction process found the parameters G_{Na} and E_{Na} to be practically unidentifiable. Using our computational methods and some algebra, we found that their identifiable combination is a product of the two. A conductance times a voltage yields a current, so the identifiable parameter GE_{Na} is the effective sodium current. This means that the voltage-dependence of the sodium current is insignificant for model behavior because it remains essentially constant, even during the spiking behavior.

One very significant reduction resulted in removing the sodium channel conductance time constant, approximating it as zero (Appendix B.8). This resulted in reducing the order of the system of coupled differential equations from four to three, with an algebraic approximation determining the sodium channel dynamics rather than a differential equation. See Appendix C for details.

3.3 Evaluating the Behavior of the Reduced Model

To evaluate the accuracy and the limitations of our most simplified version of the Hodgkin-Huxley model, we evaluated its behavior under different experimental paradigms. We first explored its behavior for various types of input currents. We then varied parameter values to simulate different experimental conditions. Lastly we explored the differences between the reduced model and the original model in synchronous neuron networks connected by gap junctions.

While the final, 14-parameter model may have certain shortcomings, it is important to note that it is not the only reduced neuron model we have produced. We have produced 8 additional models that capture Hodgkin-Huxley sodium and potassium dynamics with increasing degrees of accuracy and complexity.

3.3.1 Response to Various Input Currents

We tested the reduced model's behavior with a sine wave input current, a sharp step input current, and constant input current. We found that the model behaves very well with smooth, low frequency currents (Figures 3.1) and constant currents (Figure 3.2) but over-reacts to high frequency currents (Figure 3.3).

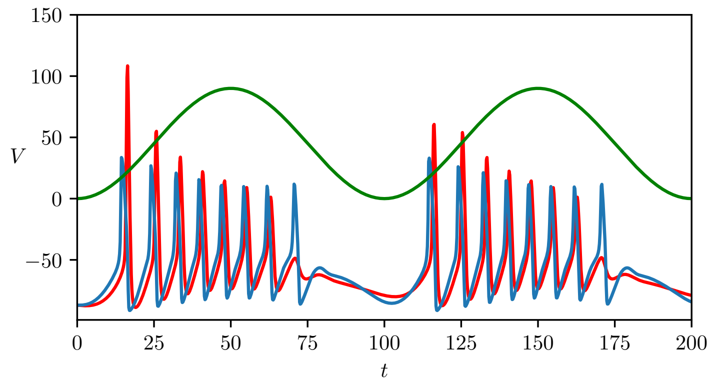

Figure 3.1 Reduced Model Response to Sine Wave Input Current. Green: Input current (mA). Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

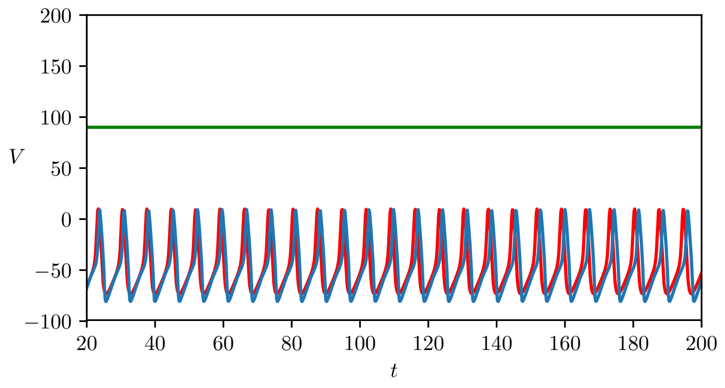

Figure 3.2 Reduced Model Response to Constant Input Current. Green: Input current (mA). Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

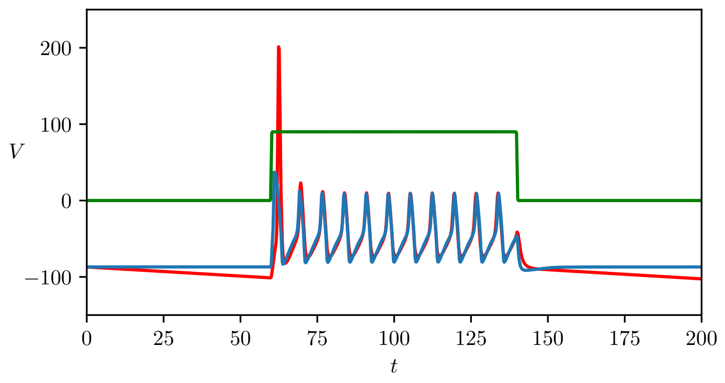

Figure 3.3 Reduced Model Response to Step Input Current. Green: Input current (mA). Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

In the Figure 3.3, you might notice that the when the input current is zero, the voltage drops rather than remaining constant. This is a consequence of our choice of experimental conditions. Because our initial conditions didn't include a time period at resting membrane potential, that aspect of the model- stability at zero input current- was lost. After some investigation, we pinpointed this effect on one specific parameter reduction approximating the expression $G_L(V - E_L)$ as simply the product $G_L \times E_L$, reducing a voltage-dependent leak current centered around a resting leak potential to a constant leak current. This approximation is obviously problematic, making the voltage trail off towards negative infinity when the neuron isn't receiving any input current. For some applications, this specific approximation could be undone to keep the neuron's resting membrane potential stable.

3.3.2 Varying Experimental Conditions by Manipulating Parameter Values

Models are useful because the same mathematical structure can be adapted to describe different experimental conditions by simply changing the parameter values. For instance, the parameter E_{Na} can be manipulated to represent a change in sodium concentration in the extracellular fluid. We found that some parameters could be varied as much as 50 percent below or above their original values without creating too significant of a disparity between the reduced and original models. These parameters were mainly the ones related to environmental conditions, such as reversal potentials such as E_K (Figure 3.4) or membrane conductances such as G_{Na} (Figure 3.5).

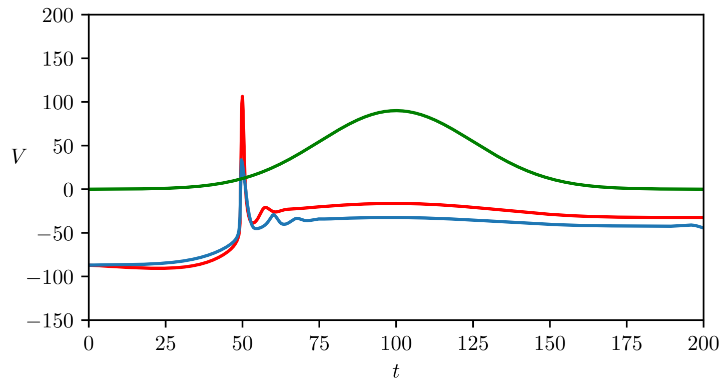

Figure 3.4 Full and reduced HH models with the parameter E_K decreased to 50 percent of the original value. Green: Input current (mA). Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

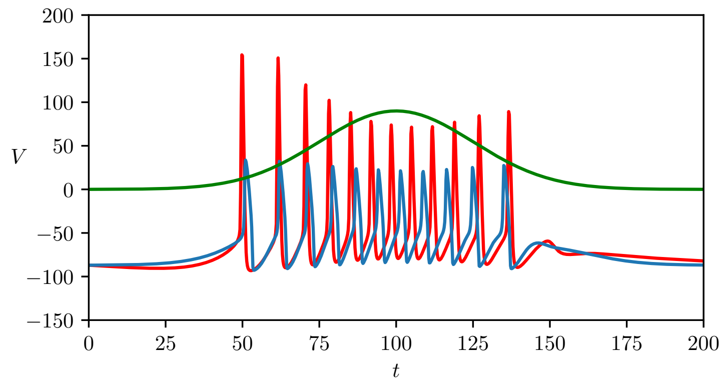

Figure 3.5 Full and reduced HH models with the parameter G_{Na} (or the parameter combination in which it is found) increased to 150 percent of the original value. Green: Input current (mA). Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

A few parameters were very sensitive to change and abolished accurate behavior (in terms of computational cost) when they were changed. These parameters seemed to be related to spiking threshold, like V_{α_n} (Figure 3.6).

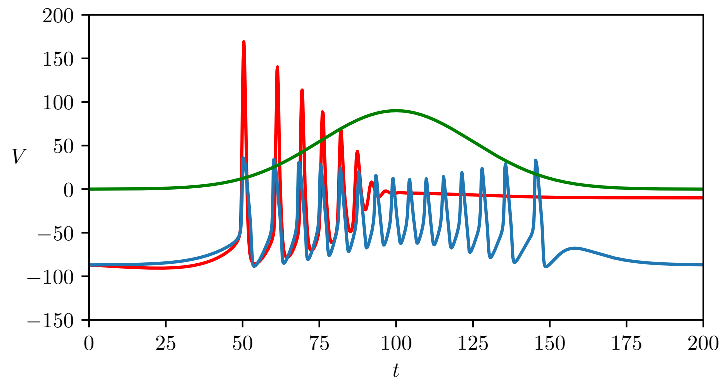

Figure 3.6 Full and reduced HH models with the parameter V_{α_n} decreased to 50 percent of the original value. Green: Input current (mA). Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

3.3.3 Behavior in Neuron Networks

Because neuroscientists are often interested in implementing neuron models in networks, we next aimed to evaluate the behavior of our 14-parameter model in simple synchronous networks. Networks representing neurons connected by gap junctions were our network of choice, as they required no time delay or modeling of neurotransmitter release. Gap junctions are the common mode of connectivity in the heart and the muscles, where synchronous behavior is required for proper physiological function.

To create a connectivity map, we constructed a matrix ε where each column i denotes all of the different inputs to neuron i and each row j represents all of the outputs of neuron j. Thus each entry ε_{ij} represent the coupling between neurons i and j.

We constructed the following connectivity matrix to model a simple circular loop of seven neurons connected by gap junctions:

$$\varepsilon = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.11}$$

In this simulation, we also included a constant applied curent, but only for neuron 1. The rest of the neurons' input would come solely from other neurons in the network (Figure 3.7).

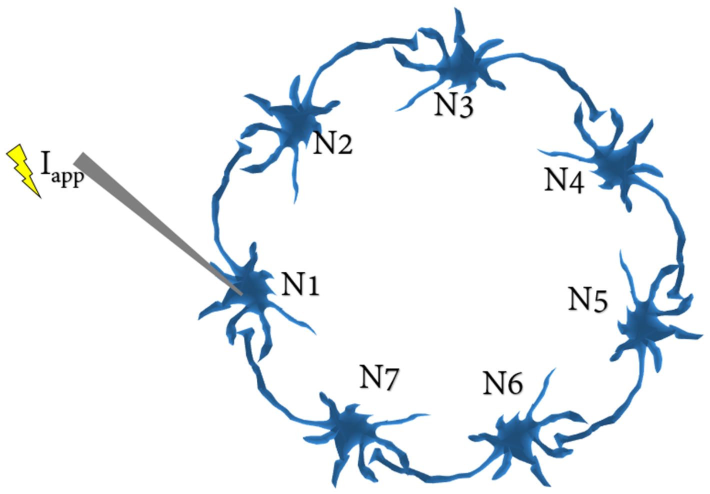

Figure 3.7 Physiological representation of the simulated neuron network. The seven neurons are connected in a loop, and only neuron 1 (N1) receives an external input current.

