University of Nevada, Reno

Modeling Sodium Channel Functional Variation in Garter Snakes With Known Tetrodotoxin Resistance Mutations

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Mathematics

by

Ryan Gustafson

Dr. Paul Hurtado/Thesis Advisor

May, 2023

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

RYAN GUSTAFSON

entitled

Modeling Sodium Channel Functional Variation in Garter Snakes With Known Tetrodotoxin Resistance Mutations

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Paul Hurtado, Ph.D.
Advisor

Deena Schmidt, Ph.D.
Committee Member

Chris Feldman, Ph.D.
Graduate School Representative

Markus Kemmelmeier, Ph.D., Dean
Graduate School

May, 2023

Abstract

The action potential is fundamental to animal life as it gives a mechanism for rapid communication between cells, organs, and body regions, and a means for rapid muscular and nervous activity. Action potentials are bioelectrical signals that are carried through the body via neurons or other excitable cells and work by displacing ions in across cells which creates a propagating charge gradient. Interfering with the machinery that makes action potentials possible can be detrimental to organisms, even fatal.

Sodium ion channels (Na_V) are key players in the production of action potentials (as well as potassium ion channels). These channels allow for a voltage-gated regulation of the distribution of sodium and potassium ions in nerve and muscle cells. Opening these channels allows for rapid influx (Na^+) and a rapid efflux (K^+) of ions across the cell membrane. This rapid movement of ions creates the propagating electrical current in the nerve and muscle cells, indicating the intent to fire or contract. Without this signal, nerves and muscles remain inactive. Because this conserved molecular machinery is needed for organismal function, it has become the target of a host of neurotoxins. Indeed, Pacific newts (*Taricha*) possess tetrodotoxin (TTX). TTX blocks the pore of the sodium ion channels, stopping the flow of ions across the membrane, and rendering any newt predator paralyzed. This includes respiratory and cardiac muscles which leads to asphyxiation and cardiac arrest -- death.

However, there are some populations of garter snakes (*Thamnophis*) that can prey upon these deadly newts. Evolution has offered a partial solution to these predators in the form of a handful of mutations in the sodium ion channels of the snake's muscle cells. These mutations decrease the ability for TTX to bind to sodium-ion channels which allow the snake to remain mobile even with TTX in its system (referred to a "TTX-resistance"). But, this small change in the shape of the pore of the sodium-ion

channel affects the ability of the channel to conduct sodium ions in the absence of TTX. So, snakes with these modifications have a lessened baseline muscle function. So, there is a trade-off that occurs where the TTX-resistant snakes have an expanded menu, but they lose some motility (which could result in their own predation).

In this thesis, the aim is to determine if the action potentials from the aquatic garter snake (*Thamnophis atratus*) with a single point mutation are phenotypically different from those *T.atratus* with no mutation (referred to as "conserved" or "TTX-sensitive"). To do this, a model using first-order linear differential equations will be produced that approximates experimentally collected action potential data (sometimes referred to as a trace). This model will then be simplified to use the least number of parameters possible. With the simplified equations, the model will be fit to the experimental data via an optimization routine that uses least squares as its objective function. At this point, it will be apparent that there are practical identifiability issues where the model does not produce a unique solution for the same action potential trace when given different starting points. This issue will be mitigated by only optimizing a subset of the possible parameters and by implementing a multi-start approach. The parameters that are optimized are the free parameters. This means that some parameters will be set to some fixed value during the optimization.

Because some parameters will be fixed, and the true value of those parameters is unknown (because of the identifiability issue) many different fixed parameter sets will be generated, and the free parameters (once optimized) will all be traded as part of a group, sampled from a single population. This will be done on both the TTX-sensitive action potential data and the TTX-resistant action potential data. With those two groups, a group-level comparison will be performed (via a linear mixed effects model), and the differences that are found between the two groups will be the basis for explaining the differences between the group.

It was found that the parameter that describes the conductance of sodium (G_{Na}), in the mechanistic model, was less in the mutant snakes than in the snakes with the ancestral condition of no sodium channel mutations. This provides evidence that the modifications that confer resistance to TTX reduce the ability of those snakes to conduct sodium ions and have enough impact to be quantified by the ensemble phenotype of the action potential. In addition to the sodium conductance, two other parameters in the mechanistic model were found to be significantly different between the groups. In both cases the parameter change indicated that the sodium ion channel exists in a less permissive (to sodium ions) state in the mutant snakes, further corroborating the initial finding. This also opens up the doors for further research into the true action of the sodium-ion channels in the mutant snakes in order to find a biochemical reason for this restriction of ion flow.

Dedication

I dedicate this to my parents for caring for and supporting me through my educational journey.

Acknowledgments

I want to acknowledge Robert del Carlo, Ph.D. for the data collection of the action potential traces analyzed in this work, as well as his substantial contribution to my knowledge of the biochemistry of action potentials. I also want to thank him for his unfaltering support in the completion of this project; he is just as much a part of my committee as everyone else -- if not more -- but could not be formally listed as a committee member. I also want to acknowledge Medha Vallurupalli for cleaning the action potentials to have standardized features for easier computational comparison.

Table of Contents

List of Tables

List of Figures

List of Programs

1 Introduction

Consider, briefly, a gazelle nested safely in the dry, orange grasses of the Sahara. It is unaware of a lion slowly inching toward her next meal. In the blink of an eye, the lion lunges, but without more than a few milliseconds delay, the gazelle is running, at top speed, away from the lion. This feat of orchestrating the movement of vast swaths of tissue (principally muscle tissue) happens nearly instantaneously as far as the gazelle or the lion are concerned. It, like other animals, has an organ that is dedicated to decision-making (the brain) that needs to communicate its intent to flee to the muscular and cardiovascular systems without leaving any time to become a lion's dinner. On the scale of a large animal, the diffusion of molecules and ions is so slow that it might take weeks for a signal from the brain to reach the organs need to arouse the gazelle into action. The circulatory system is inadequate for the task as well. A hormone secreted by the brain can travel to the rest of the body quickly, certainly -- in only a couple of heartbeats. But that delay is enough to render the gazelle non-viable to ever respond to a threat again.

There is another communication system that can send messages from the brain to the desired organ in milliseconds -- fast enough to instruct the body to escape from a predator -- the nervous system. This system relays its messages via electrochemical impulses called action potentials. These electrical signals work by rapidly changing the local ion concentration -- at a spatial scale where diffusion is useful -- within the cell to create a movement of positive charge from the start to the end of a nerve cell. The movement of this positive charge is much quicker than the diffusion of ions or hormones alone and is enough to help the gazelle live to see another day.

This ability to send information via electrical signals (action potentials) is pervasive across the entirety of the animal kingdom and functions via the movement of sodium and potassium ions across the cellular membrane via their respective ion channels

(discussed in greater detail in Section 1.3). An interesting interaction between a specific group of predatory snakes and their newt prey demonstrates how this ubiquitous system can be compromised -- but at a cost. The insights that can be uncovered by investigating this predator-prey interaction have broad-reaching consequences for all members of the animal kingdom (including humans) in terms of understanding their underlying biochemical machinery. From the human perspective, without action potentials, a person could take a breath or a heartbeat, enjoy good food, or even consider the implications of this work, so the action potential's fidelity is of great importance for survival. An understanding of what happens when nerves and muscles can no longer produce action potentials (as is the case with the neurotoxin described below) would offer insights into organismal function, including the human nervous and muscular systems.

1.1 Snake-Newt System

In western North America, some garter snakes (*Thamnophis*) live in close proximity to Pacific newts (*Taricha*) and exploit newts as a food source (Brodie and Brodie, 1999; Brodie et al., 2002; Feldman et al., 2009). However, all species of *Taricha* possess tetrodotoxin (TTX) (Brodie et al., 1974; Hanifin, 2010) which diminishes the ability to produce an action potential (Heistracher and Hunt, 1969) causing a reduction in the snake's ability to contract its own muscles (Brodie and Brodie, 1999). This is caused by an interaction of TTX with voltage-gated sodium-ion channels in the snake's skeletal muscle ($Na_v1.4$). The terminology, $Na_v1.4$, describes a sodium (Na) channel that is activated by voltage changes (subscript v) and is the fourth type (1.4) which is the type expressed in skeletal muscle by the *SCN4A* gene. Even at very low (nanomolar) concentrations of TTX, a snake may be paralyzed and subsequently die (Hanifin, 2010). However, there exist populations of *Thamnophis* that have mutations

in the *SCN4A* gene which produces a different version of the $Na_v1.4$ protein that reduces the binding affinity of TTX in skeletal muscle (Geffeney et al., 2005; Feldman et al., 2009). This results in a phenotype that is resistant to TTX exposure (Brodie et al., 2002; Geffeney et al., 2005; Feldman et al., 2010).

The reason that some *Thamnophis* have the capacity to resist TTX poisoning is due to a coevolutionary arms race between newts and snakes (Brodie and Brodie, 1999; Hanifin et al., 2008; Reimche et al., 2020). *Thamnophis* use newts as a food source, and in response, newts have evolved to produce TTX. This has imposed an evolutionary pressure onto *Thamnophis* spp. which, in return, has responded by accumulating genetic mutations that have reduced the toxicity of TTX. This has become a back-and-forth process whereby newts respond by producing more TTX which drives snakes to develop more resilient mutations to protect themselves against TTX. For example, in the case of the aquatic garter snake, *T. atratus*, two mutant populations have evolved -- a single point mutation population (A1281P mutation -- the "P" mutant group, for short) and a triple point mutation population (D1277E + A1281P + D1568N -- aka. the "EPN" mutant group).

However, these mutations do not appear to perfectly solve the problem of a toxic diet. While these mutations lead to snakes that are resistant to TTX, they appear to be associated with deleterious muscle or organismal function (Hague et al., 2018; del Carlo, 2020; Moniz et al., 2022). The research leading up to this point has postulated that there is a negative effect on the functionality of the sodium channels in TTX-resistant snakes.

1.2 Qualifying Physiological Cost

One question that arises from the above studies is the extent to which TTX-resistant snakes are compromised at the molecular level. Compromised sodium channels might

Figure 1: Pictures of *Taricha sierrae* (top row) and *Thamnophis* (*T. atratus* bottom left, *T. sirtalis* bottom right).
Photo credit: Devon A Picklum and Robert E del Carlo

manifest as a reduction in the ability to conduct sodium ions across the cell membrane secondary to a sterically-constrained ion channel due to a small number of permissible amino acid substitutions when compared to conserved (TTX-sensitive) *Thamnophis* (Feldman et al., 2012).

Experimentally derived traces of the net ion activity in the muscle cells (action potentials) were gathered to investigate this question in *T. atratus* by del Carlo (2020). It is difficult to directly compare action potentials between conserved and mutant (TTX-resistant) snake action potentials. This arises due to the nature of the action potential being a composite phenotype with many underlying molecular phenomena. Extracting precise information about just one of these molecular mechanisms (sodium conductance) from the entire aggregate electrical activity (action potential) is a crude substitute for more sophisticated analysis. It was previously found, by del Carlo (2020), that there was a decrease in the ability of a single sodium-ion channel to conduct sodium in snakes that possessed the triple point mutation (EPN). Specifically, the TTX-sensitive snakes had a channel conductance of 22.4 ($\pm$0.7)pS and the EPN mutants had a channel conductance of 12.4 ($\pm$0.8)pS. A quadruple point mutant (LVNV) from *T. sirtalis* had a channel conductance of 10.3 ($\pm$1.7)pS suggesting that more mutations, despite cofering greater resistance to TTX, seriously disadvantage the channel's performance. It is also noted that these mutations in *T. sirtalis* have been associated with functional trade-offs (Lee et al., 2011; Hague et al., 2018; del Carlo, 2020).

To determine the effect of the mutated sodium channels on the action potential phenotype, it is possible to model an action potential using ordinary differential equations (Hodgkin and Huxley, 1952a). The original research into modeling action potentials performed by Hodgkin and Huxley was done on the neuron of a squid giant axon (*Loligo paeleii*), but the underlying mathematical

machinery remains the same for modeling action potentials in skeletal muscle of *T. atratus*.

In this thesis, I seek to mathematically demonstrate that the deleterious effect of mutations altering the pore of the sodium channel is present at the level of the action potential which is a complex interaction of the action of many cellular mechanisms. Specifically, it is expected that the conductance of the sodium channels in mutant populations will be diminished when compared to the TTX-sensitive populations, and is dramatic enough that it can be inferred from the action potentials alone (indicating that the negative impact of the mutated channels is not fully compensated for at the neuromuscular level).

1.3 Cellular Biology Background

To properly motivate the reasoning behind the development of this model to determine the biochemical differences between two species of *Thamnophis*, an explanation of the underlying mechanisms of the action potential is warranted (Gerstner et al., 2014). In animals, there exists a mechanism for bio-electrical communication: the sending and receiving of information about the body via electrical impulses. This mechanism can be thought of as a wire conducting electricity, and the analogy to a nerve cell is a clear example of this phenomenon since nerve cells' principal utility is in sending or receiving information about the state of the body to and from the central nervous system/brain. However, this same mechanism is utilized by skeletal muscles to receive signals to contract muscle tissue. In neurons, the cell is tasked with ferrying a high-fidelity electrical signal from one side to the other as a means of ‘‘communicating" with other cells; the neuron is merely a messenger. The way the neuron does this is by moving an electrical potential down the length of its axon. At ‘‘rest", the neuron maintains a state where the inner surface of the plasma membrane exhibits a greater concentration

of negative charges relative to the outside, thereby creating an electrical potential across the cell membrane which ranges from $-55mV$ to $-80mV$ depending on the type and location of the cell in the body. This baseline potential of the cell that is achieved while the cell is not involved in sending electrical signals is called the resting membrane potential (RMP).

When the cell is tasked with sending a signal down the length of the axon (which is initiated by a voltage increase known as the stimulus), the key players are the sodium-potassium pump (Na^+-K^+-ATPase), the voltage-gated sodium-ion (Na^+) channel, and the voltage-gated potassium-ion (K^+) channels. Na^+-K^+-ATPase acts to maintain the negative electrical potential by expending energy in the form of adenosine triphosphate (ATP) to shuttle sodium and potassium ions against their electrochemical gradients such that there is more Na^+ on the outside of the cell and there is more K^+ on the inside of the cell. The signal itself is sent when the voltage-gated Na^+ channel opens and sodium ions flow into the cell causing the voltage to increase (become less negative) due to the increase in positive ions in the cell. This process is called *depolarization*. The influx of sodium ions is halted by a short amino acid sequence about $\frac{3}{4}$ between the C- and N-termini. This hydrophobic motif consists of an Isoleucine-Phenylalanine-Mehtionine plug which blocks the intracellular face of the pore in a voltage- and time-dependent manner (Yu and Catterall, 2003; Pan et al., 2018). Between the onset of activation and the completion of inactivation, enough sodium has rushed into the cell, down its concentration gradient, to change the cellular potential. This, in turn, causes nearby voltage-gated sodium channels to open, causing a cascade effect where the positive electrical potential appears to propagate down the axon, thereby sending information from one side of the cell to the other.