We simulated both the 25 and the 14 parameter models. Results are shown in Figure 3.8. The code files used to perform this simulation are included in appendix E.

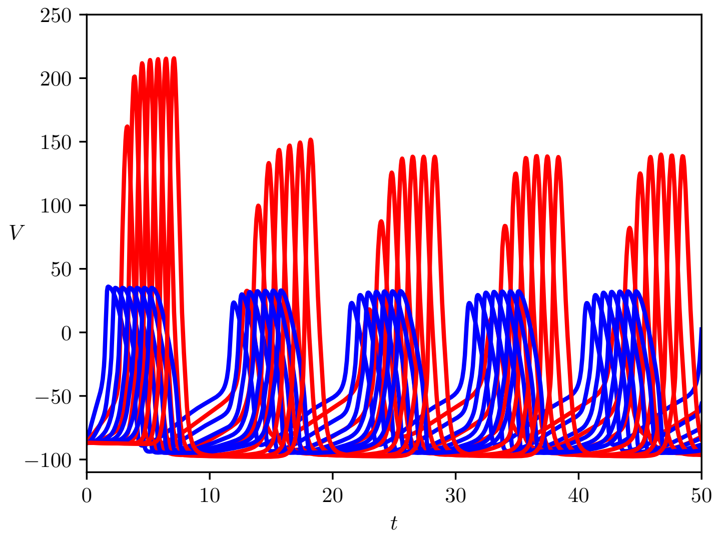

Figure 3.8 Full and reduced HH models implemented in a circular loop network comprised of seven neurons. Blue: 25-parameter model (mV). Red: 14-parameter model (mV).

We observed that in neuron networks, input current constantly changes and does so rather sharply. In our prior evaluations varying input current, we noticed that over-reaction to high frequency input was an issue with our reduced model. Our network simulation shows similar results-spike amplitude is almost double compared to the original model. However, spike shape and timing are still very similar to the 25 parameter model. With some sort of scaling factor to reduce the spike amplitude of HH14 neurons, they might prove useful in network simulations.

Chapter 4

Conclusion

4.1 Significance of Results

Our work using MBAM on the HH model had three main outcomes. First, we learned that MBAM can indeed be performed on models like HH, in principle. Second, we found structural unidentifability in the HH model. Finally, we have elucidated a clear path for future work.

After using MBAM on the HH model, we have obtained a series of new spiking neuron models, each with varying degrees of both accuracy and complexity- ranging from 14 to 25 parameters and either 3 or 4 dynamical variables. Our work has also revealed which parameters might be of lesser importance to model behavior, and how to implement a new mathematical structure that leaves model predictions practically unchanged.

As explained previously, structurally unidentifiable parameters make a mathematical model superfluously complex. Our observation that there are three instances of structural unidentifiability in the Hodgkin-Huxley model's parameters is quite surprising, considering the history and popularity of the HH model.

Because our model reduction method defines new parameters as combinations of original parameters, the parameters in our model have physical significance. If we wish to adapt our reduce model to a different set of experimental conditions, this can be done without obtaining new experimental data first and fitting our parameters to that data. Stated differently, our models' parameters represent real physical quantities and so our model is more useful because it can make predictions for a different system without empirically fitting data to determine parameter values.

4.2 Implications for Neuroscientists

Our work suggests that many detailed neurological models likely contain more information than necessary for the modeling tasks in which they are employed. Top-down reduction using a technique like MBAM could cut down on computational cost while still preserving the predictive power of a model and maintaining the physical significance of its parameters.

One very interesting hypothesis that emerged during our work on this model was the possibility that parameter reduction of detailed HH-type models could actually lead to the derivation of the simpler neuron models meant to capture their phenomenology. For instance, using the MBAM model reduction method for a constant input current, perhaps the 25-parameter HH model could be reduced to the same (or a very similar) form as the Fitzhugh-Nagumo model. Likewise, a detailed HH-type model of bursting involving sodium, potassium, and calcium currents might be reducible to something similar to the Hindmarsh-Rose model. Such a derivation would be of great consequence, showing which combinations of physical quantities correspond to the different parameters in these models. This might solve the problem of dimensionless parameters and the constant need for optimization in simulations with simplified neuron models.

4.3 Future Directions

The Hodgkin-Huxley model is the most basic of all detailed neuron models, with only two types of ion currents, sodium and potassium. Hundreds of different types of detailed neuron models exist today, with a great variety of behaviors, however their only difference from the HH model is the incorporation of additional ionic currents. Our methods of model reduction might easily be applied to any one of these models, following the steps outlined in this thesis.

In this study, we chose low frequency observational conditions to investigate the Hopf bifurcation in the HH model, or the transition to the spiking state. Only after our reduction did we observe that high frequency inputs actually play a prominent role in this transition. This was not a result we expected, and is novel to the neuroscience community. Future experiments using MBAM on HH with a step input current or a spiking input would yield key insights that might explain the effect of high-frequency inputs on the behavior of neuron models.

While single neurons are the building blocks of the brain, the computations performed by the human brain may be simpler than the massive 100-billion neuron network which comprises it. Model reduction on large neuronal networks might someday help provide valuable insights into the true mechanisms governing processes like cognition, decision making, or sensory processing. In an age of large datasets and cumbersome models, future work applying the principles of model reduction and information theory to neuroscience will be very important. Such work will advance our ability to simulate human brain activity, and may even help us understand the very processes which govern the human mind.

Bibliography

[1] M. K. Transtrum and P. Qiu, “Model reduction by manifold boundaries,” Phys. Rev. Lett. **113,** 098701 (2014).

[2] M. Sparkes, “Supercomputer models one second of human brain activity,” The Telegraph http://www.telegraph.co.uk/technology/10567942/Supercomputer-models-one-second-of-human-brain-activity.html (2014).

[3] A. L. HODGKIN and A. F. HUXLEY, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” J. Physiol. (Lond.) **117,** 500–544 (1952).

[4] J. Bardeen, L. N. Cooper, and J. R. Schrieffer, “Microscopic Theory of Superconductivity,” Phys. Rev. **106,** 162–164 (1957).

[5] R. Fitzhugh, “Impulses and Physiological States in Theoretical Models of Nerve Membrane,” Biophys. J. **1,** 445–466 (1961).

[6] J. Nagumo, S. Arimoto, and S. Yoshizawa, “An active pulse transmission line simulating nerve axon,” Proceeding IRE **50,** 2061–2070 (1962).

[7] J. L. Hindmarsh and R. M. Rose, “A model of neuronal bursting using three coupled first order differential equations,” Proc. R. Soc. Lond., B, Biol. Sci. **221,** 87–102 (1984).

[8] M. K. Transtrum, B. B. Machta, and J. P. Sethna, “Geometry of nonlinear least squares with applications to sloppy models and optimization,” Phys Rev E Stat Nonlin Soft Matter Phys **83,** 036701 (2011).

Appendix A

Parameter Information

A.1 Physical Significance of the Dynamical Variables

The Voltage V denotes the potential difference between the inside of the cell and the outside of the cell, $V_{in} - V_{out}$, which depends on the currents through the membrane and ion concentrations in the intra-and extra-cellular fluid at any given time. The initial condition $V = 0$ represents the membrane voltage at time $t = 0$.

Each Potassium (K+) channel is a protein made up of four subunits. In order for the channel to be open, all four subunits must be activated simultaneously. The value of n is the proportion of subunits that are currently activated. Thus n^4 represents the proportion of activated K+ channels. The initial condition n_0 is the proportion of activated K+ subunits at time $t = 0$.

Activation of the Sodium (Na+) channel depends on four subunits. Each sodium channel has three voltage-dependent subunits that control the opening of the pore. Another region of the protein can fold and dock itself in the mouth of the pore, inactivating the ion channel. Even when the three gating subunits are open, the channel is completely inactivated when blocked by the docked protein. In order for the channel to be open, all three subunits must be activated and the blocking domain must not be active.

The variable m represents the proportion of activated sodium gating subunits. The proportion of sodium channels that are blocked at a given time is given by the variable h. Thus the quantity m^3h gives the number of sodium channel proteins that are in a conductive state. The initial condition m_0 represents the proportion of activated Na+ gating subunits at time $t = 0$. Likewise, the initial condition h_0 represents the proportion of sodium channels that are blocked at time $t = 0$.

A.2 Physical Significance of the Parameters

The applied current I_{app} represents an applied current from an external source. In our simulation for model reduction, we chose a gaussian function to serve as our applied current.

The membrane capacitance C_m treats the cell membrane like a dielectric separating two plates. The two plates are the fluid bathing either side of the membrane. Note that this quantity is the capacitance per area of the cell membrane.

The average conductances of potassium channels, sodium channels, and leak channels, per area of membrane in their maximum conductance state are given by g_K, g_Na, and g_L, respectively. Likewise, the specific reversal or Nernst potential for the specific ions associated with these channels is given by E_K, E_Na, and E_L.

The remaining parameters are fitting parameters, according to Hodgkin and Huxley's 1952 paper. The equations for A_i and B_i, which are defined in section (ref?), depend on 18 parameters, six for each variable n, m and h. A_i and B_i are positive, voltage-dependent rate constants for the rate of activation and deactivation of the ion channel subunits, respectively. The physical mechanisms behind the ion channels' subunit activation are not yet understood. However, the fact that A_i and B_i must be positive requires certain of the parameters to be positive.

A.3 Allowed Values for the Variables and Parameters

Table A.1 NC = No constraints, (+) = Constrained to positive values, 0-1 = Constrained between 0 and 1

n_0	0-1	G_K	(+)	a_n	(+)	am	(+)	α_h	(+)
m_0	0-1	G_{Na}	(+)	V_{α_n}	NC	V_{α_m}	NC	V_{α_h}	NC
h_0	0-1	G_L	(+)	K_{α_n}	(+)	K_{α_m}	(+)	K_{α_h}	(+)
V_0	NC	E_K	NC	β_n	(+)	β_m	(+)	β_h	(+)
I_{app}	NC	E_{Na}	NC	V_{β_n}	NC	V_{β_m}	NC	V_{β_h}	NC
C_m	(+)	E_L	NC	K_{β_n}	(+)	K_{β_m}	(+)	K_{β_h}	(+)

A.4 Parameter Values and Initial Conditions

Table A.2 NC = No constraints, (+) = Constrained to positive values, 0-1 = Constrained between 0 and 1

n_0	0.319	G_K	36.0	α_n	0.01	α_m	0.1	α_h	0.07
m_0	0.0570	G_{Na}	120.0	V_{α_n}	-50	V_{α_m}	-36	V_{α_h}	-60
h_0	0.594	G_L	0.3	K_{α_n}	10	K_{α_m}	10	K_{α_h}	20
V_0	-59.9	E_K	-95.3	β_n	0.125	β_m	4	β_h	1
I_{app}	G(t)	E_{Na}	36.7	V_{β_n}	-60	V_{β_m}	-60	V_{β_h}	-30
C_m	1.0	E_L	-87	K_{β_n}	80	K_{β_m}	18	K_{β_h}	10

Appendix B

Iterations of MBAM Model Reduction

Using the MBAM reduction process, we found the practically identifiable combinations of the original HH parameters that yield model behavior approximately unchanged.

To aid in the reduction process, we transformed our parameters using exponential functions. This resulted in two advantages. First, distances in parameter space were reduced, increasing computational efficiency of our exploration of the model manifold. Second, it allowed us to impose restrictions on the parameters based on their allowed physical values (Table reftable).

In this section the parameter combinations we identified are listed beginning with the most practical, i.e., the effect of each subsequent parameter reduction on the overall model behavior increases with each iteration. A first graphic is provided to show our results as we follow geodesics to the manifold boundary in the transformed space. The transform must be reversed to find the real limits that the parameters approach at the boundary. The real limits indicate the identifiable combination of the parameters. We apply those limits and adapt our model to the new parameter combination. A second graphic shows the fit of the reduced model to the original 25-parameter model.

B.1 Elimination of K_{α_m}(Ps: 22 $\to$ 21)

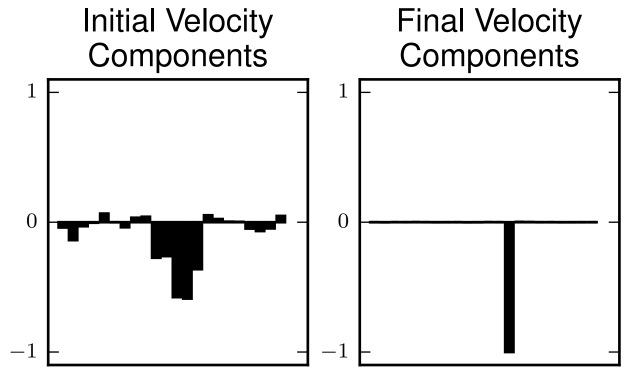

Figure B.1 Initial and final geodesic velocities for 22 parameter model: This first geodesic finds that $\log[K_{\alpha_m}]$ (the transformed analog of K_{α_m}) $\to -\varepsilon^{-1}$ at this particular boundary of the model manifold. Thus $K_{\alpha_m} \to \varepsilon$ in the real model.

We start by rearranging the equation for A_m from the HH model:

$$A_m = \alpha_m(V - V_{\alpha_m})\,\frac{1}{1 - e^{\frac{-(V-V_{\alpha_m})}{K_{\alpha_m}}}}. \tag{B.1}$$

Our geodesic found the limit $K_{\alpha_m} = \varepsilon$, with $\varepsilon \to 0$ (Figure B.1). The fractional term on the right hand side can be simplified to two possible expressions depending on the value of V:

$$\frac{1}{1 - e^{-1/\varepsilon}} = 1,\ \ V > V_{\alpha_m} \tag{B.2}$$

$$\frac{1}{1 - e^{1/\varepsilon}} = 0,\ \ V < V_{\alpha_m} \tag{B.3}$$

These two expressions taken together constitute a shifted Heaviside step function. From our limit we obtain

$$A_m \approx \alpha_m(V - V_{\alpha_m})\,\mathscr{H}(V - V_{\alpha_m}). \tag{B.4}$$

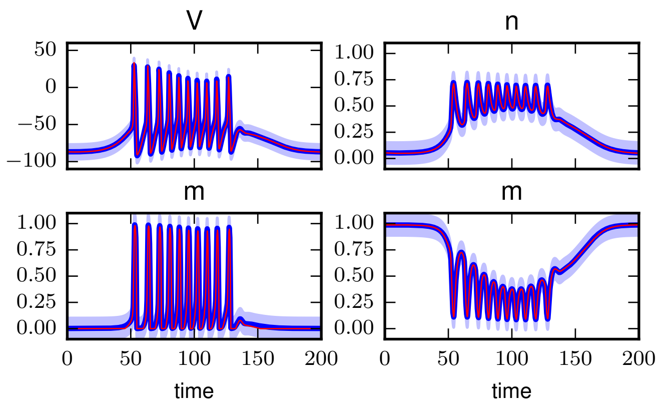

Figure B.2 21 Parameter Model Fit

The fit of the resulting 21 parameter model to the true model is shown in Figure B.2.

B.2 Elimination of K_{β_n} (Ps: 21 $\rightarrow$ 20)

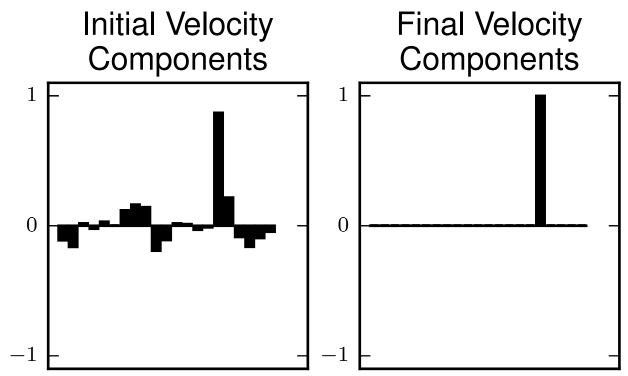

Figure B.3 Initial and final geodesic velocities for 21 parameter model: $\log[K_{\beta_n}] \rightarrow \varepsilon^{-1}$. Thus $K_{\beta_n} \rightarrow \varepsilon^{-1}$.

Beginning with the equation for B_n, we have

$$B_n = \tilde{\beta}_n \, e^{\frac{-V}{K_{\beta_n}}}. \tag{B.5}$$

From our geodesic we see that $K_{\beta_n} \to \varepsilon^{-1}$. Therefore,

$$B_n \approx \tilde{\beta}_n. \tag{B.6}$$

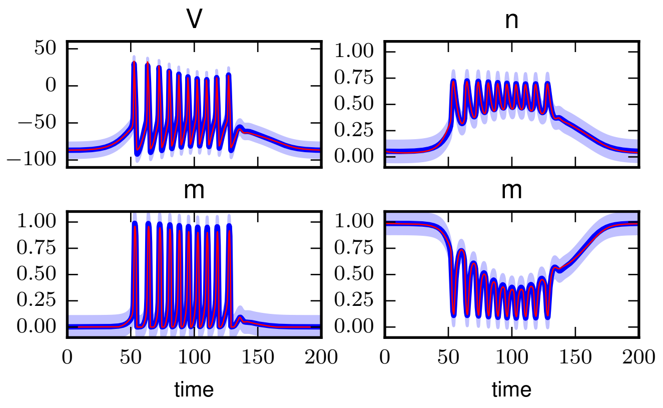

Figure B.4 20 Parameter Model Fit

B.3 Elimination of K_{α_h} (Ps: 20 $\to$ 19)

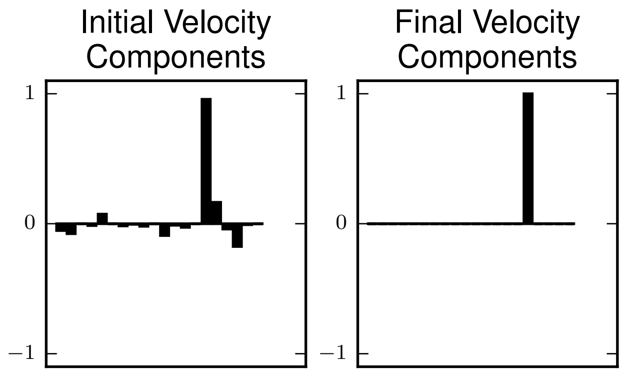

Figure B.5 Initial and final geodesic velocities for 20 parameter model: $\log[K_{\alpha_h}] \to \varepsilon^{-1}$. Thus $K_{\alpha_h} \to \varepsilon^{-1}$.

The equation for A_h has the form

$$A_h = \tilde{\alpha}_h \, e^{\frac{-V}{K\alpha_h}}. \tag{B.7}$$

The geodesic shows that $K_{\alpha_h} \to \varepsilon^{-1}$. Applying the limit reduces the equation to

$$A_h \approx \tilde{\alpha}_h \, e^{-V*\varepsilon} \approx \tilde{\alpha}_h \,. \tag{B.8}$$

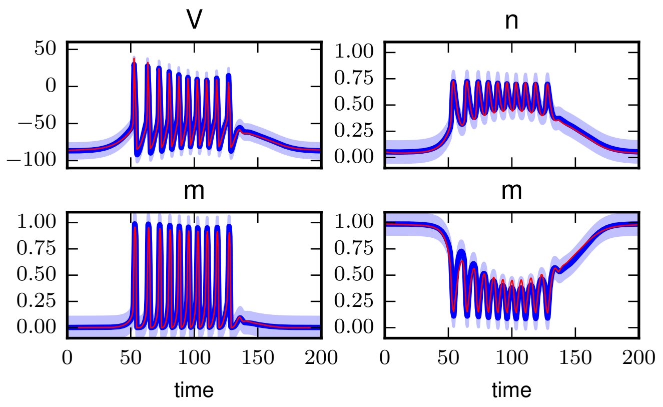

Figure B.6 19 Parameter Model Fit

B.4 Combination of G_{Na} and E_{Na} (Ps: 19 → 18)

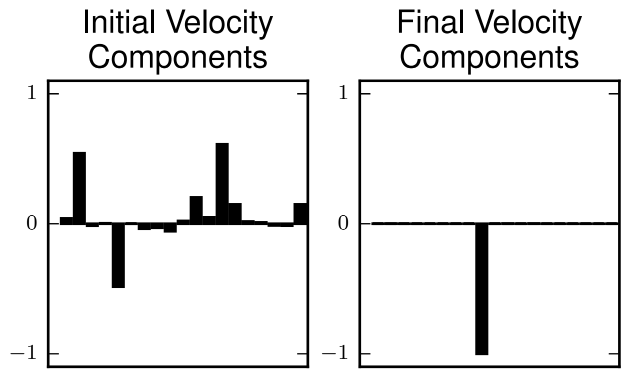

Figure B.7 Initial and final geodesic velocities for 19 parameter model: $G_{Na} \to \varepsilon$ and $E_{Na} \to \varepsilon^{-1}$, thus $G_{Na} \to \varepsilon$ and $E_{Na} \to \varepsilon^{-1}$.