The process of reestablishing the RMP via Na^+-K^+-ATPase is relatively slow, so a compensatory mechanism is used to lower the cellular voltage more quickly while the

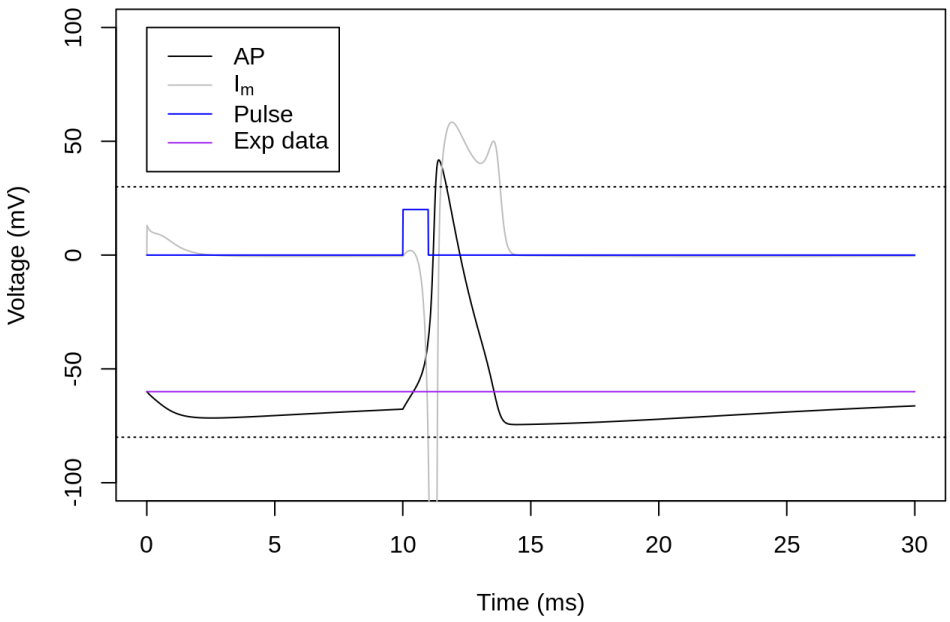

Figure 2: The black line is the voltage trace using the Hodgkin-Huxley parameters and equations. The purple line (in the rest of the document) is an example action potential of for reference, but here no neuron action potential data was available. The gray line is the current (I_m). The blue line is the current added to the system (the stimulus). The dotted black lines indicate reference voltages at $-80mV$ and $+30mV$.

baseline ion concentrations are being established. Voltage-sensitive K^+ channels open in response to the changing voltage caused by the movement of sodium ions into the cell, thereby causing a potassium ion efflux, reducing the voltage of the membrane called *repolarization*. This accomplishes two things. First, repolarization accelerates the pace at which a second action potential can fire by returning to RMP sooner. Second, sodium channels must be prepared to fire anew because the hydrophobic inactivation plug described earlier is only removed by negative intracellular voltages, such as the one established at RMP. Macroscopically, this process -- characterized by the depolarization and repolarization phases -- is called the action potential (Figure 2). The voltage of the cell can be measured with a multi-meter adapted for microscopic probing, resulting in a graph of the voltage over time, referred to as a trace.

In the context of the *Taricha-Thamnophis* system, TTX binds to the sodium

channel in the muscle cells of *Thamnophis* rendering the sodium channel unable to conduct sodium ions across the cellular membrane; thus, the action potential cannot propagate, and the muscle cell is effectively paralyzed by electrical rather than mechanical phenomena. The mutations that reduce the efficacy of TTX in *Thamnophis* work by modifying the amino acid residues where TTX primarily binds (Geffeney et al., 2005; Feldman et al., 2009). This changes the shape and biophysical properties of the sodium ion pore which is already a highly-conserved evolutionary structure in animals. So, changes to this structure are likely to have a negative effect on the ability of sodium ions to pass through the pore (Lee et al., 2011; Feldman et al., 2012; Hague et al., 2018; del Carlo, 2020). Previous research, as mentioned earlier, already noticed the negative effect on the tissue level (and on the animal level in the case of *T. sirtalis*). So, the question that needs to be addressed is whether or not the negative effect on the cellular (or animal) functional ability is related to a diminished conductance of sodium ions through the voltage-gated sodium channel.

Returning to the question of discerning information about the differences between species of *Thamnophis* with and without TTX resistance, comparing the two groups no longer comes down to comparing the two species' action potentials. Now, a comparison between the differences in the ion movement -- specifically as it pertains to the sodium channels -- should instead be made. However, it is difficult to directly measure the activity of sodium and potassium ion channels in their native tissues, often requiring complicated workarounds (expressing them transgenically in heterologous expression systems) that diminish the biological relevance of the results. Therefore, to capture the most biologically relevant insights from sodium channels (natively expressed in snake skeletal muscle), a mechanistic model will be used to fit experimentally-gathered action potential trace data to quantify the effects of the major contributors to action potential generation (Na^+ and K^+ activity).

1.4 Experimental Data

Experimental action potential data from *T. atratus* were collected by del Carlo (2020) and were cleaned by Medha Vallurupalli (Figures 3, 4, and 5) (unpublished data). This cleaning process involved selecting traces that were clearly the result of an experimentally stimulated action potential. The visual peak of the action potential was placed $2ms$ after the beginning of the trace (meaning the beginning of the experimental trace was trimmed such that the peak -- visually chosen -- occurred $2ms$ into the trace). This data was used to estimate parameters that best fit the data using the Hodgkin-Huxley model (Hodgkin and Huxley, 1952a).

1.5 Statistical Analysis

The model that will be used in this thesis was pioneered by Hodgkin and Huxley (Hodgkin and Huxley, 1952a). The original model as developed by Hodgkin and

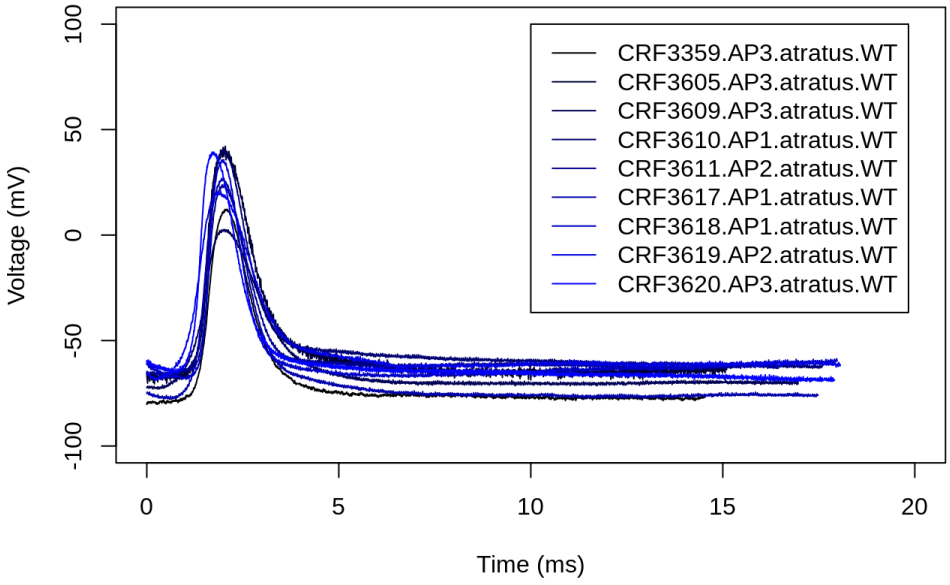

Figure 3: All traces for the experimental data on the TTX-sensitive *T. atratus*. Individual traces represent the voltage measurements taken at short time intervals ($5\mu s$).

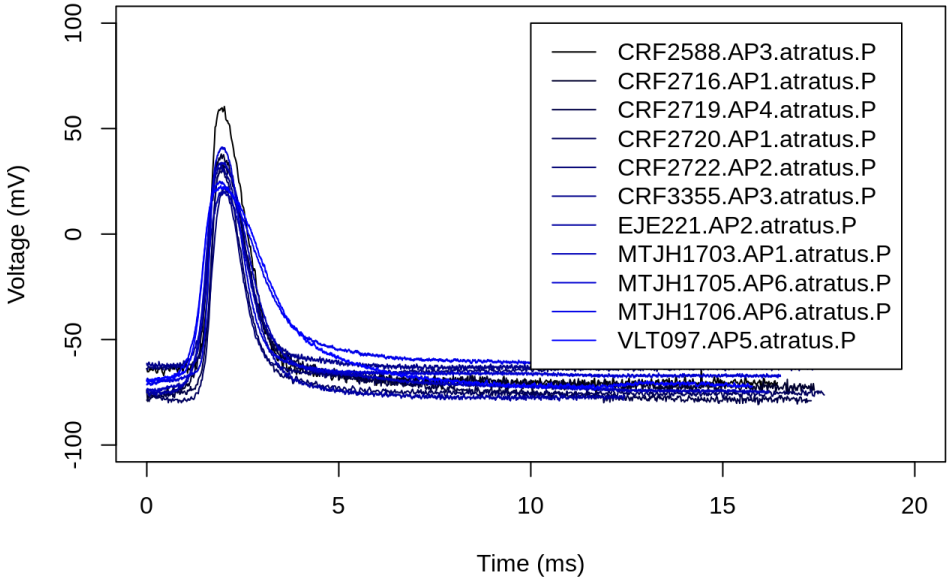

Figure 4: All traces for the experimental data on the single point mutant *T. atratus* (P). Individual traces represent the voltage measurements taken at short time intervals ($20\mu s$).

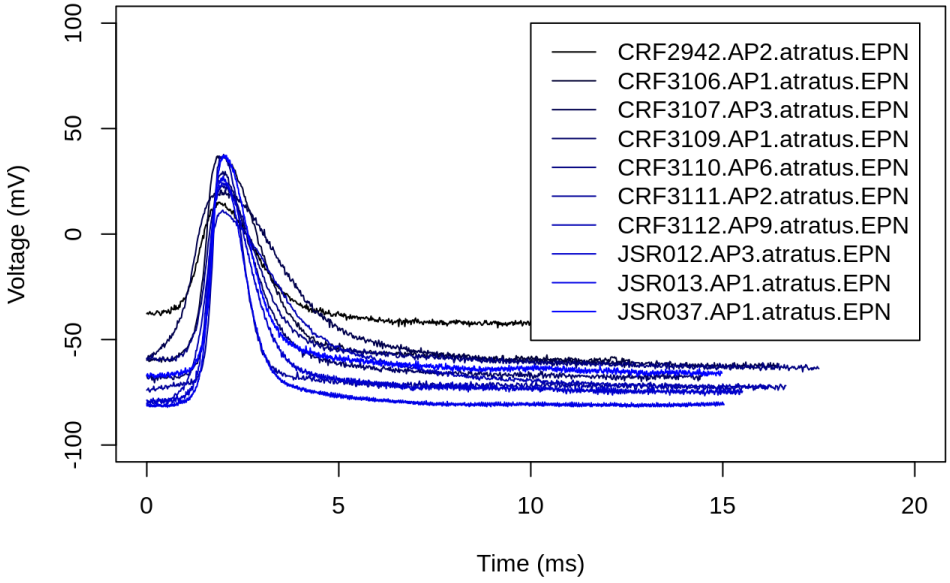

Figure 5: All traces for the experimental data on the triple point mutant *T. atratus* (EPN). Individual traces represent the voltage measurements taken at short time intervals ($20\mu s$).

Huxley is overparameterized, meaning that there are more variables than are needed to describe a system in an equivalent way. Therefore, the equations will be modified to be simpler, yet still capture the intended effect of the original model (model an action potential). Furthermore, an identifiability analysis will be conducted to ensure that the model provides unique solutions given a consistent parameter estimation technique (Section 3).

Once the Hodgkin-Huxley model is adapted to model the action potentials, group-level differences between two groups (+ and P) can be investigated using simulated data. This simulated data can then be analyzed so see which pieces of the action potential model change from one group to the other by using a linear mixed-effect model. The results from that analysis will be how differences between TTX-sensitive and P can be shown empirically.

2 Mechanistic Model

Ultimately, the goal is to achieve a model that reflects the cellular processes mentioned earlier. It is the aim to have a model that reproduces the action of the sodium ion channels and the potassium ion channels that can be tuned in depending on the experimental trace that is fed into the model. Given that the mechanistic model builds off the biochemical processes, by comparing the different types/degrees of tuning that are required to fit action potentials from each group, inferences can be made about the differences in mechanisms that underlie the production of action potentials.

The Hodgkin-Huxley model of the action potential (Hodgkin and Huxley, 1952a) simulates the ionic currents arising from the flux of sodium and potassium across the muscle cell membrane. The original model proposed by Hodgkin and Huxley was developed by deactivating certain ionic conductances and observing the commensurate change in current flow with respect to forced change in cell membrane voltage (so-called "Voltage-Clamp") (Hodgkin and Huxley, 1952c). The thinking is that the membrane of a cell that produces an action potential can be represented as an electrical circuit (Figure 6) based on the components involved in membrane conductance (Hodgkin and Huxley, 1952b).

In this circuit, the reservoirs of sodium and potassium ions ($\mathcal{E}_{Na}, \mathcal{E}_K$) can be treated as sources of electromotive force (emf), and can be thought of as the electrochemical drive of ions to move (if unimpeded) as described by the Nernst equation.

$$\mathcal{E} = \frac{RT}{zF} \ln \frac{[ion]_{\text{out}}}{[ion]_{\text{in}}} \tag{1}$$

Here, $\mathcal{E}_{Na}$ is an emf towards the outside of the cell and $\mathcal{E}_K$ is an emf towards the inside of the cell (electricity is interpreted as the movement of a negative charge, so the movement of positively charged ions is counter to the interpreted flow of negative

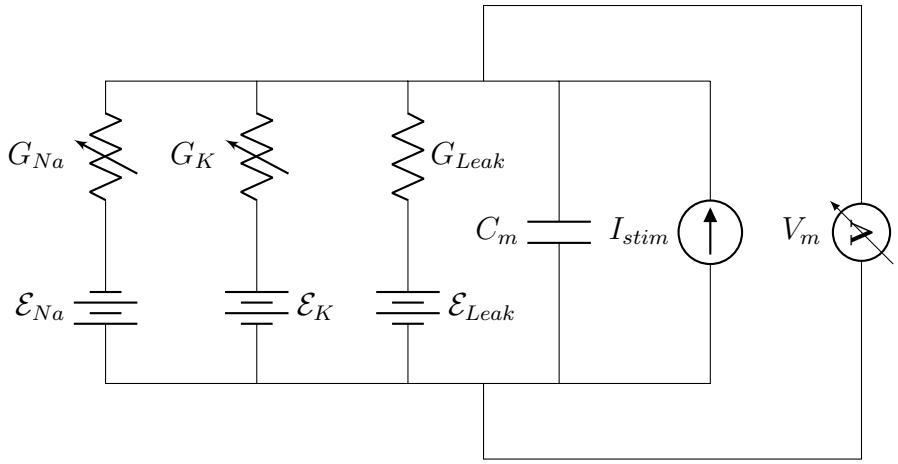

Figure 6: Electrical representation of the cellular membrane with sodium and potassium ion channels. $\mathcal{E}$ represents an electromotive force produced by ion reservoirs. G is the inverse of resistance. I_{stim} is the applied current. V_m is the measured voltage across the membrane.

charge). Both of those values are calculated with the values in Appendix A.3. Lastly, $\mathcal{E}_{Leak}$ is the emf set up by the resting membrane potential. Each of the emf sources is in series with a resistor which can be thought of as the inverse of conductivity (G):

$$G = \frac{1}{R} \tag{2}$$

In this case, conductivity is the ability of ions to cross the membrane (closely related to permeability). This is the property understood to determine the functional differences between TTX-sensitive and mutant populations of *Thamnophis*. To determine how to represent the conductivity, an investigation into the working of the sodium and potassium channels is in order.

The sodium channel in humans consists of four voltage-sensing domains (VSD) (Pan et al., 2018). Each of the four VSDs is sensitive to the voltage potential across the membrane, owing to positively charged alpha helices. At the very negative RMP, the positively charged VSDs will be attracted intracellularly while at more positive potentials, the VSDs are repelled extracellularly. Even at RMP, VSD-IV is always in

the repelled state. The remaining three VSDs are only repelled by sufficient increases in membrane potential. Once all four VSDs are repelled extracellularly, they will pull on the pore-forming domains (PFD), dilating a space large enough for hydrated sodium ions to passively conduct through the channel (i.e. according to the concentration gradient and electric field across the membrane). Together, the movement of these three VSDs and their consequent conformational changes to the PFDs is referred to as ‘‘activation" (Yu and Catterall, 2003).

Since a voltage-conducting cell will have hundreds of sodium channels, the overall effect of sodium across the membrane can be treated as a stochastic process where m represents the probability that any given VSD is active at a given voltage. Since there are three VSDs that require activation above RMP, m^3 represents the probability that all of the necessary VSDs are active. On the scale of the cell, this probability acts as the percentage of active sodium channels with one caveat. As described earlier, there is a singular inactivation particle in the sodium channel that stops the flow of sodium even when all of the VSDs are activated. This inactivation will be written as h (to the first power), and thus, the equation m^3h gives the probability that the sodium channel as a whole is able to conduct sodium ions given the voltage of the cell (Hodgkin and Huxley, 1952d). Now, let $\overline{G_{Na}}$ be the maximum conductivity (in $\frac{1}{n\Omega}$, called nano Siemens) possible for the sodium channels in the membrane. Thus, the conductivity of sodium in a membrane can be written as the product of the maximal conductance and the probability of activity for each of the subunit particles, and the probability of activity of the inactivation particle.

$$G_{Na} = \overline{G_{Na}} m^3 h \tag{3}$$

On the other hand, the potassium ion channel has four voltage-sensing domains whose activity is variable over the physiological voltage range of an action potential

(Zakon, 2012). Importantly, voltage-gated potassium channels exhibit no known inactivation gate as seen in voltage-gated sodium channels. Consequently, their inactivation kinetics are much slower, whereby potassium current slowly tapers off rather than abruptly terminating with a molecular occlusion. One can therefore think of potassium channels more like dimmer switches whose current flows more at elevated potentials and whose current barely flows, if at all, closer to negative resting membrane potentials. These differences between voltage-gated sodium channels and voltage-gated potassium channels also have a molecular explanation: whereas Na_V are monomers with tertiary structure producing four-way radial symmetry, K_V are tetramers with quaternary structure producing four-way radial symmetry. As such, the four independent molecules that interact to form the voltage-gated potassium channel (K_V) are treated as both activation and inactivation gates, accounted for by a single voltage-dependent probability "n" that one potassium VSD is active at a given voltage in a similar manner as before. Furthermore, the total conductance of potassium across the membrane (in $\frac{1}{n\Omega}$) can be given by

$$G_K = \overline{G_K} n^4 \tag{4}$$

Figure 7 gives the conductance for each sub-unit particle for a given voltage when given infinite time to "become active".

The last conductance, $\overline{G_{Leak}}$, is the background conductance of charge across the membrane that is not contributed to by sodium or potassium and does not appear to be threshold-ed in similar ways, rather producing a more linear conductance-voltage relationship. This arises from the voltage-dependent activity of NA^+-K^+-ATPase and the movement of other ions such as calcium, but that contribute much less to the variability in voltage. Finally, to complete the description of the circuit: C_m is the capacitance across the membrane ($1\frac{\mu F}{cm^2}$) (Ohki, 1969); the current applied to the

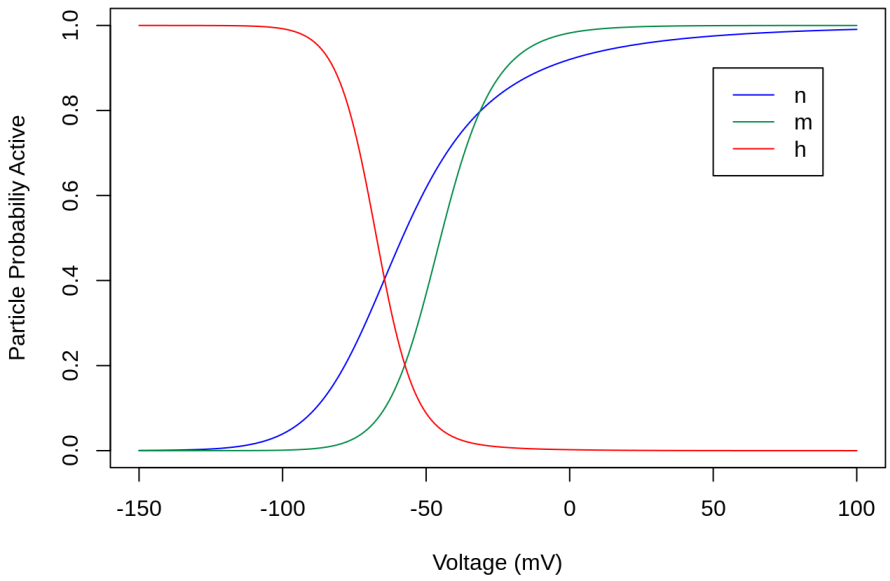

Figure 7: Plots of steady-state equilibria (s_∞) for n, m, and h particles over a range of voltages using the Hodgkin-Huxley values. (Hodgkin and Huxley, 1952a).

membrane is I_{stim}; and the total membrane voltage -- and the value that is read by the volt-meter -- is V_m. The need for these features will be disused later on. Note that $C_m \frac{dV}{dt}$ produces a current because a change in voltage, especially a change applied over a short time scale, requires ions to rearrange themselves to find new stable equilibria, and the movement of ions constitutes current. This motivates the need to determine equations for the current across the membrane.

2.1 Mathematical Model Derivation

Returning to Figure 6, the total current generated by the ions is important for building the model. So, using Kirchhoff's laws, the three current sources in parallel can be rewritten as a single current source (current across the membrane -- I_m) which is a sum of the ion currents through the membrane:

$$I_m = I_{Na} + I_K + I_{Leak} \tag{5}$$

Where I_{Na} is the current due to sodium, I_K is the current due to potassium, and I_{Leak} is the constitutive current developed primarily by calcium flow and the activity of Na^+-K^+-ATPase (all currents are in nA). Note that in this circuit, we can use the equation for electrical circuits, $V = IR$, to achieve $I = \frac{V}{R}$. By Equation 2, the new description of current can be written as $I = GV$. In this case, the voltage in question is the driving force $(V_m - \mathcal{E}_{ion})$ of the ion reservoirs distributed across the membrane and is dependent on the electrical gradient (that is, on the present voltage across the membrane). Thus, the individual current equations can be described by,

$$I_K = G_K(V_m - \mathcal{E}_K) = \overline{G_K} \cdot n^4(V_m - \mathcal{E}_K) \tag{6}$$

$$I_{Na} = G_{Na}(V_m - \mathcal{E}_{Na}) = \overline{G_{Na}} \cdot m^3 h(V_m - \mathcal{E}_{Na}) \tag{7}$$

$$I_{Leak} = \overline{G_{Leak}}(V_m - \mathcal{E}_{Leak}) \tag{8}$$

Where V_m is the total voltage across the membrane (in mV) and all other variables are as given above. Thus, by combining Equations 6, 7, and 8 with Equation 5, the total membrane current can be given as a function of voltage by,

$$I_m = \overline{G_K} \cdot n^4(V_m - \mathcal{E}_K) + \overline{G_{Na}} \cdot m^3 h(V_m - \mathcal{E}_{Na}) + \overline{G_{Leak}}(V_m - \mathcal{E}_{Leak}) \tag{9}$$

Now, given that

$$I = C_m \frac{V_m}{dt} \tag{10}$$

describes the current across a lipid bilayer where I is the total current across the membrane by Ohm's law of a capacitor (Hodgkin et al., 1952), rearranging gives

$$\frac{dV_m}{dt} = \frac{I}{C_m} \tag{11}$$

This equation describes a method to update the voltage of the model after a time step. In the experimental setup, the total current is the added effect of I_m and the applied stimulus I_{stim}. Thus, the change in membrane potential over time can be described as

$$\frac{dV_m}{dt} = \frac{I_{stim} - I_m}{C_m} \tag{12}$$

Next, a method for updating the conductance parameter after each time step. Let α be the rate that sub-units are converted from the non-permissive state to the permissive state as a function of voltage, and let β be the reverse rate. Using the notation $s \in \{n, m, h\}$ be a sub-unit with a probability of activation (or inactivation for h) as before,

$$\frac{ds}{dt} = \alpha_s(V_m)(1 - s) - \beta_s(V_m)s \tag{13}$$

This gives the needed method for updating each sub-unit. Nevertheless, investigating this model uncovers some insights that are helpful for initializing the model. By rearranging the equation,

$$\begin{aligned} \frac{ds}{dt} &= \alpha_i(1 - s) - \beta_s s \\ &= \alpha_s - \alpha_s s - \beta_s s \\ &= \alpha_s - (\alpha_s + \beta_s)s \\ &\Longrightarrow \\ \frac{1}{\alpha_s + \beta_s}\frac{ds}{dt} &= \frac{\alpha_s}{\alpha_s + \beta_s} - s \end{aligned}$$

the new version of the differential equation can be viewed as

$$\frac{ds}{dt} = \frac{s_\infty(V) - s}{\tau_s(V)} \tag{14}$$

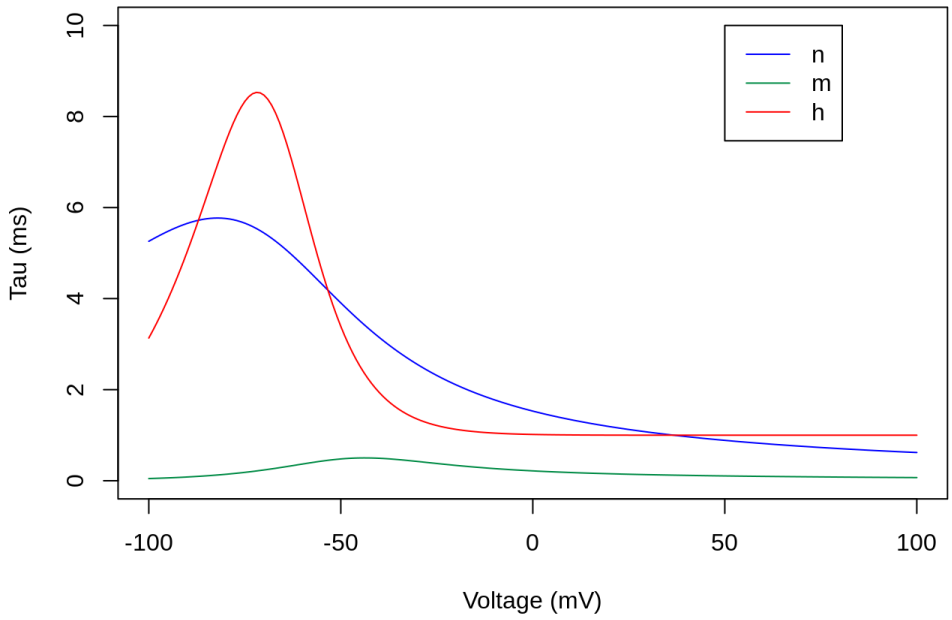

Figure 8: Plots of time constants (τ_s) for n, m, and h particles over a range of voltages using the Hodgkin-Huxley values.

where s_∞ is the steady-state solution (Figure 7)

$$s_\infty = \frac{\alpha_s(V)}{\alpha_s(V) + \beta_s(V)} \tag{15}$$

and τ_s is the time constant (Figure 8)

$$\tau_s = \frac{1}{\alpha_s(V) + \beta_s(V)} \tag{16}$$

That is to say that Equation 14 gives the magnitude of the update to the subunit particle in terms of its current distance to its maximum activity at a given voltage. The equation for S_∞ is used as the initialization value for the sub-unit particles given the starting voltage as required in the computational procedure. τ_s can be thought of as the granularity at which updates to s can be made in terms of seconds. Now, all that is needed to start the computational process is V_m which can be borrowed

from the experimental data. Nevertheless, the functions for α and β still need to be chosen. Going back to the Hodgkin-Huxley paper, the following equations were borrowed where the original values were substituted with constants (Figure 9). The functions α and β for n can be described as:

$$\alpha_n(V) = N_1 \frac{V + N_2}{e^{\frac{V+N_2}{N_3}} - 1} \tag{17}$$

$$\beta_n(V) = N_4 e^{\frac{V+N_5}{N_6}} \tag{18}$$

α and β for m

$$\alpha_m(V) = M_1 \frac{V + M_2}{e^{\frac{V+M_2}{M_3}} - 1} \tag{19}$$

$$\beta_m(V) = M_4 e^{\frac{V+M_5}{M_6}} \tag{20}$$

α and β for h

$$\alpha_h(V) = H_1 e^{\frac{V+H_2}{H_3}} \tag{21}$$

$$\beta_h(V) = \frac{1}{1 + e^{\frac{V+H_4}{H_5}}} \tag{22}$$

The constants N_1, N_2, N_3, N_4, N_5, N_6, M_1, M_2, M_3, M_4, M_5, M_6, H_1, H_2, H_3, H_4, and H_5 are chosen in order to get the best agreement with the experimental data.