The equation for the sodium current can be rearranged to take the form

$$I_{Na} = m^3 h \, G_{Na} \, V - \, m^3 h \, G_{Na} \, E_{Na}. \tag{B.9}$$

From the geodesic plot we see that $G_{Na} \to \varepsilon$ and $E_{Na} \to \varepsilon^{-1}$, giving

$$I_{Na} = m^3 h\, \varepsilon\, V - \, m^3 h\, \varepsilon\, \varepsilon^{-1}, \tag{B.10}$$

which reduces to

$$I_{Na} \approx -\, m^3 h\; GE_{Na}, \tag{B.11}$$

where GE_{Na} is a new O(1) parameter that replaces G_{Na} and E_{Na}.

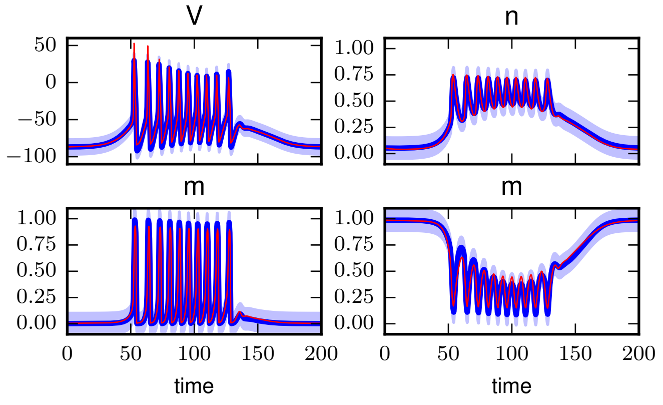

Figure B.8 18 Parameter Model Fit

B.5 Combination of α_m and V_{α_m} (Ps: 18 → 17)

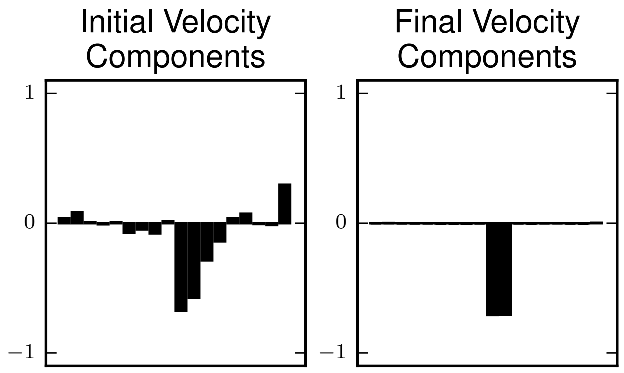

Figure B.9 Initial and final geodesic velocities for 18 parameter model: $\log[\alpha_m] \to -1/\varepsilon$ and $\operatorname{asinh}[V_{\alpha_m}] \to -1/\varepsilon$, so $\alpha_m \to \varepsilon$ and we make an informed guess that $V_{\alpha_m} \to 1/\varepsilon^{-2}$.

We continue reducing the equation for $\tilde{A}_m$ by rearranging (B.32) to take the form

$$\tilde{A}_m \;=\; (\alpha_m V - \alpha_m V_{\alpha_m})\,\mathscr{H}(V - V_{\alpha_m}). \tag{B.12}$$

The geodesic shows that $\alpha_m \to \varepsilon$ and $V_{\alpha_m} \to \varepsilon^{-2}$:

$$\tilde{A}_m \approx (\varepsilon V - \varepsilon\varepsilon^{-2})\mathscr{H}(V - \varepsilon^{-2}) \tag{B.13}$$

$$\tilde{A}_m \approx (-\varepsilon\varepsilon^{-2})\varepsilon \approx -O(1). \tag{B.14}$$

We define a new parameter αV_m to represent the product of α_m and V_{α_m} in this limit. Now we redefine $\tilde{A}_m$ as

$$\tilde{A}_m(V) \;= -\alpha V_m\,. \tag{B.15}$$

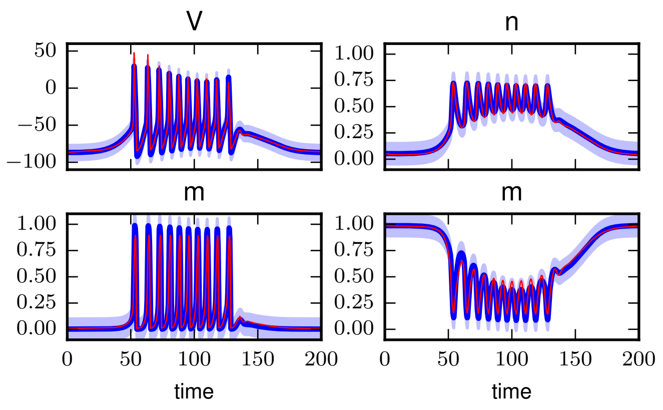

Figure B.10 17 Parameter Model Fit

B.6 Combination of G_L and E_L (Ps: 17 → 16)

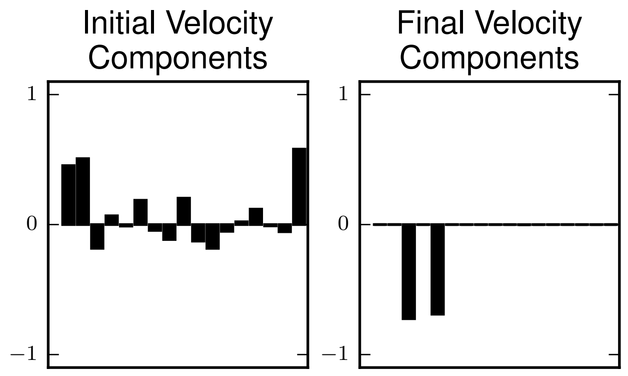

Figure B.11 Initial and final geodesic velocities for 17 parameter model: $\operatorname{asinh}[E_L] \to -1/\varepsilon$ and $\log[G_L] \to -1/\varepsilon$ so $E_L \to \varepsilon^{-1}$ and $G_L \to \varepsilon$.

The equation for I_L can be rearranged to have the form:

$$I_L = G_L\, V - \; G_L\, E_L. \tag{B.16}$$

From the geodesic we see that $E_L \to \varepsilon^{-1}$ and $G_L \to \varepsilon$ at the manifold boundary. Using these limits,

$$I_L \approx \varepsilon V - \varepsilon^{-1}\varepsilon \approx -O(1). \tag{B.17}$$

We name a new parameter GE_L to represent the practically identifiable combination of these two parameters. We obtain the following as our approximation:

$$I_L \approx -\, GE_L\,. \tag{B.18}$$

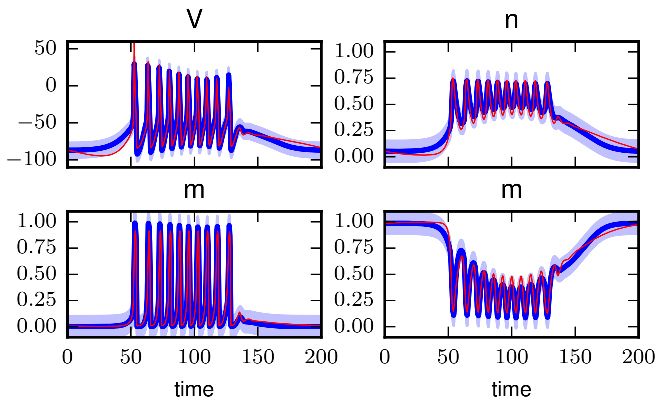

Figure B.12 16 Parameter Model Fit

B.7 Elimination of K_{α_n} (Ps: 16 → 15)

Identical to the elimination of K_{α_m}.

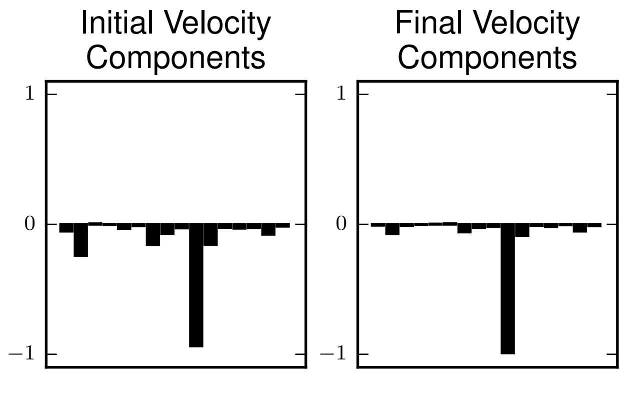

Figure B.13 Initial and final geodesic velocities for 16 parameter model: $\log[K_{\alpha_n}] \to -\varepsilon^{-1}$ so $K_{\alpha_n} \to \varepsilon$.

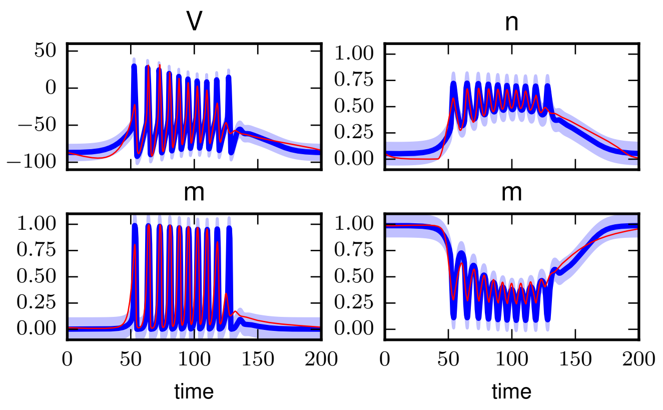

Figure B.14 15 Parameter Model Fit

B.8 Combination of αV_m and $\tilde{\beta}_m$ (Ps: $15 \to 14$)

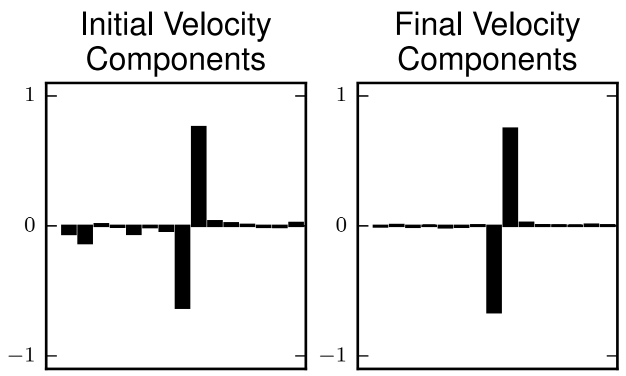

Figure B.15 Initial and final geodesic velocities for 18 parameter model: Since $\operatorname{asinh}[\alpha V_m] \to -1/\varepsilon$ and $\log[\tilde{\beta}_m] \to 1/\varepsilon$, $\alpha V_m \to -\varepsilon^{-1}$ and $\tilde{\beta}_m \to \varepsilon^{-1}$.