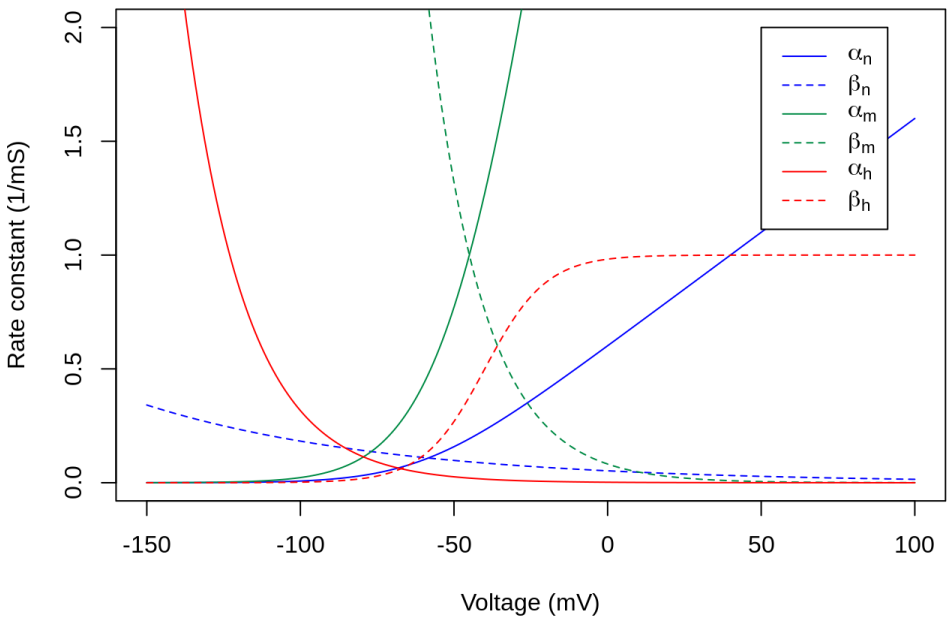

Figure 9: Plots of the α and β curves for the TTX-sensitive average data using the Hodgkin-Huxley values (Hodgkin and Huxley, 1952a).

2.2 Hodgkin-Huxley Model Equations

Let $\vec{p} = \{n, m, h, V\}$. Then the model can be described as (recalling that $I_m = \left[\overline{G_K} \cdot n^4(V - \mathcal{E}_K) + \overline{G_{Na}} \cdot m^3 h(V - \mathcal{E}_{Na}) + \overline{G_{Leak}}(V - \mathcal{E}_{Leak})\right]$):

$$\frac{d\vec{p}}{dt} = \begin{cases} \dfrac{dn(\vec{p})}{dt} = \left(N_1 \dfrac{V + N_2}{e^{\frac{V+N_2}{N_3}} - 1}\right)(1 - n) - \left(N_4 e^{\frac{V+N_5}{N_6}}\right) n & \text{(23a)} \\ \dfrac{dm(\vec{p})}{dt} = \left(M_1 \dfrac{V + M_2}{e^{\frac{V+M_2}{M_3}} - 1}\right)(1 - m) - \left(M_4 e^{\frac{V+M_5}{M_6}}\right) m & \text{(23b)} \\ \dfrac{dh(\vec{p})}{dt} = \left(H_1 e^{\frac{V+H_2}{H_3}}\right)(1 - h) - \left(\dfrac{1}{1 + e^{\frac{V+H_4}{H_5}}}\right) h & \text{(23c)} \\ \dfrac{dV(\vec{p})}{dt} = \dfrac{I_{stim} - I_m}{C_m} & \text{(23d)} \end{cases}$$

Where N_1, N_2, N_3, N_4, N_5, N_6, M_1, M_2, M_3, M_4, M_5, M_6, H_1, H_2, H_3, H_4, H_5, $\overline{G_K}$, $\overline{G_{Na}}$, $\overline{G_{Leak}}$, $\mathcal{E}_K$, $\mathcal{E}_{Na}$, $\mathcal{E}_{Leak}$, and C_m are known constants (Table 1), I_{stim} is given (as a function of time), and the initial V is determined experimentally. In

Parameter	Value	Used in	Units
N_1	0.01	α_n	-
N_2	60	α_n	-
N_3	-10	α_n	-
N_4	0.125	β_n	-
N_5	70	β_n	-
N_6	-80	β_n	-
M_1	-0.1	α_m	-
M_2	45	α_m	-
M_3	-10	α_m	-
M_4	4	β_m	-
M_5	70	β_m	-
M_6	-18	β_m	-
H_1	0.07	α_h	-
H_2	70	α_h	-
H_3	-20	α_h	-
H_4	40	β_h	-
H_5	-10	β_h	-
$\overline{G_K}$	36	I_m	$nanosiemens(nS)$
$\overline{G_{Na}}$	120	I_m	$nanosiemens(nS)$
$\overline{G_{Leak}}$	0.3	I_m	$nanosiemens(nS)$
$\mathcal{E}_K$	-75	I_m	$millivolts(mV)$
$\mathcal{E}_{Na}$	55	I_m	$millivolts(mV)$
$\mathcal{E}_{Leak}$	-50	I_m	$millivolts(mV)$

Table 1: Values for parameters as given in Hodgkin and Huxley (1952a)

this document the group of parameters $\{N_i : i \in \{1, 2, 6, 7\}\}$, $\{M_i : i \in \{1, 2, 6, 7\}\}$, $\{H_i : i \in \{3, 4, 5, 6\}\}$, and $\{G_i : i \in \{K, Na, Leak\}\}$ will be referred to as $\{N_i\}$, $\{M_i\}$, $\{H_i\}$, and $\{G_i\}$ respectively.

Altogether, this gives an action potential with currents from each channel as given in Figure 11 and individual conductances as given in Figure 10. Furthermore, one can see how the subunits evolve over time with a change in the voltage (Figure 12).

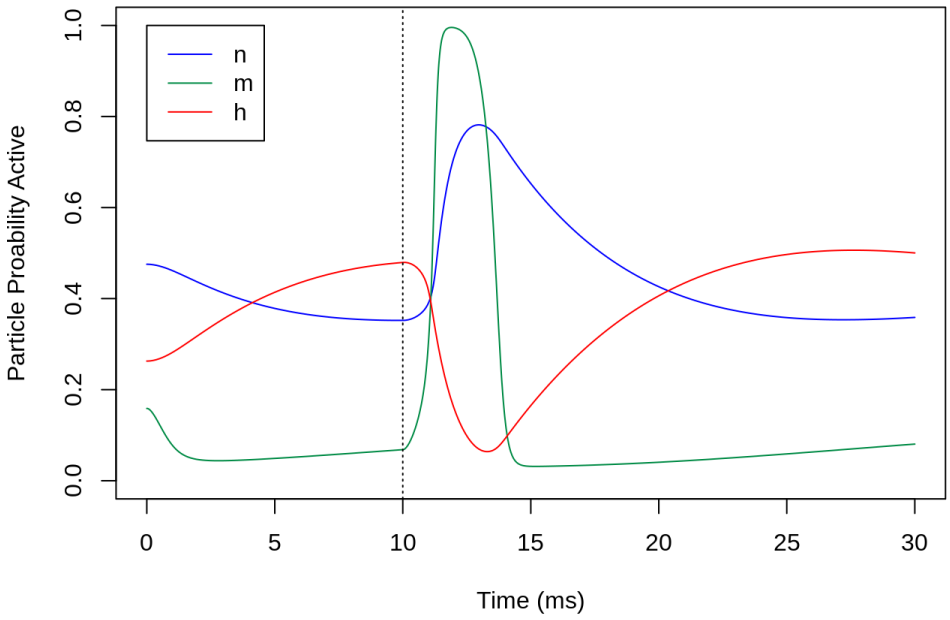

Figure 10: Plots of the conductance (the ability for the channel to allow ion flow) for sodium, potassium, and leak channels starting at $10ms$ based on the Hodgkin-Huxley values.

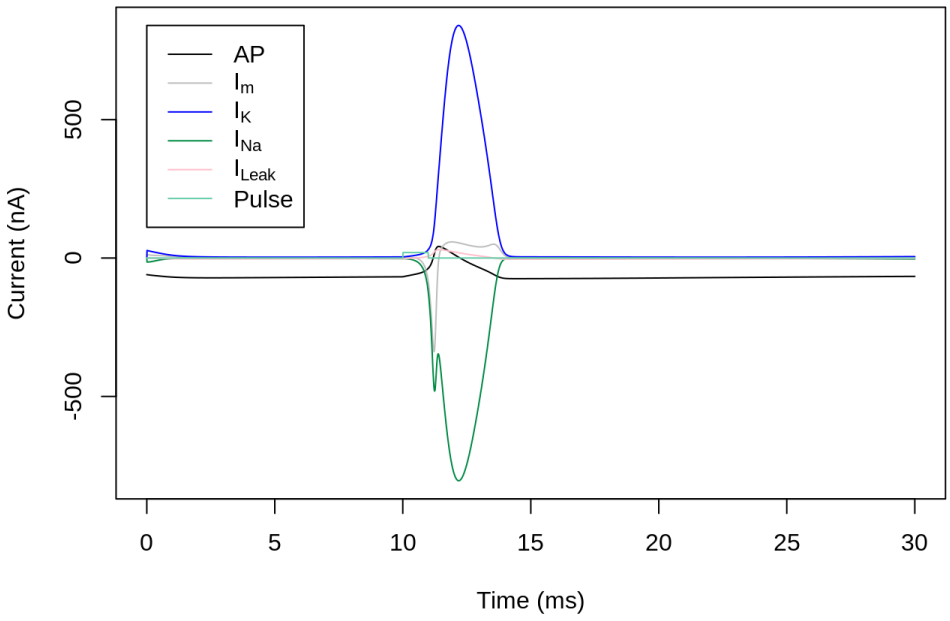

Figure 11: Plots of the currents for sodium, potassium, and leak channels based on the Hodgkin-Huxley values. Here the stimulus is applied after $10ms$.

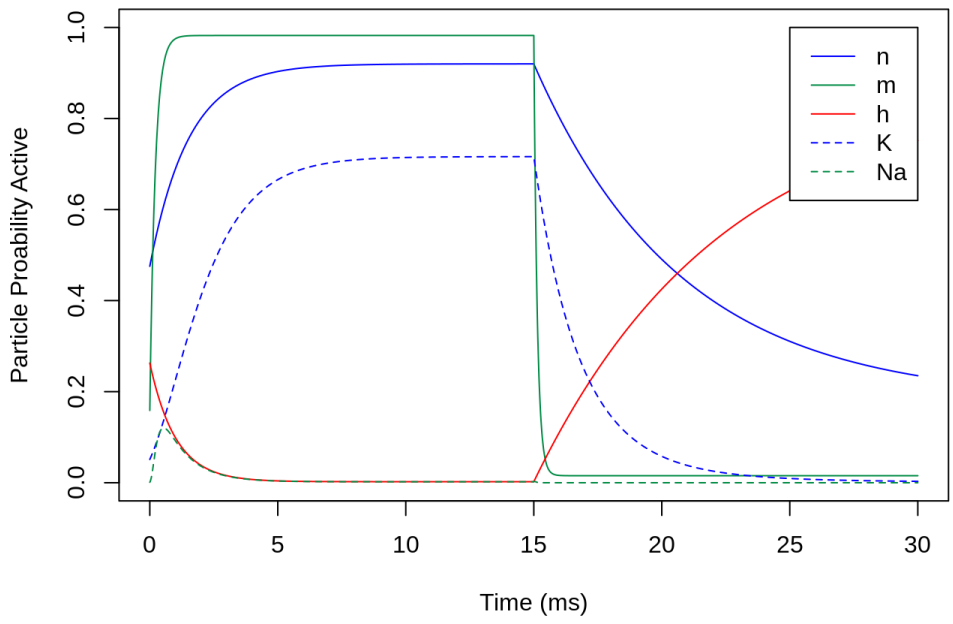

Figure 12: Simulated voltage-clamp experiment. Plots of the subunit activation percentages or n, m, and h particles. At time $t = 0ms$, the current was switched from $-80mV$ to $0mV$. After $15ms$ of equilibration, the current was switched back to $-80mV$.

3 Parameter Estimation Methods

The above mechanistic action potential model can be incorporated into a nonlinear regression framework to analyze data from *T. atratus* data (Section 1.4). That entails determining values for the constants (N_i, M_i, H_i, G_i) that best fit these experimental action potential data. At this point, a framework to fit each trace individually is needed. One problem arising from the complexity of the system is that the model will be overparameterized (Walch and Eisenberg (2016), Daly et al. (2015)). This means that there are more parameters than are necessary to uniquely model a given action potential trace. This lack of a best parameter set is known as an identifiability problem which will be addressed below (Section appendix B). Second, group-level comparisons between TTX-sensitive and mutant *T. atratus* will be needed. The approach used is detailed below and is applied to the *T. atratus* data in Section 4.

Once a methodology was found to uniquely fit a given trace, the two groups were fit separately (details to come) by producing a number of fixed parameter sets using Sobol sequences (Appendix D). Now using action potential traces generated to have the character of one of the two groups, each fixed parameter set was paired with a free parameter set and allowed to fit the generated action potentials. With this new set of optimized parameters, group-level statistics can be done. Computing the statistical differences in this way (rather than a direct comparison) helps resolve the practical identifiability problem by saying that the known values for the fixed parameters might not be correct, but if a variety of guesses are made and the (optimized) free parameters follow a statically significant trend regardless, group level statistical interpretations can be made on this system of high complexity.

3.1 Model Fit to a Single Trace

First, it is important that the model be simplified wherever possible, so a structural identifiability analysis will be performed to address the overparameterization concern. This will reduce the number of distinct parameters that are needed to model an action potential. With a simplified model, it can now be applied to the action potential data from *T. atratus*. To determine how well the model is at fitting the action potential data, an objective function will be developed using least squares that will numerically describe how well the model fits the data. At this point, the model is unable to find a unique solution. Using a multi-start method, it will be shown that, using a subset of the parameter space, a unique solution can be found. That is to say that some subset of parameters is allowed to be optimized (free parameters) while the rest are held at some reasonable value (fixed parameters). So, the last step in fitting the model to a trace will be determining what that subset is.

3.1.1 Structural Identifiablity

Consider the equation $f(x) = a \cdot e^{\frac{x+b}{c}}$ were a, b, c are constants. Take the set of constant values $a = e^4, b = -2, c = 2$. Then the equation becomes $f_1(x) = e^4 \cdot e^{\frac{x-2}{2}}$ (Figure 13). Now suppose that $a = e, b = -\frac{1}{2}, c = 2$. This gives $f_2(x) = e^{\frac{x-\frac{1}{2}}{2}}$. Clearly there are 2 distinct sets of parameters, however

$$f_1(x) = e^4 \cdot e^{\frac{x-2}{2}} = e^{\frac{x-2}{2}+4} = e^{x/2} = e^{\frac{x-\frac{1}{2}}{2}+1} = e \cdot e^{\frac{x-\frac{1}{2}}{2}} = f_2(x)$$

Hence the two equations are the same. Further, the equation can be reduced to a 2-parameter set

$$f(x) = a \cdot e^{\frac{x+b}{c}} = e^{\frac{x+b}{c}+\ln(a)} = e^{\frac{x+d}{c}}$$

where $d = b + \ln(a)c$.

Notice that in both cases $d = 0$. So it can be said that this three-parameter system is overparameterized and can be reduced (as described) to a 2 parameter system. Thus, there was a structural identifiability problem with the way the system was introduced. This is much the same as what was found with the Hodgkin-Huxley equations.

By inspecting the equations for β_n, β_m, and α_h, it can be seen that the three equations that govern those parameters are, in fact, only dependent on 2 parameters. The relevant demonstration is shown below. The equation for β_n can be simplified to

$$\beta_n(V) = N_4 e^{\frac{V+N_5}{N_6}} = e^{\frac{V+N_5}{N_6}+\ln(N_4)} = e^{\frac{V+N_5+\ln(N_4)N_6}{N_6}} = e^{\frac{V+N_7}{N_6}} \tag{24}$$

where $N_7 = N_5 + \ln(N_4)N_6$. Equally,

$$\beta_m(V) = M_4 e^{\frac{V+M_5}{M_6}} = e^{\frac{V+M_5}{M_6}+\ln(M_4)} = e^{\frac{V+M_5+\ln(M_4)M_6}{M_6}} = e^{\frac{V+M_7}{M_6}} \tag{25}$$

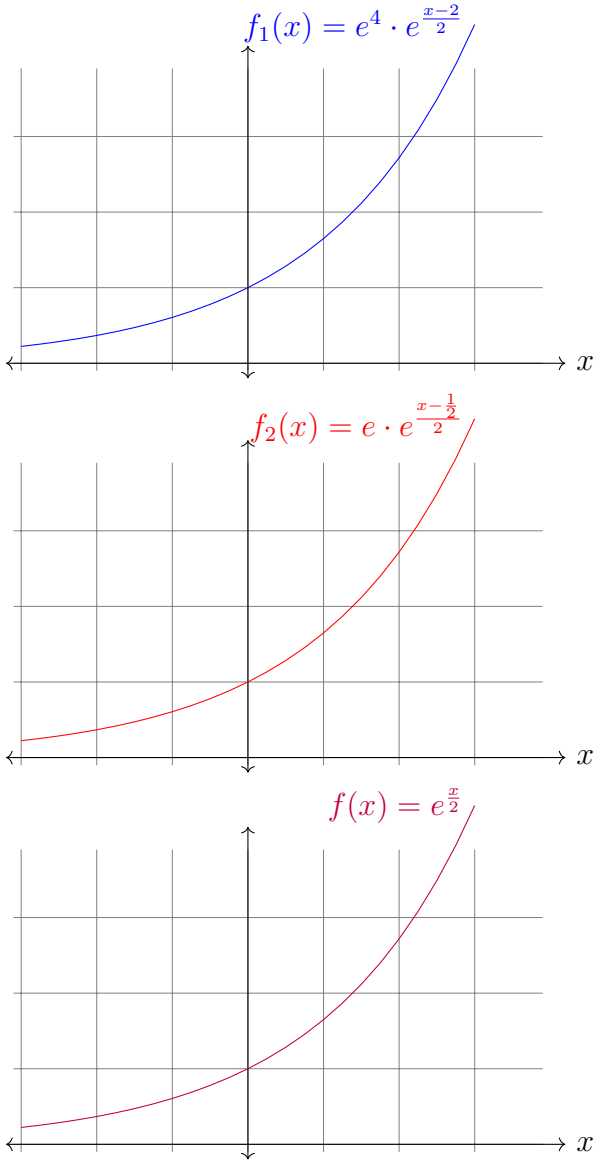

Figure 13: Example showing structural identifiability. All three equations give the same graph.

where $M_7 = M_5 + \ln(M_4)M_6$. And,

$$\alpha_h(V) = H_1 e^{\frac{V+H_2}{H_3}} = e^{\frac{V+H_2}{H_3}+\ln(H_1)} = e^{\frac{V+H_2+\ln(H_1)H_3}{H_3}} = e^{\frac{V+H_6}{H_3}} \tag{26}$$

where $H_6 = H_2 + \ln(H_1)H_3$.

Upon inspection of the graphs of α_n, it appeared that the function was mostly linear over the range of voltages used by the action potential. Looking over the range of voltages used in the action potential $e^{\frac{V+N_2}{N_3}} < 0.05 \approx 0$. So, α_n can be approximated by

$$\alpha_n \approx N_1 \frac{V + N_2}{-1} = -N_1 V - N_1 N_2$$

The signs of N_1 and N_2 were flipped for ease of programmatic implementation and the final equation used is

$$\alpha_n(V_m) = \begin{cases} N_1 V_m - N_1 N_2 & , V_m \geq N_2 \\ 0 & , V_m < N_2 \end{cases} \tag{27}$$

This change is corroborated by a non-uniqueness issue between N_1 and N_2 which showed a strong linear correlation to each other, as well as a large coefficient of variance of N_3 over many optimizations of the model. This evidence agrees with the change since N_3 has been removed from the parameter set. Testing showed that α_m is also able to handle this adjustment, so M_3 was removed from the parameter set with the new equation

$$\alpha_m(V_m) = \begin{cases} M_1 V_m - M_1 M_2 & , V_m \geq M_2 \\ 0 & , V_m < M_2 \end{cases} \tag{28}$$

With the updates to the equations made an initial set of parameters was found.

The new set of equations is given as:

$$\frac{d\vec{p}}{dt} = \begin{cases} \dfrac{dn(\vec{p})}{dt} = \begin{cases} (N_1 V_m - N_1 N_2)(1-n) - \left(e^{\frac{V+N_7}{N_6}}\right) n & , V_m \geq N_2 \\ -n e^{\frac{V+N_7}{N_6}} & , V_m < N_2 \end{cases} & \text{(29a)} \\ \dfrac{dm(\vec{p})}{dt} = \begin{cases} (M_1 V_m - M_1 M_2)(1-m) - \left(e^{\frac{V+M_7}{M_6}}\right) m & , V_m \geq M_2 \\ -m e^{\frac{V+M_7}{M_6}} & , V_m < M_2 \end{cases} & \text{(29b)} \\ \dfrac{dh(\vec{p})}{dt} = \left(e^{\frac{V+H_6}{H_3}}\right)(1-h) - \left(\dfrac{1}{1+e^{\frac{V+H_4}{H_5}}}\right) h & \text{(29c)} \\ \dfrac{dV(\vec{p})}{dt} = \dfrac{I_{stim} - I_m}{C_m} & \text{(29d)} \end{cases}$$

Where $\vec{p} = \{n, m, h, V\}$ as before.

3.1.2 Objective Function

An objective function was developed to ‘‘score” the quality of the fit that the model produced in comparison to the experimental trace. A model estimate that is closer to the experimental trace was scored better. In order to score the model fit compared to the experimental trace, a difference of squares method was used. For each time point, the difference between the experimental and the model output voltage was calculated and subsequently squared. In this way, the model is scored purely on its vertical distance from the trace. This means that during the depolarization or repolarization phases, a small deviation from a good fit would lead to a much worse score for that fit.

To further compound this penalty (since the desired feature of the model is to model the action potential itself), the section of the action potential that begins immediately after the stimulus and lasts for $5ms$ is given a 5x score multiplier. This ensures that the important portions of the action potential are fit most accurately.

Additionally, the $5ms$ before the stimulus and the $5ms$ after the de-/repolarization phase of the action potential were weighted 1x since it is also important that the model accurately reflects the behavior of a resting cell. Otherwise, the model may have a good fit with the model but will continue to erroneously generate action potentials in the absence of stimulation (a condition known as hyperexcitability); if the model predicted that the resting membrane potential was a sufficient voltage to initiate the depolarization phase of the action potential, then multiple depolarizations may have been predicted where only one was expected. This result is not representative of the conditions under which the experimental traces were gathered so that behavior should be penalized.

In the process of fitting the model to experimental data, there were a handful of issues that arose. The first issue is that not all of the experimental traces had the same resting membrane potential. After that, there was difficulty in ensuring that the stimulation of the action potential matched that of the experimental trace data. Finally, it was unclear what were the proper values to use for the emfs since there is scant data that describe those values for *Thamnophis*. The conductances were initially assumed to be the same as those reported by Hodgkin and Huxley, but were later used to help fit the action potentials once a reasonable fit was achieved by only varying N_i, M_i, H_i. A handful of other details needed to be ironed out to help achieve good fitting, and those are addressed in Appendix A.

With those details addressed, attempting to fit the model to action potential data the model has a convergence problem. That is to say that even if there exists a set of parameters that uniquely fit the action potential data, the model will have a hard time reaching that optimum due to the complexity of the model developed. In the next section, finding a set of parameters that has a better fit to the general shape of curves produced by *T. atratus* and the convergence problems that the model has will

be addressed.

3.1.3 Multistart Optimization

In order to fit experimental traces, a computer model was programmed with the equations from the entire equation system (Appendix I.2.1) As an original guess for the known constants, the numbers from the original Hodgkin-Huxley paper were used (Hodgkin and Huxley, 1952a). From there, the routine in `R` known as `optim` was used to fit the parameters of the aforementioned equations system to the experimental data using the Nelder-Mead method (Appendix I.2.2) which is a simplex-based method. The computer code takes in the initial guesses for the parameters and computes the objective function on the given trace using the methodology described above. To produce a parameter set that was in the range expected from *T. atratus*, the average of the TTX-sensitive traces was created, and the optimization process was run on that curve first. The parameters for conductances were held constant at this point in the optimization process since they are the variables for which information about the efficacy of the ion channels in the snake populations is drawn.

It was noted that the model did not initially converge well to the experimental data. This is likely because the parameters of the Hodgkin-Huxley system were derived from experiments with squid giant axons rather than with muscle tissue. Nevertheless, the underlying idea persists that the activities of sodium and potassium ion movement can be representative of the total action potential phenotype. To produce curves that were closer to *T. atratus*, numerous initial guesses were made using Sobol sequences (Appendix D) centered around the Hodgkin-Huxley parameters until `optim` was able to find an adequate fit. After repeating the process of making a number of guesses, optimizing, and picking the best-fit parameter set, the model began to produce action potentials that looked more like *T. atratus*. Table 2 gives the parameter set that was

Parameter	Value
N_1	0.04
N_2	-119
N_6	-23
N_7	64.5
M_1	0.523
M_2	-69.7
M_6	-17.6
M_7	30.8
H_3	-21.1
H_4	68.3
H_5	-3.67
H_6	161.5
$\overline{G_K}$	36
$\overline{G_{Na}}$	120
$\overline{G_{Leak}}$	0.3
$\mathcal{E}_K$	-83.6
$\mathcal{E}_{Na}$	69.3
$\mathcal{E}_{Leak}$	-77.0

Table 2: Values for parameters of the TTX-sensitive fit. These are variations of parameters found in Table 1

found to approximate the average TTX-sensitive data and was used as a reference point for initiating optimizations.

With the new equations and starting point identified, the action potential plots change (Figures 14, 15, 16, 17, and 20).

Now, with a set of parameters that produce action potentials that look more like *T. atratus*, the same methodology can be applied to individual traces. Since the individual traces are from the same species, the action potentials are already qualitatively similar, so the convergence to an optimum is much quicker and the fits are reasonably good. However, when the model is asked to find an optimum from slightly different starting parameters, the model can find an optimum for each set of starting parameters that both have similar objective function scores. This means that an arbitrary optimum that is produced by the model might not be the only optimum for a given trace.

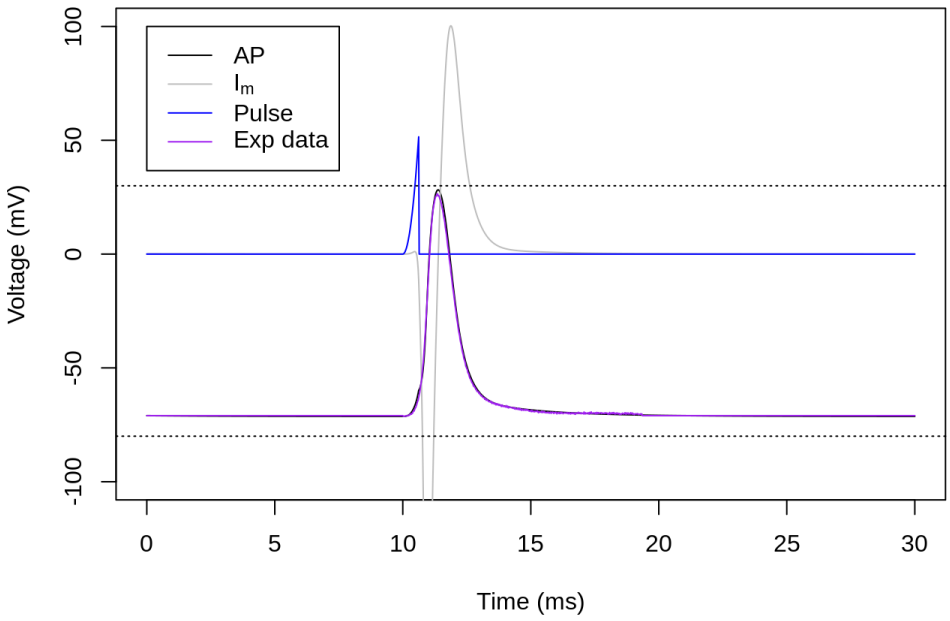

Figure 14: Example of fitting with the average of the TTX-sensitive experimental traces. Colors are the same as in Figure 2.

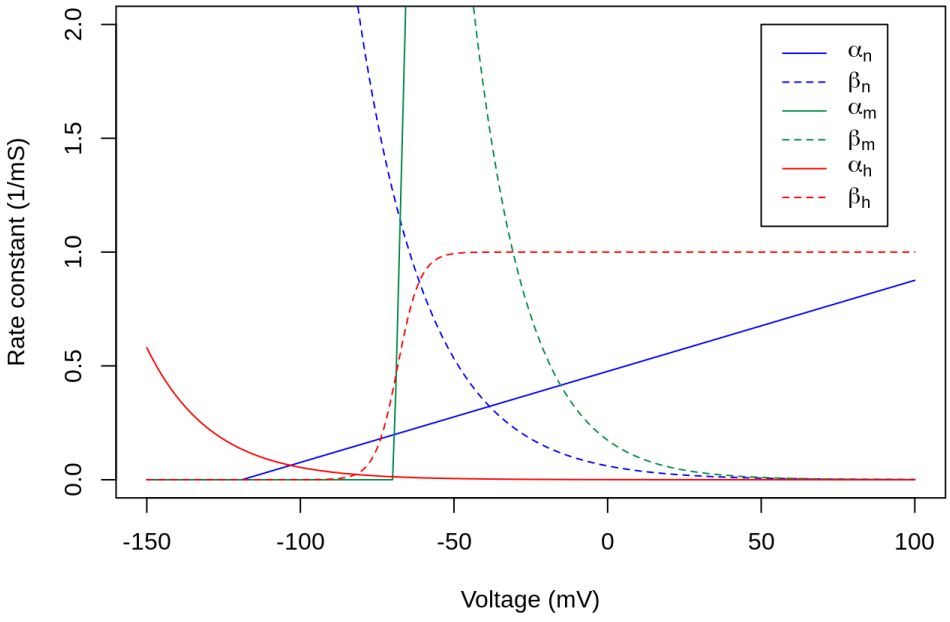

Figure 15: Plots of the α and β curves for the TTX-sensitive average using the TTX-sensitive parameter values.