Substituting our new parameters into the equation for m_∞ yields

$$m_\infty \approx \frac{-\alpha V_m}{-\alpha V_m + \tilde{\beta}_m e^{\frac{-V}{K_{\beta m}}}}, \tag{B.19}$$

which can be rearranged to the form

$$m_\infty \approx \frac{1}{1 - \frac{\tilde{\beta}_m}{\alpha V_m} e^{\frac{-V}{K_{\beta m}}}}. \tag{B.20}$$

The geodesic shows that $\alpha V_m \rightarrow -\varepsilon^{-1}$ and $\tilde{\beta}_m \rightarrow \varepsilon^{-1}$. Below is the result for m_∞:

$$m_\infty = \frac{1}{1 - \frac{-\varepsilon^{-1}}{\varepsilon^{-1}} e^{\frac{-V}{K_{\beta m}}}} \tag{B.21}$$

$$m_\infty = \frac{1}{1 + O(1) e^{\frac{-V}{K_{\beta m}}}}. \tag{B.22}$$

The finite O(1) term represents the practically identifiable ratio of $\tilde{\beta}_m$ to αV_m which we will call $\beta_m \alpha V_m$. The equation for τ_m now has the form

$$\tau_m \approx \frac{1}{-\alpha V_m + \tilde{\beta}_m e^{\frac{-V}{K_{\beta m}}}}. \tag{B.23}$$

Taking the limits indicated by the geodesic, we make a find with major implications:

$$\tau_m \approx \frac{1}{\varepsilon^{-1} + \varepsilon^{-1} e^{\frac{-V}{K_{\beta m}}}}. \tag{B.24}$$

It follows that

$$\tau_m \rightarrow \varepsilon. \tag{B.25}$$

Rearranging the differential equation for sodium channel activation constant m (Eq 2.6)and postulating that $\tau_m \rightarrow \varepsilon$ we obtain

$$\varepsilon \frac{dm}{dt} = m - m_\infty. \tag{B.26}$$

Using the principles of perturbation theory, this differential equation can be approximated by the following algebraic equation:

$$0 = m - m_\infty. \tag{B.27}$$

Therefore

$$m \approx m_\infty. \tag{B.28}$$

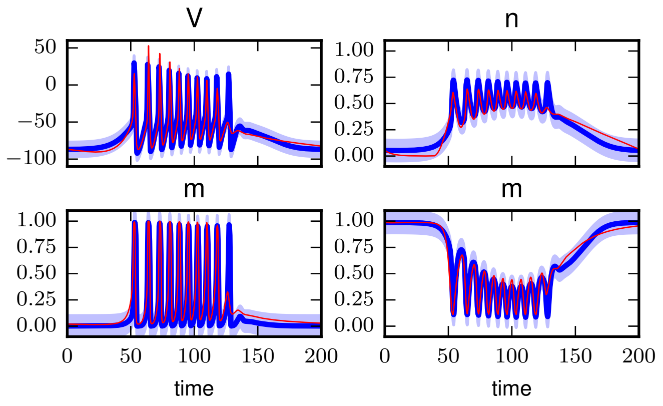

Figure B.16 14 Parameter Model Fit

B.9 Elimination of K_{β_h} and Model Failure (Ps: 14 $\rightarrow$ 13)

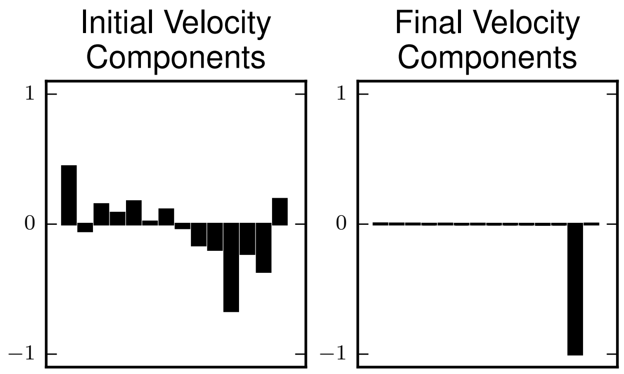

Figure B.17 Initial and final geodesic velocities for 14 parameter model: $\log[K_{\beta_h}] \rightarrow -1/\varepsilon$, so $K_{\beta_h} \rightarrow 0$

.

We start by rearranging the equation for B_h from the HH model:

$$B_h = \beta_h \frac{1}{1+e^{\frac{-(V-V_{\beta_h})}{K_{\beta_h}}}}. \tag{B.29}$$

Our geodesic found the limit $K_{\beta_h} = \varepsilon$, with $\varepsilon \to 0$ (Figure B.17). The fractional term on the right hand side can be simplified to two possible expressions depending on the value of V:

$$\frac{1}{1+e^{-1/\varepsilon}} = 1, \;\; V > V_{\beta_h} \tag{B.30}$$

$$\frac{1}{1+e^{1/\varepsilon}} = 0, \;\; V < V_{\beta_h}. \tag{B.31}$$

These two expressions taken together constitute a shifted Heaviside step function. From our limit we obtain

$$B_h \;\approx\; \beta_h \,\mathscr{H}(V - V_{\beta_h}). \tag{B.32}$$

Interestingly, implementing this new structure into the model resulted in model failure, as defined by our standard of 5 percent error. Interestingly, spiking halts when the derivative on the input current is near zero. Judging from this information and the model fit plot, it is possible that spiking behavior in response to a steady above-threshold input requires the paramater K_{β_h} (Figure B.18).

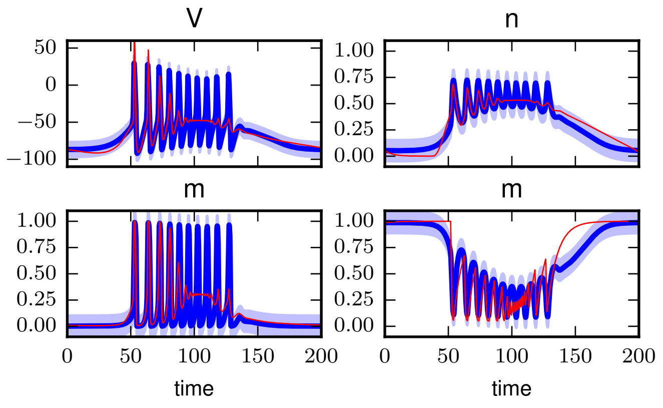

Figure B.18 13 Parameter Model Fit

Because the reduced model's predictions no longer satisfy our accuracy criteria, we end the process of model reduction. We reject the 13 parameter model because of its failure to replicate the predictions of the original model within 95 percent accuracy, and we accept the 14 parameter model as the simplest version of the HH model with predictive accuracy in the context of a simple Gaussian pulse for the input current.

Appendix C

Julia Scripts

C.1 Geodesic.jl

```
import Logging
using NPZ


push!(LOAD_PATH, "./modeldefs")
using hh14b
name = model.description
direction = "f5"
fname = @sprintf("geodesics/geo.%s.%s.npz", name, direction)


logger = Logging.Logger("Geodesic", level = Logging.INFO)
Logging.configure(logger, level=Logging.INFO)
Logging.info(logger, @sprintf("PID = %s", getpid()) )
Logging.info(logger, @sprintf("Geodesic for %s", name) )
Logging.info(logger, @sprintf("Direction: %s", direction) )

flush(STDOUT)

#x = readdlm(@sprintf("xvalues/xtrue.%s.txt", name))
x = readdlm("xvalues/xtrue.hh14.txt") |> vec

import Geometry

# Set Direction

if direction == "f"
    v = Geometry.Geodesics.vi(x, model.j, model.avv)
end
if direction == "r"
    v = -1*Geometry.Geodesics.vi(x, model.j, model.avv)
end
if direction == "f2"
    j = model.j(x)
    v = svd(j)[end][:,end-1]
    a = vec( - (j'*j) \ (j'*model.avv(x,v))) ## acceleration
    if dot(v,a) < 0.0
        v *= -1.0
    end
end
```

```
if direction == "f3"

    j = model.j(x)

    v = svd(j)[end][:,end-2]

    a = vec( - (j'*j) \ (j'*model.avv(x,v))) ## acceleration

    if dot(v,a) < 0.0

        v *= -1.0

    end

end

if direction == "f4"

    j = model.j(x)

    v = svd(j)[end][:,end-3]

    a = vec( - (j'*j) \ (j'*model.avv(x,v))) ## acceleration

    if dot(v,a) < 0.0

        v *= -1.0

    end

end

if direction == "f5"

    j = model.j(x)

    v = svd(j)[end][:,end-4]

    a = vec( - (j'*j) \ (j'*model.avv(x,v))) ## acceleration

    if dot(v,a) < 0.0

        v *= -1.0

    end

end
```

```
geo = Geometry.Geodesics.GeodesicODE(x, v, model.j, model.avv, rtol = 1e-2, atol = 1e-2)

function callback(geo)
    Logging.info(logger, @sprintf("Iteration: %i, t = %e, |v| = %e\n", length(geo.ts), g
    flush(STDOUT)
    npzwrite(fname, Dict("taus"=>geo.ts, "xs"=>geo.xs, "vs"=>geo.vs)) # we use tau for d
    return norm(geo.vs[end,:]) < 10000.0 #was 100
end

callback(geo)

Geometry.Geodesics.integrategeodesic(geo, 10000.0, 10000, callback = callback) #was(geo,

npzwrite(fname,Dict("taus"=>geo.ts, "xs"=>geo.xs, "vs"=>geo.vs))
```

C.2 Fit.jl

```
import Logging
using GeodesicLM

push!(LOAD_PATH, "./modeldefs")
using hh14b
name = model.description
```

```
logger = Logging.Logger("Fit", level = Logging.INFO)

Logging.configure(logger, level=Logging.INFO)

Logging.info(logger, @sprintf("PID = %s", getpid()) )

Logging.info(logger, @sprintf("Fit for %s", name) )


imethod = 11

iaccel = 0

ibold = 0

ibroyden = 1


xstart = readdlm(@sprintf("xvalues/xstart.%s.txt", name)) |> vec


xf, info = geodesiclm(model.r, xstart, model.M, model.N,

                      jacobian = model.j, avv = model.avv,

                      imethod = imethod, iaccel = iaccel, ibold = ibold, ibroyden = ibro

                      print_level = 5, initialfactor = 0.1,maxiter=95)


writedlm(@sprintf("xvalues/xtrue.%s.txt", name), xf)
```