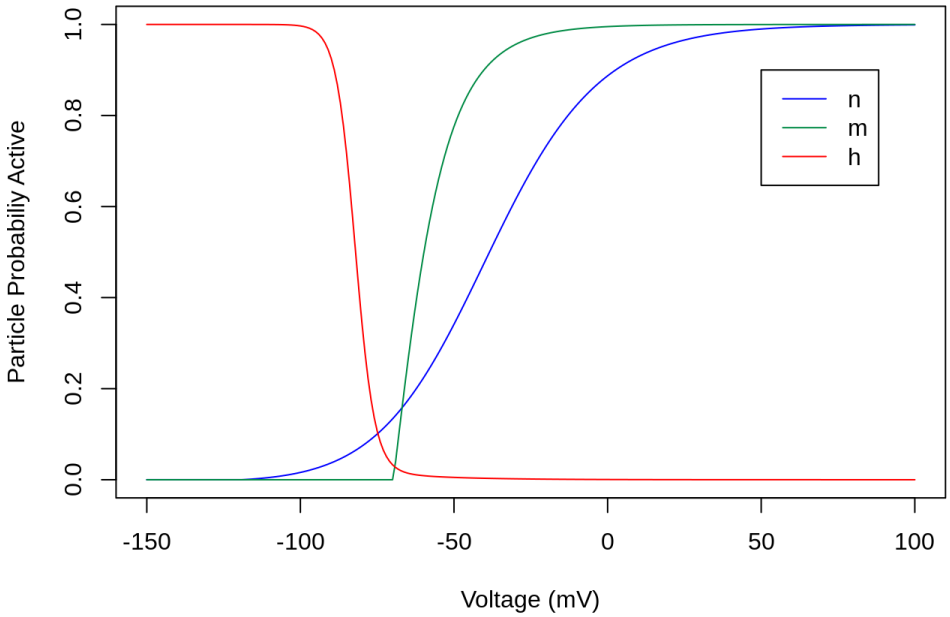

Figure 16: Plots of steady-state equilibria for n, m, and h particles over a range of voltages using the TTX-sensitive parameter values.

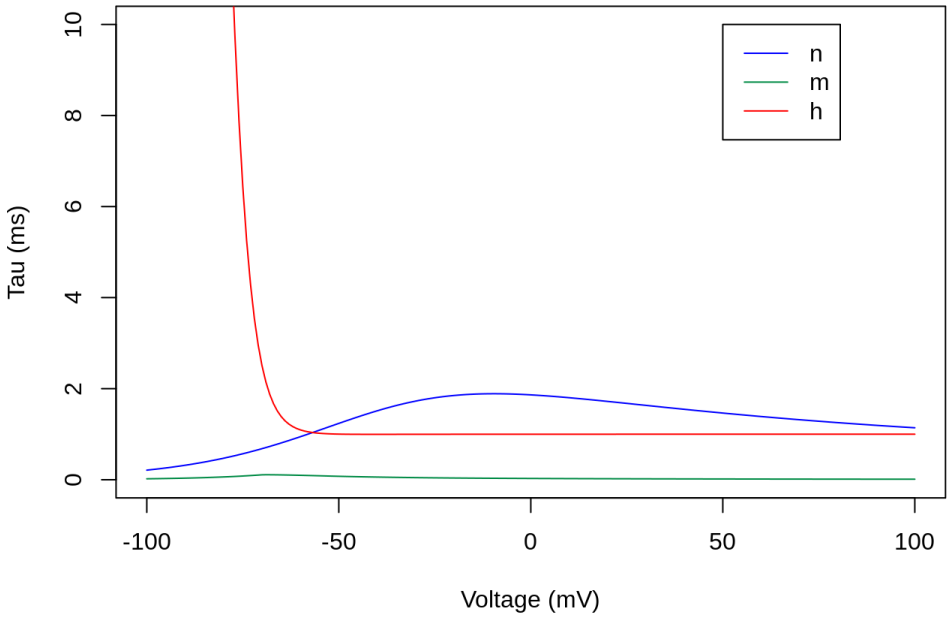

Figure 17: Plots of rate constants for n, m, and h particles over a range of voltages using the TTX-sensitive parameter values.

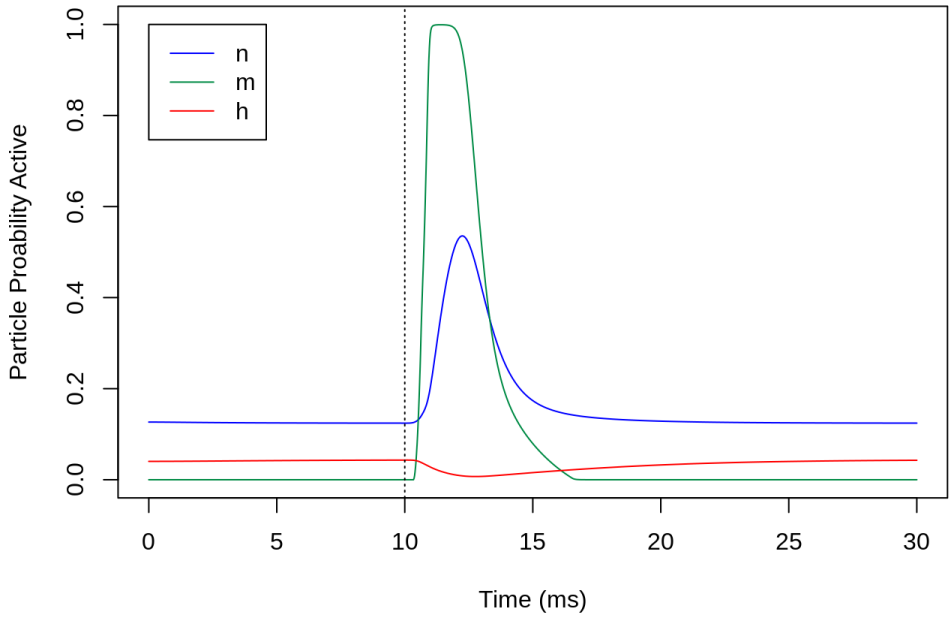

Figure 18: Plots of the conductance for sodium potassium and leak channels using the TTX-sensitive parameter values. The stimulus occurs at $10ms$.

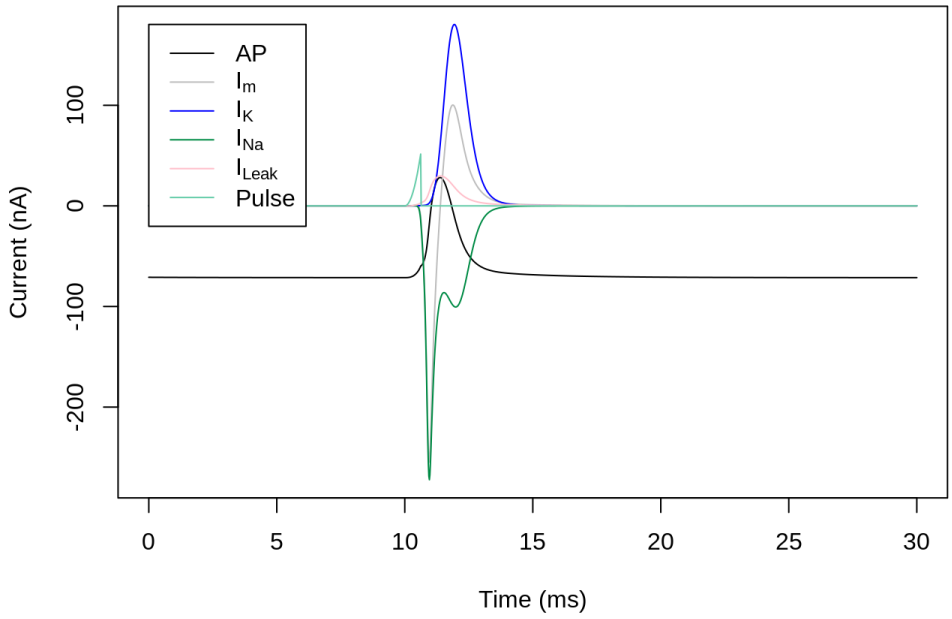

Figure 19: Plots of the currents for sodium potassium and leak channels using the TTX-sensitive parameter values. The stimulus occurs at $10ms$.

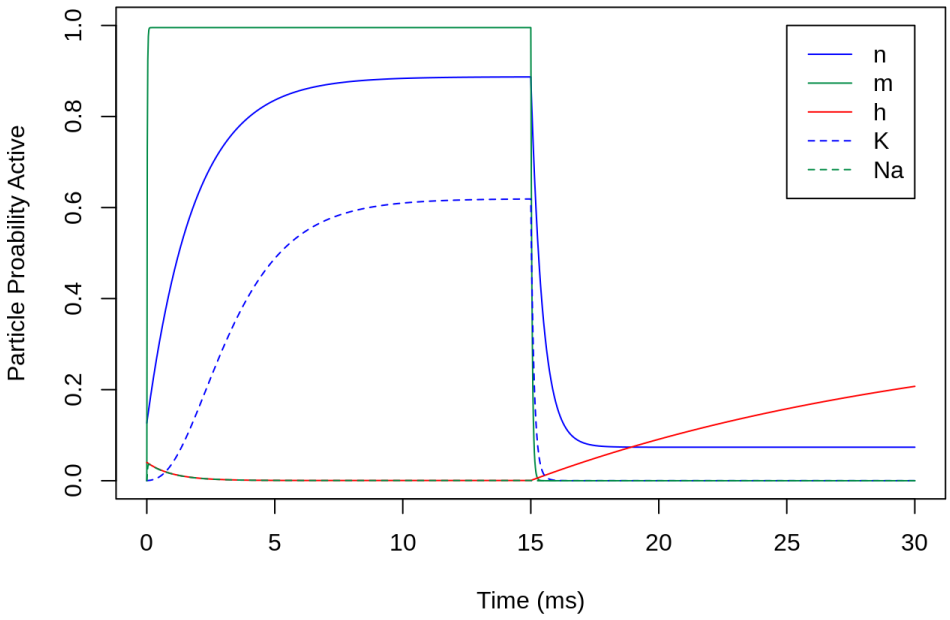

Figure 20: Plots of the subunit activation percentages or n, m, and h particles. As before, at $t = 0ms$, the voltage is switched from $-80mV$ to $0ms$, and the reverse change is made at $t = 15ms$.

In order to test this, a multi-start method was used. In practice, this was achieved by generating many parameter sets was produces via Sobol sequences (Appendix D), and each set of the parameters was optimized. Here, all of the parameters are used to produce the set of Sobol sequences, and all of the parameters are optimized. When optimizing over all of the parameters, it appears that there will be combinations of parameters that produce similarly well-fit models that have markedly different parameter values (Figure 21).

This means that there is a practical identifiability problem with the data. However, it is possible to apply the multi-start method to a subset of parameters that are optimized (free parameters), and the rest of the parameters remain at their initial values (fixed parameters).

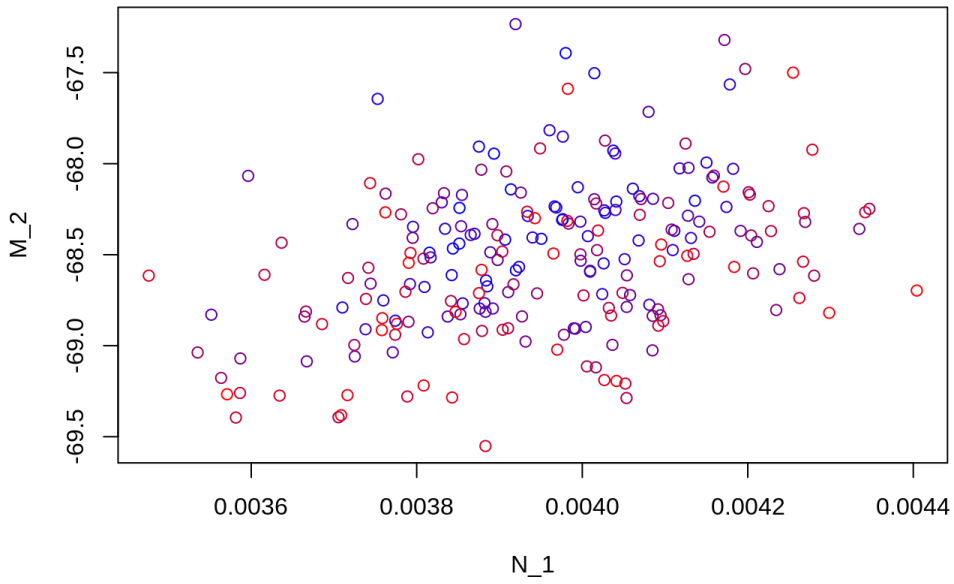

Figure 21: Example with all parameters optimized showing that there is no noticeable pattern in the data that produce better optima. Optimization that has a lower objective function score is in blue (worst in red).

3.1.4 Practical Identifiability Analysis

With the mathematical identifiability issues partially addressed, a practical identifiability analysis was conducted to determine which parameters needed to be fixed in order to uniquely determine the others via regression. This was done by starting with a core subset of high-priority parameters (G_i, then N_i, then H_i, then N_i), and using the multi-start method to estimate parameters using simulated data. The number of free parameters was increased until the appeared that uniqueness was lost.

To assess whether or not a given set of free parameters was uniquely estimable, the following computational checks were conducted. Take the simplified set of estimations shown in Figure 22. In that hypothetical example, it is assumed that the figure gives the objective function value against a range of parameter values for many (11) parameter sets that have already been optimized. So, even after optimization, the estimated parameter does not converge to a single value. However, the parameters in

the identifiable plot show that the parameters that have better agreement with the experimental data are clustered together. This means that there is a good chance that there exists an optimal parameter set that fits the model the best, but that there are convergence issues with the model. So, one can make numerous initial guesses about the true parameter set for a given trace, optimize starting at all of those parameter sets, and choose the final estimate to be the one with the best fit. This will likely be the parameter set that uniquely approximates the experimental trace (or a parameter set that is close to the true unique optimum). In the unidentifiable theoretical example, estimated parameter values achieve practically equal goodness-of-fit values. So, there is some non-identifiable parameter in the set of free parameters which will produce highly variable estimated values due to the lack of uniqueness.

When looking at Figure 22, it may be clear that there is a possible ‘‘best” fit, but an immediate consequence of the figure is that a convergence issue is certainly present, as alluded to earlier. To test whether this is, in fact, the case with the present mode, thousands of distinct parameter sets were optimized, and the best handful of parameters was selected to undergo a second round of optimization (details in Section C).

It was found that the conductance parameters on their own (G_K, G_{Na}, G_{Leak}) can uniquely return to their original value when scrambled via Sobol sequences (Figure 23 -- Notice the scale of the axes).