C.3 PlotGeodesics.jl

```
module PlotGeodesic


using NPZ
using PyPlot


push!(LOAD_PATH, "./modeldefs")
using hh19
name = model.description
direction = "f"
fname = @sprintf("geodesics/geo.%s.%s.npz", name, direction)


geo = npzread(fname)
xs = geo["xs"]
vs = geo["vs"]
taus = geo["taus"]
vnorms = zeros(length(taus))
vsnormed = copy(vs)
for i = 1:length(taus)
    vnorms[i] = norm(vs[i,:])
    vsnormed[i,:] /= norm(vsnormed[i,:])
end
ss = zeros(length(taus)) # parameter space arclength
for i = 2:length(taus)
    ss[i] = ss[i-1] + norm(vs[i,:])*(taus[i] - taus[i-1])
end


if true
    figure()
    plot(taus, xs)
    title("Parameter values vs tau")
end

if true
    figure()
    plot(taus, vs)
    title("Velocity Components vs tau")
end

if false
    figure()
    plot(ss, vsnormed)
    title("Normalized Velocity Components vs s")
end



if false
    figure()
    semilogy(taus, vnorms)
    title("|v| vs taus")
end

if true
    figure()
    subplot(1,2,1)
    plot(range(1,model.N), vs[1,:]')
    title("Initial velocity")
    subplot(1,2,2)
    plot(range(1,model.N), vs[end,:]')
    title("Final velocity")
    suptitle("Velocity Componentaus")
end

if true
    figure()
    bar(range(1,model.N), vs[end,:]'./norm(vs[end,:]), color=(0,0,0))
    title("Final Velocity Components")
    ylim(-1,1)
end
show()

end #module
```

C.4 PlotFit.jl

```
module PlotFit
```

```
using PyPlot


push!(LOAD_PATH, "./modeldefs")

figure()

using hh13

using Expt

name = model.description


x = vec( readdlm( @sprintf("xvalues/xtrue.%s.txt", name)))

y = reshape(baremodel.r(x), (length(ts), 4))

Ytrue = reshape( vec( readdlm("data/Ytrue.txt") ), (length(ts), 4) )

sigma = reshape( 1.0./weights, (length(ts), 4))


titles = ["Voltage", "n", "m", "h"]


#for i = 1:4

#  subplot(2,2,i)

#  fill_between(ts, Ytrue[:,i] + sigma[:,i], Ytrue[:,i] - sigma[:,i], color = (0,0,1,0.2

#  plot(ts, Ytrue[:,i], "b-", linewidth = 4)

#  plot(ts, y[:,i], "r-", linewidth = 1)

#  title(titles[i], fontsize = 20)

#end



#fill_between(ts, Ytrue[:,1] + sigma[:,1], Ytrue[:,1] - sigma[:,1], color = (0,0,1,0.25)
I = vec(90.*exp(-(ts-100).^2/1250))

#I = vec(90.*ones(length(ts)))



plot(ts, Ytrue[:,1], "b-", linewidth = 4)
plot(ts, y[:,1], "r-", linewidth = 2)

plot(ts, I, "g-", linewidth = 2)
#title(titles[1])



suptitle(@sprintf("%i Parameters", model.N), fontsize = 32)

end #module
```

C.5 Tranform.jl

```
push!(LOAD_PATH, "./parametertransforms")
using hh14_hh13
using NPZ
```

```
oldname = "xtrue.hh14.txt"


newname = "xstart.hh13.txt"

#fname = @sprintf("geodesics/geo.%s.npz", geoname)

#geo = npzread(fname)

# x = vec(geo["xs"][end,:])


x=vec(readdlm(@sprintf("xvalues/%s",oldname)))


writedlm(@sprintf("xvalues/%s", newname), r(x))
```

Appendix D

Model Definition Files

D.1 hh25.jl

```
module hh25


using Expt

import Modeling



function rhs(t, y, x)

    V, n, m, h = y

    GK = exp(x[1])

    GNa = exp(x[2])

    GL = exp(x[3])

    EK = sinh(x[4])

    ENa = sinh(x[5])
```

```
EL = sinh(x[6])

an = exp(x[7])

Van = sinh(x[8])

Kan = exp(x[9])

bn = exp(x[10])

Vbn = sinh(x[11])

Kbn = exp(x[12])

am = exp(x[13])

Vam = sinh(x[14])

Kam = exp(x[15])

bm = exp(x[16])

Vbm = sinh(x[17])

Kbm = exp(x[18])

ah = exp(x[19])

Vah = sinh(x[20])

Kah = exp(x[21])

bh = exp(x[22])

Vbh = sinh(x[23])

Kbh = exp(x[24])

Cm = exp(x[25])


A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

A_m = am*(V - Vam)/(1 - exp(-(V - Vam)/Kam))

A_h = ah*exp(-(V - Vah)/Kah)
```

```
    B_n = bn*exp(-(V - Vbn)/Kbn)
    B_m = bm*exp(-(V-Vbm)/Kbm)
    B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))

    t_n = 1 /(A_n + B_n)
    t_m = 1 /(A_m + B_m)
    t_h = 1 /(A_h + B_h)

    n_inf = A_n /(A_n + B_n)
    m_inf = A_m /(A_m + B_m)
    h_inf = A_h /(A_h + B_h)

    I = Iapp(t)
    return [(I - GK*(n^4)*(V-EK) - GNa*(m^3)*h*(V-ENa)-GL*(V-EL))/Cm,
    ((n_inf - n)/t_n),
    ((m_inf - m)/t_m),
    ((h_inf - h)/t_h) ]

end

calc = Modeling.ODECalcs.ODECalc(4, 25, rhs, y0)

function rbare(x)
    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))
end
```

```
function r(x)

    return (rbare(x) - ydata).*weights

end


baremodel = Modeling.ModelAD( length(ts)*4, 25, rbare, "hhbare25")

model = Modeling.ModelAD( length(ts)*4, 25, r, "hh25")


export model, baremodel


end # module
```

D.2 hh22.jl

```
module hh22


using Expt

import Modeling



function rhs(t, y, x)

    V, n, m, h = y
    GK = exp(x[1])

    GNa = exp(x[2])

    GL = exp(x[3])

    EK = sinh(x[4])

    ENa = sinh(x[5])

    EL = sinh(x[6])

    an = exp(x[7])

    Van = sinh(x[8])

    Kan = exp(x[9])

    b_tilde_n = exp(x[10])

    Kbn = exp(x[11])

    am = exp(x[12])

    Vam = sinh(x[13])

    Kam = exp(x[14])

    b_tilde_m = exp(x[15])

    Kbm = exp(x[16])

    a_tilde_h = exp(x[17])

    Kah = exp(x[18])

    bh = exp(x[19])

    Vbh = sinh(x[20])

    Kbh = exp(x[21])

    Cm = exp(x[22])


    A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

    A_m = am*(V - Vam)/(1 - exp(-(V - Vam)/Kam))
    A_h = a_tilde_h*exp(-V/Kah)


    B_n = b_tilde_n*exp(-V/Kbn)

    B_m = b_tilde_m*exp(-V/Kbm)

    B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


    t_n = 1 /(A_n + B_n)

    t_m = 1 /(A_m + B_m)

    t_h = 1 /(A_h + B_h)


    n_inf = A_n /(A_n + B_n)

    m_inf = A_m /(A_m + B_m)

    h_inf = A_h /(A_h + B_h)


    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) - GNa*(m^3)*h*(V-ENa)-GL*(V-EL))/Cm,

    ((n_inf - n)/t_n),

    ((m_inf - m)/t_m),

    ((h_inf - h)/t_h) ]


end


calc = Modeling.ODECalcs.ODECalc(4, 22, rhs, y0)


function rbare(x)
```

```
    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))
end


function r(x)
    return (rbare(x) - ydata).*weights
end


baremodel = Modeling.ModelAD( length(ts)*4, 22, rbare, "hhbare22")
model = Modeling.ModelAD( length(ts)*4, 22, r, "hh22")


export model, baremodel


end # module
```

D.3 hh21.jl

```
module hh21


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end
```

```
function rhs(t, y, x)
    V, n, m, h = y


    GK = exp(x[1])
    GNa = exp(x[2])
    GL = exp(x[3])
    EK = sinh(x[4])
    ENa = sinh(x[5])
    EL = sinh(x[6])
    an = exp(x[7])
    Van = sinh(x[8])
    Kan = exp(x[9])
    b_tilde_n = exp(x[10])
    Kbn = exp(x[11])
    am = exp(x[12])
    Vam = sinh(x[13])
    #Kam = 1e-8
    b_tilde_m = exp(x[14])
    Kbm = exp(x[15])
    a_tilde_h = exp(x[16])
    Kah = exp(x[17])
    bh = exp(x[18])
    Vbh = sinh(x[19])
```

```
Kbh = exp(x[20])

Cm = exp(x[21])


A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

#A_m= am*(V - Vam)/(1 - exp(-(V - Vam)/Kam))

A_m = am*(V - Vam)*Step( V - Vam)

A_h = a_tilde_h*exp(-V/Kah)


B_n = b_tilde_n*exp(-V/Kbn)

B_m = b_tilde_m*exp(-V/Kbm)

B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


t_n = 1 /(A_n + B_n)

t_m = 1 /(A_m + B_m)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = A_m /(A_m + B_m)

h_inf = A_h /(A_h + B_h)


I = Iapp(t)

return [(I - GK*(n^4)*(V-EK) - GNa*(m^3)*h*(V-ENa)-GL*(V-EL))/Cm,

((n_inf - n)/t_n),

((m_inf - m)/t_m),

((h_inf - h)/t_h) ]
```

```
end


calc = Modeling.ODECalcs.ODECalc(4, 21, rhs, y0)


function rbare(x)
    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))
end


function r(x)
    return (rbare(x) - ydata).*weights
end


baremodel = Modeling.ModelAD( length(ts)*4, 21, rbare, "hhbare21")
model = Modeling.ModelAD( length(ts)*4, 21, r, "hh21")


export model, baremodel


end # module
```

D.4 hh20.jl

```
module hh20
```

```
using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, m, h = y


    GK = exp(x[1])
    GNa = exp(x[2])
    GL = exp(x[3])
    EK = sinh(x[4])
    ENa = sinh(x[5])
    EL = sinh(x[6])
    an = exp(x[7])
    Van = sinh(x[8])
    Kan = exp(x[9])
    b_tilde_n = exp(x[10])
    am = exp(x[11])
    Vam = sinh(x[12])
    b_tilde_m = exp(x[13])
    Kbm = exp(x[14])
```

```
a_tilde_h = exp(x[15])

Kah = exp(x[16])

bh = exp(x[17])

Vbh = sinh(x[18])

Kbh = exp(x[19])

Cm = exp(x[20])


A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

#A_m= am*(V - Vam)/(1 - exp(-(V - Vam)/Kam))

A_m = am*(V - Vam)*Step( V - Vam)

A_h = a_tilde_h*exp(-V/Kah)


B_n = b_tilde_n

B_m = b_tilde_m*exp(-V/Kbm)

B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


t_n = 1 /(A_n + B_n)

t_m = 1 /(A_m + B_m)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = A_m /(A_m + B_m)

h_inf = A_h /(A_h + B_h)


I = Iapp(t)
```

```
    return [(I - GK*(n^4)*(V-EK) - GNa*(m^3)*h*(V-ENa)-GL*(V-EL))/Cm,
    ((n_inf - n)/t_n),
    ((m_inf - m)/t_m),
    ((h_inf - h)/t_h) ]

end

calc = Modeling.ODECalcs.ODECalc(4, 20, rhs, y0)

function rbare(x)
    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))
end

function r(x)
    return (rbare(x) - ydata).*weights
end

baremodel = Modeling.ModelAD( length(ts)*4, 20, rbare, "hhbare20")
model = Modeling.ModelAD( length(ts)*4, 20, r, "hh20")

export model, baremodel

end # module
```

D.5 hh19.jl

```
module hh19


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, m, h = y

    GK = exp(x[1])
    GNa = exp(x[2])
    GL = exp(x[3])
    EK = sinh(x[4])
    ENa = sinh(x[5])
    EL = sinh(x[6])
    an = exp(x[7])
    Van = sinh(x[8])
    Kan = exp(x[9])
    b_tilde_n = exp(x[10])
    am = exp(x[11])
    Vam = sinh(x[12])

    b_tilde_m = exp(x[13])

    Kbm = exp(x[14])

    a_tilde_h = exp(x[15])

    bh = exp(x[16])

    Vbh = sinh(x[17])

    Kbh = exp(x[18])

    Cm = exp(x[19])


    A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

    #A_m= am*(V - Vam)/(1 - exp(-(V - Vam)/Kam))

    A_m = am*(V - Vam)*Step( V - Vam)

    A_h = a_tilde_h


    B_n = b_tilde_n

    B_m = b_tilde_m*exp(-V/Kbm)

    B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


    t_n = 1 /(A_n + B_n)

    t_m = 1 /(A_m + B_m)

    t_h = 1 /(A_h + B_h)


    n_inf = A_n /(A_n + B_n)

    m_inf = A_m /(A_m + B_m)

    h_inf = A_h /(A_h + B_h)
    I = Iapp(t)
    return [(I - GK*(n^4)*(V-EK) - GNa*(m^3)*h*(V-ENa)-GL*(V-EL))/Cm,
    ((n_inf - n)/t_n),
    ((m_inf - m)/t_m),
    ((h_inf - h)/t_h) ]

end

calc = Modeling.ODECalcs.ODECalc(4, 19, rhs, y0)

function rbare(x)
    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))
end

function r(x)
    return (rbare(x) - ydata).*weights
end

baremodel = Modeling.ModelAD( length(ts)*4, 19, rbare, "hhbare19")
model = Modeling.ModelAD( length(ts)*4, 19, r, "hh19")

export model, baremodel

end # module
```

D.6 hh18.jl

```
module hh18


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, m, h = y


    GK = exp(x[1])
    GENa = sinh(x[2])
    GL = exp(x[3])
    EK = sinh(x[4])
    EL = sinh(x[5])
    an = exp(x[6])
    Van = sinh(x[7])
    Kan = exp(x[8])
```

```
b_tilde_n = exp(x[9])

am = exp(x[10])

Vam = sinh(x[11])

b_tilde_m = exp(x[12])

Kbm = exp(x[13])

a_tilde_h = exp(x[14])

bh = exp(x[15])

Vbh = sinh(x[16])

Kbh = exp(x[17])

Cm = exp(x[18])


A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

A_m = am*(V - Vam)*Step( V - Vam)

A_h = a_tilde_h


B_n = b_tilde_n

B_m = b_tilde_m*exp(-V/Kbm)

B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


t_n = 1 /(A_n + B_n)

t_m = 1 /(A_m + B_m)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = A_m /(A_m + B_m)
```

```
    h_inf = A_h /(A_h + B_h)


    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) + GENa*(m^3)*h-GL*(V-EL))/Cm,

    ((n_inf - n)/t_n),

    ((m_inf - m)/t_m),

    ((h_inf - h)/t_h) ]


end


calc = Modeling.ODECalcs.ODECalc(4, 18, rhs, y0)


function rbare(x)

    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))

end


function r(x)

    return (rbare(x) - ydata).*weights

end


baremodel = Modeling.ModelAD( length(ts)*4, 18, rbare, "hhbare18")

model = Modeling.ModelAD( length(ts)*4, 18, r, "hh18")


export model, baremodel
```

```
end # module
```

D.7 hh17.jl

```
module hh17


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, m, h = y


    GK = exp(x[1])
    GENa = sinh(x[2])
    GL = exp(x[3])
    EK = sinh(x[4])
    EL = sinh(x[5])
    an = exp(x[6])
    Van = sinh(x[7])
    Kan = exp(x[8])
    b_tilde_n = exp(x[9])
    aVm = sinh(x[10])
    b_tilde_m = exp(x[11])
    Kbm = exp(x[12])
    a_tilde_h = exp(x[13])
    bh = exp(x[14])
    Vbh = sinh(x[15])
    Kbh = exp(x[16])
    Cm = exp(x[17])

    A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))
    #A_m = am*(V - Vam)*Step( V - Vam)
    A_m = -aVm
    A_h = a_tilde_h

    B_n = b_tilde_n
    B_m = b_tilde_m*exp(-V/Kbm)
    B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))

    t_n = 1 /(A_n + B_n)
    t_m = 1 /(A_m + B_m)
    t_h = 1 /(A_h + B_h)

    n_inf = A_n /(A_n + B_n)
    m_inf = A_m /(A_m + B_m)

    h_inf = A_h /(A_h + B_h)


    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) + GENa*(m^3)*h-GL*(V-EL))/Cm,

    ((n_inf - n)/t_n),

    ((m_inf - m)/t_m),

    ((h_inf - h)/t_h) ]


end


calc = Modeling.ODECalcs.ODECalc(4, 17, rhs, y0)


function rbare(x)

    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))

end


function r(x)

    return (rbare(x) - ydata).*weights

end


baremodel = Modeling.ModelAD( length(ts)*4, 17, rbare, "hhbare17")

model = Modeling.ModelAD( length(ts)*4, 17, r, "hh17")


export model, baremodel
```

```
end # module
```

D.8 hh16.jl

```
module hh16


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, m, h = y


    GK = exp(x[1])
    GENa = sinh(x[2])
    GEL = sinh(x[3])
    EK = sinh(x[4])
    an = exp(x[5])
    Van = sinh(x[6])
```

```
Kan = exp(x[7])

b_tilde_n = exp(x[8])

aVm = sinh(x[9])

b_tilde_m = exp(x[10])

Kbm = exp(x[11])

a_tilde_h = exp(x[12])

bh = exp(x[13])

Vbh = sinh(x[14])

Kbh = exp(x[15])

Cm = exp(x[16])


A_n = an*(V - Van)/(1 - exp(-(V - Van)/Kan))

A_m = -aVm

A_h = a_tilde_h


B_n = b_tilde_n

B_m = b_tilde_m*exp(-V/Kbm)

B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


t_n = 1 /(A_n + B_n)

t_m = 1 /(A_m + B_m)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = A_m /(A_m + B_m)
```

```
    h_inf = A_h /(A_h + B_h)


    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) + GENa*(m^3)*h + GEL)/Cm,

    ((n_inf - n)/t_n),

    ((m_inf - m)/t_m),

    ((h_inf - h)/t_h) ]


end


calc = Modeling.ODECalcs.ODECalc(4, 16, rhs, y0)


function rbare(x)

    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))

end


function r(x)

    return (rbare(x) - ydata).*weights

end


baremodel = Modeling.ModelAD( length(ts)*4, 16, rbare, "hhbare16")

model = Modeling.ModelAD( length(ts)*4, 16, r, "hh16")


export model, baremodel
```

```
end # module
```

D.9 hh15.jl

```
module hh15


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, m, h = y


    GK = exp(x[1])
    GENa = sinh(x[2])
    GEL = sinh(x[3])
    EK = sinh(x[4])
    an = exp(x[5])
    Van = sinh(x[6])
    b_tilde_n = exp(x[7])
```

```
aVm = sinh(x[8])

b_tilde_m = exp(x[9])

Kbm = exp(x[10])

a_tilde_h = exp(x[11])

bh = exp(x[12])

Vbh = sinh(x[13])

Kbh = exp(x[14])

Cm = exp(x[15])


A_n = an*(V - Van)*Step(V - Van)

A_m = -aVm

A_h = a_tilde_h


B_n = b_tilde_n

B_m = b_tilde_m*exp(-V/Kbm)

B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


t_n = 1 /(A_n + B_n)

t_m = 1 /(A_m + B_m)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = A_m /(A_m + B_m)

h_inf = A_h /(A_h + B_h)
```

```
    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) + GENa*(m^3)*h + GEL)/Cm,

    ((n_inf - n)/t_n),

    ((m_inf - m)/t_m),

    ((h_inf - h)/t_h) ]


end


calc = Modeling.ODECalcs.ODECalc(4, 15, rhs, y0)


function rbare(x)

    return vec(Modeling.ODECalcs.evaluate(calc,x,ts))

end


function r(x)

    return (rbare(x) - ydata).*weights

end


baremodel = Modeling.ModelAD( length(ts)*4, 15, rbare, "hhbare15")

model = Modeling.ModelAD( length(ts)*4, 15, r, "hh15")


export model, baremodel


end # module
```

D.10 hh14.jl

```
module hh14


using Expt

import Modeling


function Step(x)

    return 0.5.*(tanh(10.*x) + 1.0)

end




function rhs(t, y, x)

    V, n, mdummy, h = y


    GK = exp(x[1])

    GENa = sinh(x[2])

    GEL = sinh(x[3])

    EK = sinh(x[4])

    an = exp(x[5])

    Van = sinh(x[6])

    b_tilde_n = exp(x[7])

    bmaVm = sinh(x[8])

    Kbm = exp(x[9])

    a_tilde_h = exp(x[10])

    bh = exp(x[11])
    Vbh = sinh(x[12])

    Kbh = exp(x[13])

    Cm = exp(x[14])


    A_n = an*(V - Van)*Step(V - Van)

    A_h = a_tilde_h


    B_n = b_tilde_n

    B_h = bh /( 1.0 + exp(-(V-Vbh)/Kbh))


    t_n = 1 /(A_n + B_n)

    t_h = 1 /(A_h + B_h)


    n_inf = A_n /(A_n + B_n)

    m_inf = 1/(1 + bmaVm*exp(-V/Kbm))

    h_inf = A_h /(A_h + B_h)


    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) + GENa*(m_inf^3)*h + GEL)/Cm,

    ((n_inf - n)/t_n),

    0,

    ((h_inf - h)/t_h) ]


end
```

```
calc = Modeling.ODECalcs.ODECalc(4, 14, rhs, y0)


function rbare(x)

    bmaVm = sinh(x[8])

    Kbm = exp(x[9])


    y = Modeling.ODECalcs.evaluate(calc,x,ts)

    V = y[:,1]

    y[:,3] = 1./(1. + bmaVm*exp(-V./Kbm))

    return vec(y)

end


function r(x)

    return (rbare(x) - ydata).*weights

end


baremodel = Modeling.ModelAD( length(ts)*4, 14, rbare, "hhbare14")

model = Modeling.ModelAD( length(ts)*4, 14, r, "hh14")


export model, baremodel


end # module
```