However, these three parameters alone are not sufficient to adequately predict the experimental action potentials. So, the M_i parameters were added, and those parameters appear to have properties of uniqueness but have difficulty returning to their original values with the given `optim` settings (Figure 24). Using only the conductances and the N_i parameters lends itself to better uniqueness, but does not do a good job of fitting experimental traces (25). If the conductance parameters are with

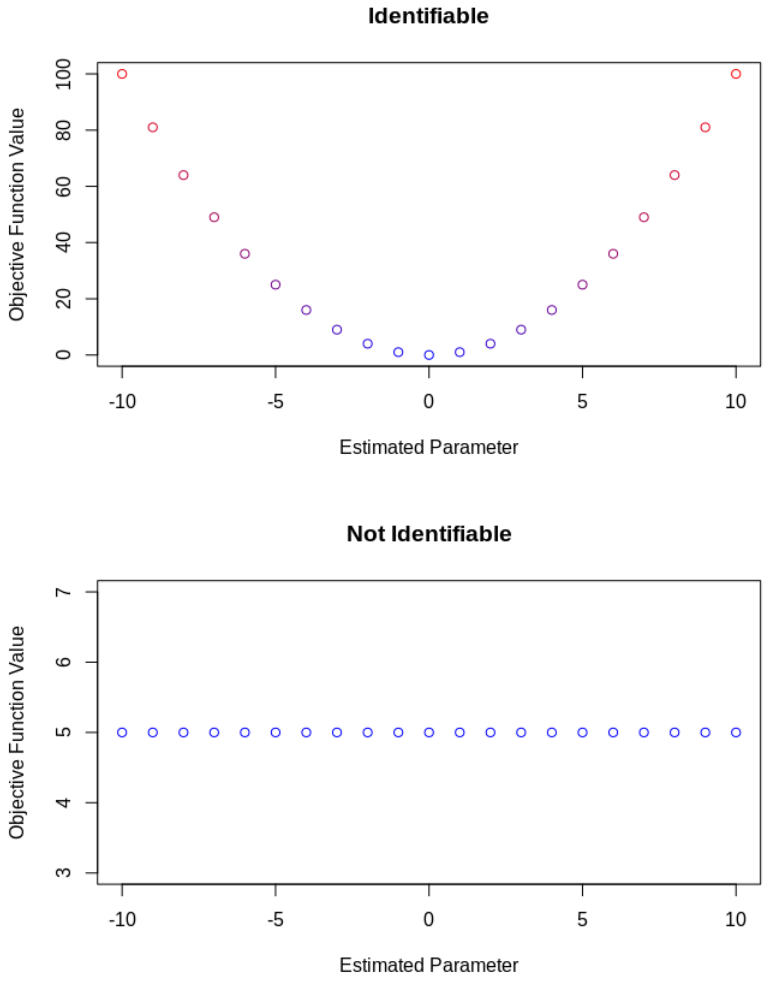

Figure 22: Plot of hypothetical optimizations where the parameter pairs that are closer to the ideal fit are color-coded blue and those further away in red.

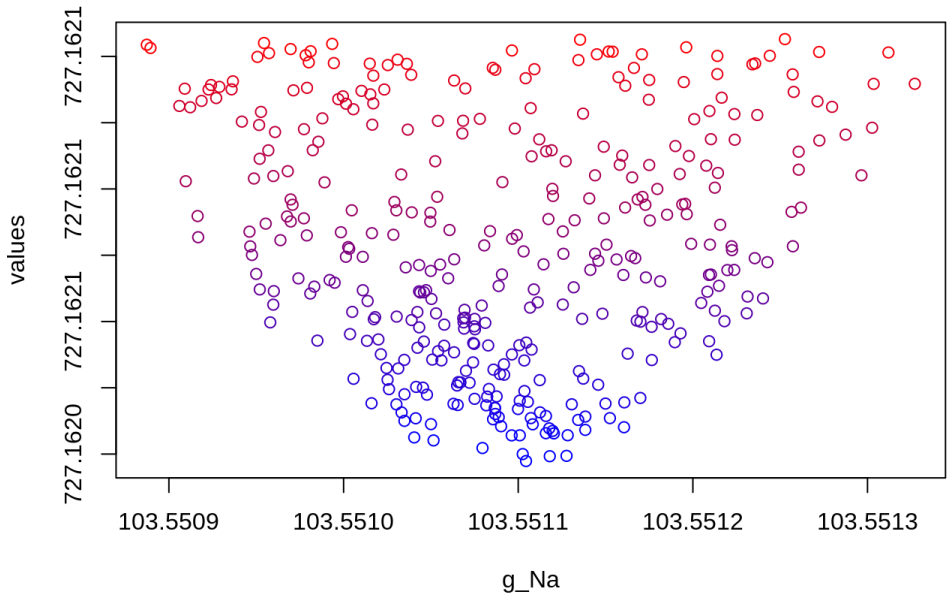

Figure 23: Scatter plot of best 20% of optimized parameters used for testing uniqueness where only G_i are optimized

the M_i and H_i parameters, the model is on the way to recovering the parameters, but a more complex optimization would be necessary to use that parameter set for the experimental calculations.

As previously mentioned it was found that the parameter set used including M_i parameters and the conductances have unique solutions, but do not reliably converge (Figure 26, 27, 24). In Figure 26, despite the outlier, it is evident that it follows the same trend as the identifiable example from Figure 22. In figures 27 and 24, it can be seen, by comparing the relationship between 2 parameters that, there may or may not be a trend relating the two parameters. However, there is a combination of parameters that arise together when the model achieves its best fit (the parameter sets that provide the best fit to the generated experimental data are clustered very closely together). This gives further evidence to the claim that (with this parameter subset) there is a unique best way to fit a trace, but that the computational procedure has difficulty converging.

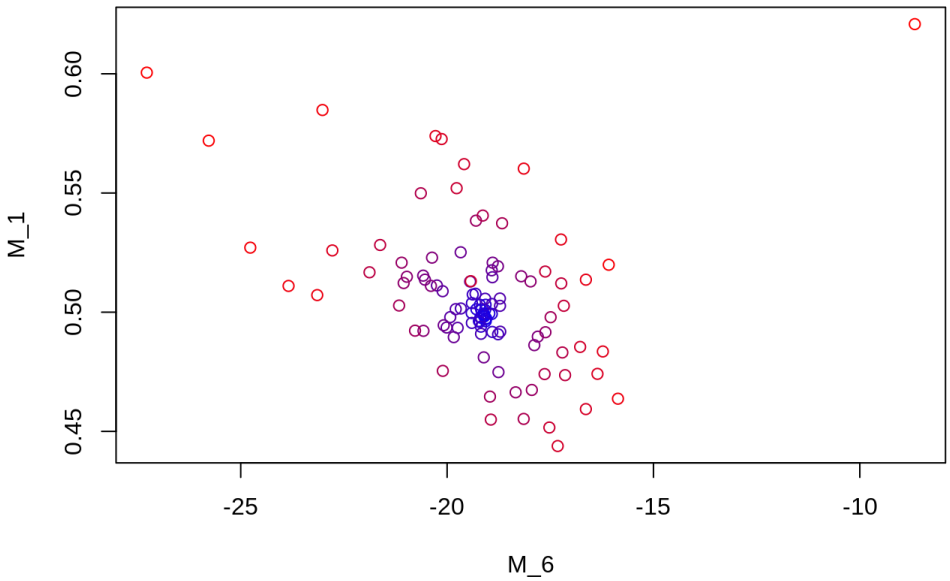

Figure 24: Scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

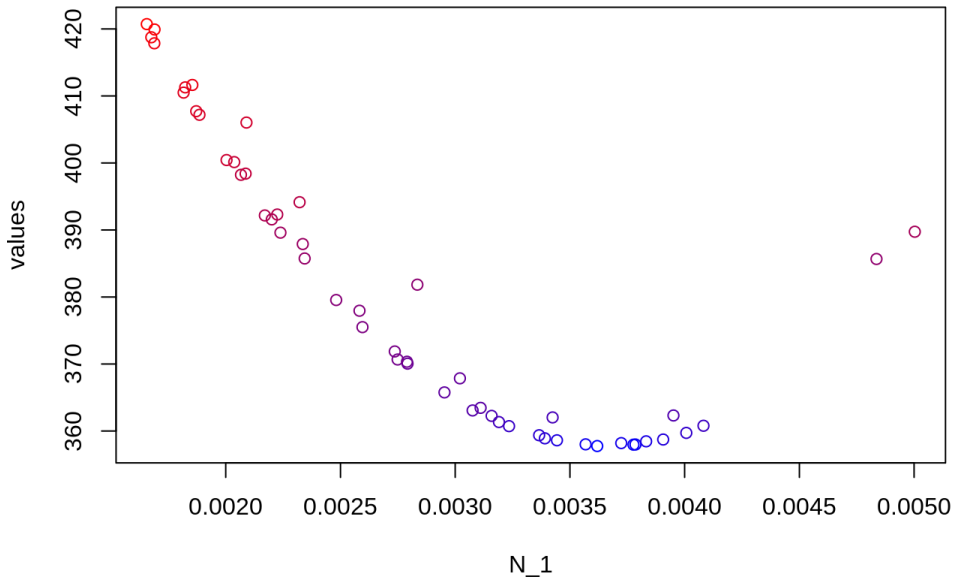

Figure 25: Scatter plot of best fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over N_i and G_i.

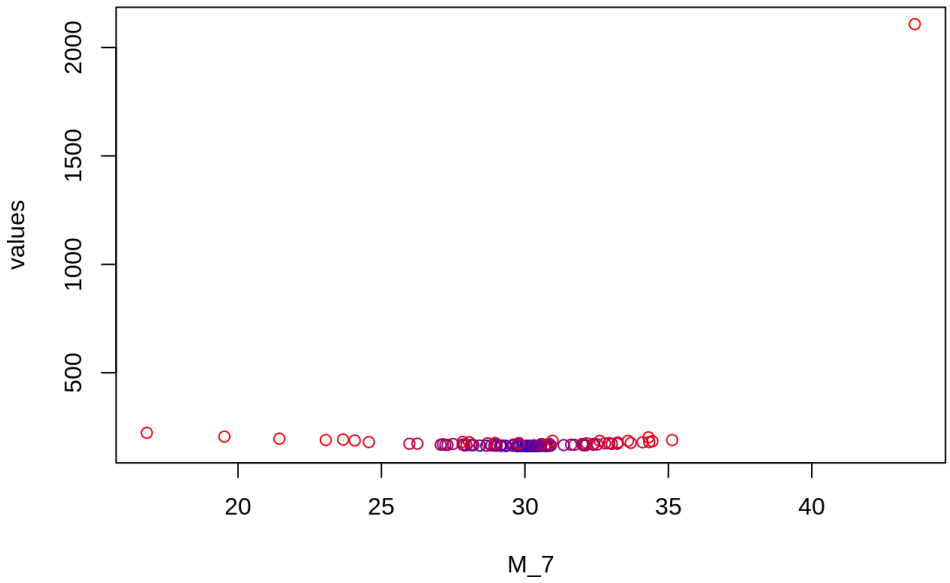

Figure 26: Scatter plot fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

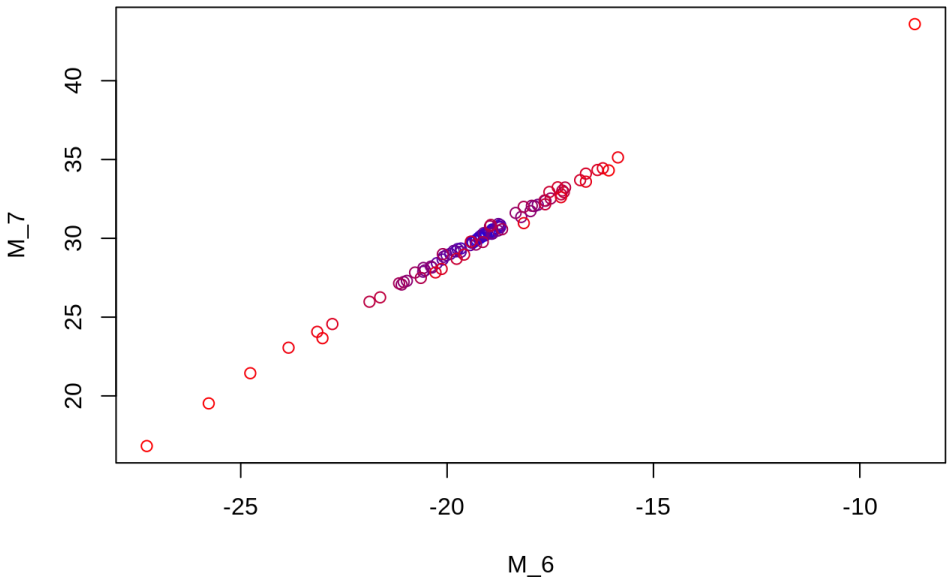

Figure 27: Scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

So, it appears that this parameter set can uniquely identify the parameters that fit an experimental action potential, but may require fine-tuning to ensure that a given action potential is fit with its unique, optimal parameter set. below, this convergence issue was mitigated by applying the multi-start optimization procedure as detailed above. Furthermore, any small bias that may be present, as a result, is likely to have little impact since the same estimation procedure is used across many replicate fixed parameter sets to then compare group-level parameter values. Thus, with this parameter estimation and comparison procedure in hand, the next step is to investigate whether or not the action potentials among the TTX-sensitive group are quantifiably different from the mutant group. To do this, a group analysis was used.

4 Group Comparison Procedure

In order to use the above model fitting procedure to compare TTX-sensitive and mutant groups, the following procedure was used in order to account for the fact that considerable uncertainty is introduced by fixing some parameter values during the parameter estimation setup. To overcome this obstacle, multiple sets of values for the fixed parameters were made, and the estimates for the remaining parameters (by performing the multi-start approach outlined earlier) were estimated for each of the fixed parameter sets (Figure 28). These replicate parameter sets can be used to assess any group-level differences in the estimated parameters. This is accomplished in two steps. First, multiple estimates are generated for each trace as described below. Second, a random-slope, random-intercept regression model is used to quantify any group-level differences in the model parameters.

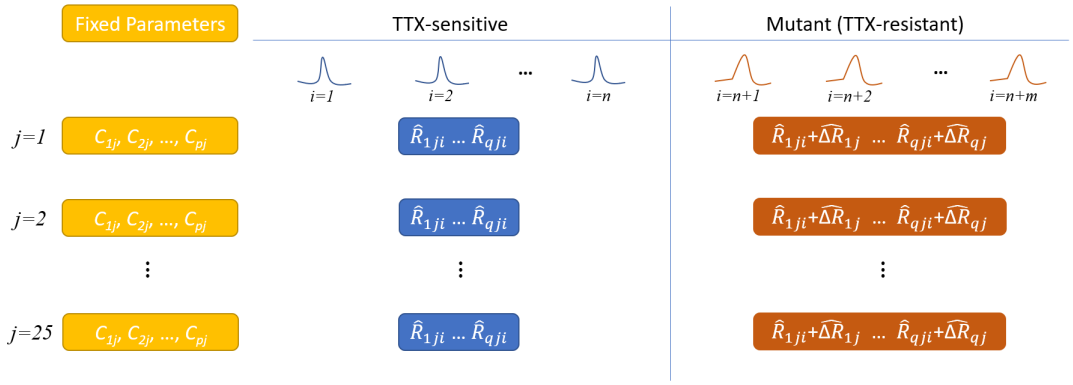

Figure 28: Gives a diagram of the group analysis procedure (for intermediate -- trace-level -- parameter estimates). Fixed parameters are in yellow represented by C_{pj}. Free parameters are blue (for TTX-sensitive) and orange (P mutant) represented by $\hat{R}_{kji}$. There are $J = 25$ sets of constant parameters (uniform over plausible range). With the q remaining free parameters, estimate R_{kji} for each trace, J times. Here $k = 1 \ldots q$ indexes the parameters, $i = traceID$, $j = 1 \ldots J$.