D.11 hh13.jl

```
module hh13


using Expt
import Modeling


function Step(x)
    return 0.5.*(tanh(10.*x) + 1.0)
end



function rhs(t, y, x)
    V, n, mdummy, h = y


    GK = exp(x[1])
    GENa = sinh(x[2])
    GEL = sinh(x[3])
    EK = sinh(x[4])
    an = exp(x[5])
    Van = sinh(x[6])
    b_tilde_n = exp(x[7])
    bmaVm = sinh(x[8])
    Kbm = exp(x[9])
    a_tilde_h = exp(x[10])
    bh = exp(x[11])
    Vbh = sinh(x[12])

    Cm = exp(x[13])


    A_n = an*(V - Van)*Step(V - Van)

    A_h = a_tilde_h


    B_n = b_tilde_n

    B_h = bh*Step(V-Vbh)


    t_n = 1 /(A_n + B_n)

    t_h = 1 /(A_h + B_h)


    n_inf = A_n /(A_n + B_n)

    m_inf = 1/(1 + bmaVm*exp(-V/Kbm))

    h_inf = A_h /(A_h + B_h)


    I = Iapp(t)

    return [(I - GK*(n^4)*(V-EK) + GENa*(m_inf^3)*h + GEL)/Cm,

    ((n_inf - n)/t_n),

    0,

    ((h_inf - h)/t_h) ]


end


calc = Modeling.ODECalcs.ODECalc(4, 13, rhs, y0)
```

```
function rbare(x)
    bmaVm = sinh(x[8])
    Kbm = exp(x[9])

    y = Modeling.ODECalcs.evaluate(calc,x,ts)
    V = y[:,1]
    y[:,3] = 1./(1. + bmaVm*exp(-V./Kbm))
    return vec(y)
end

function r(x)
    return (rbare(x) - ydata).*weights
end

baremodel = Modeling.ModelAD( length(ts)*4, 13, rbare, "hhbare13")
model = Modeling.ModelAD( length(ts)*4, 13, r, "hh13")

export model, baremodel

end # module
```

Appendix E

Neuron Network Scripts

E.1 7-loop-network.py

```
import rhs

reload(rhs)

rhs = rhs.rhs

from HH25 import hh25 as hh

import numpy as np

import scipy

from scipy.integrate import odeint

from scipy.integrate import ode

import matplotlib.pyplot as plt

from numpy import matlib


Iapp = 41.25

Nneurons = 7
```

```
ep=np.matlib.zeros((Nneurons,Nneurons))
ep[1,0]=1
ep[2,1]=1
ep[3,2]=1
ep[4,3]=1
ep[5,4]=1
ep[6,5]=1
#ep[0,6]=1
#ep[3,0]=1
#ep[6,1]=1
#ep[5,2]=1
#ep[2,0]=1
#ep[5,3]=1
#ep[6,4]=1
#ep[0,3]=1
epsilon = np.asarray(ep)



def Iext(t):
return [41.25,0,0,0,0,0,0]


V_IC = -86.8091
n_IC = 0.0536886
m_IC = 0.00178839
```

```
h_IC = 0.986998


y0=np.zeros(Nneurons*4)

for i in range(Nneurons):

y0[4*i:4*(i+1)]=[V_IC, n_IC, m_IC, h_IC]


t0=0


t = np.linspace(0,50,1001)

args = (Nneurons, epsilon, hh, Iext)

soln=odeint(rhs,y0,t,args)




for i in range(Nneurons):

        plt.plot(t,soln[:,4*i],'r', linewidth=2.0)


plt.legend(loc='best')

plt.xlabel('t')

plt.title('Loop Network of Seven Neurons Linked by Gap Junctions')

plt.grid()

plt.show()


"""

Now Evaluate for 14-Param Model
```

```
"""



import rhs

reload(rhs)

rhs = rhs.rhs

from HH14 import hh14 as hh

import numpy as np

import scipy

from scipy.integrate import odeint

from scipy.integrate import ode

import matplotlib.pyplot as plt

from numpy import matlib


Iapp = 41.25

Nneurons = 7


ep=np.matlib.zeros((Nneurons,Nneurons))

ep[1,0]=1

ep[2,1]=1

ep[3,2]=1

ep[4,3]=1

ep[5,4]=1

ep[6,5]=1

#ep[0,6]=1
```

```
#ep[3,0]=1

#ep[6,1]=1

#ep[5,2]=1

#ep[2,0]=1

#ep[5,3]=1

#ep[6,4]=1

#ep[0,3]=1

epsilon = np.asarray(ep)



def Iext(t):

return [41.25,0,0,0,0,0,0]


V_IC = -86.8091

n_IC = 0.0536886

m_IC = 0.00178839

h_IC = 0.986998


y0=np.zeros(Nneurons*4)

for i in range(Nneurons):

y0[4*i:4*(i+1)]=[V_IC, n_IC, m_IC, h_IC]


t0=0


t = np.linspace(0,50,1001)
```

```
args = (Nneurons, epsilon, hh, Iext)

soln2=odeint(rhs,y0,t,args)


plt.figure(2)

for i in range(Nneurons):

        plt.plot(t,soln[:,4*i],'b', t, soln2[:,4*i],'r', linewidth=2.0)


plt.legend(loc='best')

plt.xlabel('t')

plt.ylabel('V[t]')

plt.title('Loop Network of Seven Neurons Linked by Gap Junctions')

plt.grid()

plt.show()
```

E.2 rhs.py

```
import numpy as np


def rhs(y,t,Nneurons,epsilon,hh,Iext):

        Vs = y[0::4]

        ydot = np.zeros(4*Nneurons)

I_ext= Iext(t)

        for i in range(Nneurons):

Iapp= I_ext[i]
```

```
                for j in range(Nneurons):
Iapp += (epsilon[i][j]*(Vs[j]-Vs[i]))
                V,n,m,h = y[4*i:4*(i+1)]
                ydot[4*i:4*(i+1)]=hh(V,n,m,h,Iapp)

        return ydot
```

E.3 HH25.py

```
import math

def hh25(V,n,m,h,Iapp):

x=[3.58351893845611,4.787491742782046,-1.2039728043259361,-5.250204516787248,\
4.296109496745657,-5.159088327040512,-4.605170185988091,-4.605270170991424,\
2.302585092994046,-2.0794415416798357,-4.78756117999381,4.382026634673881,\
-2.3025850929940455,-4.276858964458208,2.302585092994046,1.3862943611198906,\
-4.78756117999381,2.8903717578961645,-2.659260036932778,-4.78756117999381,\
2.995732273553991,0,-4.09462222433053,2.302585092994046,0]

GK = math.exp(x[0])
GNa = math.exp(x[1])
GL = math.exp(x[2])
EK = math.sinh(x[3])
```

```
ENa = math.sinh(x[4])

EL = math.sinh(x[5])

an = math.exp(x[6])

Van = math.sinh(x[7])

Kan = math.exp(x[8])

bn = math.exp(x[9])

Vbn = math.sinh(x[10])

Kbn = math.exp(x[11])

am = math.exp(x[12])

Vam = math.sinh(x[13])

Kam = math.exp(x[14])

bm = math.exp(x[15])

Vbm = math.sinh(x[16])

Kbm = math.exp(x[17])

ah = math.exp(x[18])

Vah = math.sinh(x[19])

Kah = math.exp(x[20])

bh = math.exp(x[21])

Vbh = math.sinh(x[22])

Kbh = math.exp(x[23])

Cm = math.exp(x[24])


if abs(V - Van) > 1e-15:

A_n = an*(V - Van)/(1 - math.exp(-(V - Van)/Kan))

else:
```

```
A_n = an*Kan


if abs(V - Vam) > 1e-15:

A_m = am*(V - Vam)/(1 - math.exp(-(V - Vam)/Kam))

else:

A_m = am*Kam


A_h = ah*math.exp(-(V - Vah)/Kah)


B_n = bn*math.exp(-(V - Vbn)/Kbn)

B_m = bm*math.exp(-(V-Vbm)/Kbm)

B_h = bh /( 1.0 + math.exp(-(V-Vbh)/Kbh))


t_n = 1 /(A_n + B_n)

t_m = 1 /(A_m + B_m)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = A_m /(A_m + B_m)

h_inf = A_h /(A_h + B_h)


Vdot =(Iapp - GK*(n**4)*(V-EK) - GNa*(m**3)*h*(V-ENa)-GL*(V-EL))/Cm

ndot =(n_inf - n)/t_n

mdot =(m_inf - m)/t_m

        hdot =(h_inf - h)/t_h
        return Vdot, ndot, mdot, hdot
```

E.4 HH14.py

```
import math



def Step(x):

return 0.5*(math.tanh(10.*x)+1.0)

def hh14(V,n,m,h,Iapp):


        x=[3.7667763268469816,

8.591364412633121,

-.6962264330078806,

-5.273603490788323,

-5.260297559459469,

-5.11925709055039,

-1.7772604296394117,

.026670164755117125,

2.4459173154152882,

-1.7588923758373796,

.4293570431447797,

-4.537740282109125,
1.8811461146761448,
1.1435726560783879]



    GK = math.exp(x[0])
    GENa = math.sinh(x[1])
    GEL = math.sinh(x[2])
    EK = math.sinh(x[3])
    an = math.exp(x[4])
    Van = math.sinh(x[5])
  b_tilde_n = math.exp(x[6])
     bmaVm = math.sinh(x[7])
     Kbm = math.exp(x[8])
     a_tilde_h = math.exp(x[9])
     bh = math.exp(x[10])
     Vbh = math.sinh(x[11])
    Kbh = math.exp(x[12])
    Cm = math.exp(x[13])

A_n = an*(V - Van)*Step(V - Van)
A_h = a_tilde_h

B_n = b_tilde_n
B_h = bh /( 1.0 + math.exp(-(V-Vbh)/Kbh))
```

```
t_n = 1 /(A_n + B_n)

t_h = 1 /(A_h + B_h)


n_inf = A_n /(A_n + B_n)

m_inf = 1/(1 + bmaVm*math.exp(-V/Kbm))

h_inf = A_h /(A_h + B_h)


Vdot =(Iapp - GK*(n**4)*(V-EK) + GENa*(m_inf**3)*h+GEL)/Cm

ndot =(n_inf - n)/t_n

mdot = 0

        hdot =(h_inf - h)/t_h


        return Vdot, ndot, mdot, hdot
```