4.1 Replicate Trace-Level Parameter Estimates

In order to determine the efficacy and uniqueness of the model that is being used, one can generate quasi-experimental data and then attempt to model the simulated data with the technique in question. The results can then be compared to the known inputs that generated the simulated data to determine whether or not the results of the model are reliable. For the model being used here, start with the parameters that were optimized against the TTX-sensitive average data.

An assumption can be made -- with relative confidence in its accuracy -- that the parameters involved with the sodium channel are the only parameters that will be markedly different between the different snake genotypes since this has been the case with other animals (Jan and Jan, 2012). This is because the DNA sequences between the snake groups differ in the regions responsible for the manufacture of the sodium channel (Feldman et al., 2010) but do not differ in those regions responsible for the manufacture of potassium channels or the inactivation protein of the sodium channel. Additionally, the use of simulated data has a two-fold benefit. Firstly, the data does not have experimental noise, and secondly, the group from which the generated data originated is known. So, when the optimization runs, the expected outcome of the optimization is known.

Armed with this information and the results from the identifiability analysis, the fixed parameter set was chosen to be N_i and H_i, and the free parameter set was chosen to be G_i and M_i. Using the fixed parameters, Sobol sequences were used to generate 25 sets of fixed parameters. The free parameters were optimized against the generated data (follow Figure 28 for clarification). Details about the specifics of the algorithm can be found in Appendix E. Once the data from the trace-level parameter estimates are found (with their 25-fold replication), a group-level comparison can be made.

4.2 Linear Mixed Effects Analysis (Group-Level Comparison)

The linear effects analysis is used to tease out the variability in the entirety of the dataset versus the variability that arises from true differences between the groups that are being compared. In this case, the random effects come from random variability in the experimental conditions (which is subsequently captured in the intentional differences introduced by the quasi-experimental data generation). The fixed effects come from the underlying differences in the action potentials of the two groups. With the notation as in Figure 28, assume the estimates in each group, for each individual, arise from a TTX-sensitive group-level parameter value $\mathbf{R}_k$ -- or a mutant group-level parameter written as $\mathbf{R}_k + \Delta\mathbf{R}_k$ -- with some iid (Gaussian) errors,

$$(\eta_{kj}, \xi_{kj}) \sim \mathcal{N}(0, \mathbf{\Sigma}_k) \tag{30}$$

that capture variation in slope ($\Delta\mathbf{R}_k$) and intercept ($\mathbf{R}_k$) estimates across each set of fixed parameter values (i.e., for each j), and some additional Gaussian errors,

$$\varepsilon_{kji} \sim \mathcal{N}(0, s_k) \tag{31}$$

that capture additional variation in estimates of R_{kji} at the trace level. This means that $\mathbf{R}_k$ and $\Delta\mathbf{R}_k$ can be estimated from the R_{kji} data via a mixed-effect model,

$$R_{kji} = (\mathbf{R}_k + \eta_{kj}) + (\Delta\mathbf{R}_k + \xi_{kj}) \times \mathbf{X}_i + \varepsilon_{kji} \tag{32}$$

where $\mathbf{X}_i$ is either 0 (TTX-sensitive) or 1 (mutant).

Figure 29 gives an example output of the `R` code (Appendix J.2) once the aforementioned analysis is performed.

Below is a table of expectations for the parameter values under the assumption

```
## Linear mixed-effects model fit by REML
##  Data: sim_data
##        AIC      BIC    logLik
##   3419.582 3440.663 -1703.791
##
## Random effects:
##  Formula: ~1 + group | tbl
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev   Corr
## (Intercept) 379.3987 (Intr)
## group        84.6009 -0.98
## Residual    198.2742
##
## Fixed effects: reformulate("group", param)
##                Value Std.Error  DF   t-value p-value
## (Intercept) 343.3251  85.61683 224  4.010019  0.0001
## group       -66.9633  30.25384 224 -2.213383  0.0279
##  Correlation:
##       (Intr)
## group -0.85
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.57939422 -0.26210612 -0.07900183  0.08409977  5.93676534
##
## Number of Observations: 250
## Number of Groups: 25
```

Figure 29: Output of the LME code in `R` using the procedure as outlined. The given group-level comparison is for the G_{Na} parameter.

Parameter	Description	P - TTX-sensitive Change
M_1	increases α with voltage increase	=
M_2	sets α activation voltage	=
M_6	inversely regulates β	=
M_7	directly regulates β	=
$\overline{G_K}$	maximal K^+ conductance	=
$\overline{G_{Na}}$	maximal Na^+ conductance	-
$\overline{G_{Leak}}$	leak conductance	=

Table 3: Expectation of the change in the value of the optimized parameters from TTX-sensitive to P groups. This expectation is given based on patch-clamp data on rat sodium channels which did not demonstrate changes in what is being described here as α.

that the P mutant *T. atratus* have sodium channels that are worse at conducting sodium ions than the TTX-sensitive group. Table 3 lists the coarse changes that are expected from the TTX-sensitive to the P group based on visual inspection of the curves (Appendix H) and intuition derived from previous patch-clamp data (del Carlo et al., 2018).

5 Data Analysis Application

5.1 Group Fitting

Table 4 shows the TTX-sensitive group average for a given parameter and its difference in the second group along with the p-value associated with that difference.

Parameter	TTX-sensitive	Δ TTX-sensitive	p-value
M_1	0.61 ($\pm$0.18)	0.03 ($\pm$0.10)	0.7932
M_2	-121.59 ($\pm$11.18)	23.84 ($\pm$6.69)	0.0004
M_6	-16.4 ($\pm$3.15)	-4.19 ($\pm$1.80)	0.0205
M_7	18.7 ($\pm$3.64)	0.39 ($\pm$2.00)	0.8448
$\overline{G_K}$	65.27 ($\pm$11.21)	-4.94 ($\pm$4.21)	0.2419
$\overline{G_{Na}}$	343.32 ($\pm$85.62)	-66.96 ($\pm$30.25)	0.0279
$\overline{G_{Leak}}$	0.84 ($\pm$0.13)	-0.12 ($\pm$0.08)	0.1265

Table 4: Changes in the value of the optimized group-level parameters from the estimate for TTX-sensitive to P groups. The data points from `LME` are given with standard error as well as their respective p-value.

Parameter	Hypothesis	Observed
M_1	=	=
M_2	=	+
M_6	=	-
M_7	=	=
$\overline{G_K}$	=	=
$\overline{G_{Na}}$	-	-
$\overline{G_{Leak}}$	=	=

Table 5: Hypothesized change from estimated TTX-sensitive parameter versus Actual Parameter Changes.

Table 5 shows the actual change against the hypothesized change. If the p-value was greater than 0.05, the observed change was considered to be zero since there was no evidence to claim that the calculated change was present.

6 Discussion

The primary result of interest is the conductance of sodium. The current hypothesis regarding evolutionary trade-offs indicates that if the sodium channel of TTX-resistant snakes is altered to affect the binding capacity of TTX, then it may also compromise the ability of the channel to function, thus decreasing G_{Na}. This is indeed the result that was found, providing new evidence that supports the claim that these two genotypes of *T. atratus* produce qualitatively different action potentials. Moreover, the mechanistic underpinnings of the model indicate that the observed difference arises from a decreased ability of the sodium channel to ferry sodium ions. Furthermore, the assumption that the potassium channel does not act as a compensatory mechanism in the evolution of TTX resistance is supported because there was no statistical evidence to conclude that G_K was significantly different between the two groups.

6.1 Biological Implications

An important outcome of this work is the possible discovery of a kinetic component of the trade-off between toxin resistance and channel function. There are two ways that the total sodium current through the membrane can be reduced. The first explanation is that, even thought the sodium channel is open, less sodium is moving through (each channel). The second explanation is that the flow of sodium is not lessened, but rather there are fewer sodium channels open at a given voltage. The latter explanation is (about the tendency of sodium channels to open or close at a given voltage) is referred to as ‘‘channel kinetics”. Up until now, all investigators have found virtually identical activation and inactivation kinetics between TTX-sensitive and TTX-resistant snakes (Lee et al., 2011; Hague et al., 2018; del Carlo, 2020), as well as unpublished research by Robert del Carlo, Ph.D.). Specifically, no mutations to the pore-forming domain that confer any amount of TTX-resistance have been associated with significant changes

in sodium channel activation or inactivation. There is a molecular explanation that supports these findings. The central pore-forming domains where TTX-resistance conferring mutations occur are physically distant from the peripheral voltage-sensing domains that influence activation and inactivation (Yu and Catterall, 2003). So it is logical that these regions may be physically independent.

However, there are many caveats that leave room for future investigations. First, the sodium channel is formed by a single peptide chain, meaning that no two regions anywhere along that chain can be wholly considered "independent." Next, Lee et al. (2011). used a combination of human and rat sodium channel backbones to search for possible deleterious effects of resistance-conferring mutations. That study used the human channel as the WT control whereas snake resistance mutations were inserted into a rat channel. While the voltage-sensing domains and the pore-forming domains are conserved in sequence across phyla, mild variation exists throughout the rest of the protein. Any difference in the structural background of the channel between humans and rats that modulate the effect of a mutation would be considered a "background-dependent effect." For instance, if sufficient structural differences exist between the human and rat channels, such that the molecular dynamics of the region between the PFD and VSDs are altered, perhaps in stiffness or flexibility, then the resistance-conferring mutations in the PFD may have a variable relationship to the VSDs on different channel backgrounds. That is, the genetic background might dictate how much influence mutations in one region of a protein have on other, farther regions.

For this reason, it would be most prudent to carry out experiments on snake channel mutations on a snake channel background. There are many expedient reasons that a group might forego generating a snake channel clone and its subsequent mutants, from cost to experimental feasibility. Hague et al. (2018) avoided this conundrum

by working on snake clones but opened another inconsistency by expressing those snake channels in *Xenopus* oocytes. Lee et al. (2011) also expressed their mammalian channels in amphibious *Xenopus* oocytes. While there are no known snake cell expression systems on which to carry out this kind of electrophysiology, working in Xenopus does reduce the biological relevance of the findings (though, to a lesser degree than the **rat** channels with **snake** mutations expressed in **human** cells by del Carlo (2020)). None of the above systems, no matter what channel background (human, rat, or snake) or in which cellular expression system (human or *Xenopus*), were able to provide any indication that channel kinetics governed by the VSDs are meaningfully affected by mutations to the PFDs.

Despite this consensus by research in heterologous expression systems, nothing beats measuring a protein in its native environment. None of the above experiments were conducted on native snake skeletal muscle expressing naturally-selected snake sodium channels alongside the entirety of their protein partners (β subunits and mechanical proteins), with all of their naturally occurring glycosylations. There is a very good reason for this: the technique needed to achieve a voltage clamp on such a complicated tissue (two-electrode voltage clamp, TEV) is extremely difficult, expensive, and rare. The beauty of our current preparation, using a simple intracellular recording from a sharp electrode impalement into native skeletal muscle, followed by a mathematical treatment to decompose the action potential, is a much cheaper and expedient procedure to estimate the channel conductance and kinetics. And from this procedure, we observed two novel findings that do indeed suggest kinetics may be at play in native tissues.

Whereas heterologous expression electrophysiology suggests that reduced sodium current is due to reduced channel conductance without any significant kinetic component, this model suggests that single-point mutant channels may favor remaining

in the inactive state, thereby reducing total sodium current through the membrane. Parameters from the model, M_2 and M_6, both deviated from expectations (Table 5) that they would be unchanged between sensitive and resistant snakes. M_2 is a component in the $\alpha_m(V)$ equation and sets the threshold for activation for the sodium channel and was demonstrated to be elevated in the P mutant group. This suggests that mutant P snake channels require greater depolarization to activate than their TTX-sensitive counterparts. Next, M_6, which is used in $\beta_m(V)$, inversely affects the rate at which channels transition from permeant to impermeable states. Parameter M_6 was found to be reduced in the P mutant group. Consequently, $\beta_m(V)$ would be increased, favoring channels transitioning to or remaining in the inactive state. Both changes, increased M_2 and decreased M_6, favor a greater proportion of the channel population remaining closed. This would compound the fallout of the reduced G_{Na} that is observed in the current model as well by del Carlo (2020). suggesting that unitary conductance is reduced by resistance-conferring mutations.

A molecular explanation of these kinetic findings might come down to the substitution itself. The mutation in the P snakes, A1281P, takes a relatively small, flexible α-amino acid like alanine whose residue sidechain is simply a methyl group and replaces it with a conformationally rigid, secondary amine like Proline that is known to disrupt protein secondary structure (Morris et al., 1992). It may be that such a mutation not only changes the topological face of the pore, reducing the binding affinity of TTX but also introduces a stiffness into the PFD that could apply sufficient tension onto the VSD so as to interfere with the voltage-sensing function. It would be very interesting to examine the triple-point mutant, EPN, to compare G_{Na}, M_2, and M_6. While we expect G_{Na} to be even further reduced in EPN, it is possible that such a reduction is conferred by D1568N which is also observed in highly resistant populations of a sister taxon, *Thamnophis sirtalis* (Hague et al., 2018; del Carlo, 2020).

That leaves some room for D1277E to act in a compensatory manner, repairing some of the kinetic damage incurred to M_2 and M_6. At the least, these findings from sharp electrode recordings in native snake skeletal muscle warrant further corroboration with gold-standard TEV electrophysiology to both confirm the findings and validate this mathematical methodology.

6.2 Further Research

For future research, there are a number of other species of *Thamnophis* as well as another class of mutant *T. atratus* (EPN) that can be investigated to elaborate on these findings. It is expected that snake populations with a greater number of mutations (that have been shown to have better resistance to TTX) will have a decreased conductance of sodium. This would corroborate the hypothesis that there is an evolutionary trade-off in acquiring more modifications to the sodium ion channel. Additionally, it may be possible to isolate the individual effects of the point mutation across different species since the mutations that confer TTX resistance are not the same across species (but there does appear to be a mutation that is able to be made first -- P). It may also be the case that is not statistically possible to obtain the best collection of single-point mutations in a single generation that would confer a large resistance to TTX as well as maintain peak functionality of the sodium ion channel. If this were true the snake would have to traverse an evolutionary trajectory that only makes a single, viable change at a time and, up to this point, has worsened the efficacy of the ion channel.

Furthermore, with respect to the analysis itself, the parameter subset that was used in the experimental calculations could be broadened. Up to this point, the efforts that have been employed to search for unique parameter sets have been extensive, but not exhaustive. With a more complex estimation technique where some action potentials

are generated as a baseline using Sobol sequences as seed values, these parameters would be optimized and could then be further used to create a multidimensional normal distribution of parameters for which new parameter sets could be sampled from the inner quartiles of the distribution to reduce noise in the system. This gives the opportunity to reduce the effects of convergence issues on the model, but a procedure like this would have to be verified against the experimental traces to ensure that it is a reliable fitting technique. If a new refined procedure for fitting experimental traces provided adequate results (when applied to different groups of *Thamnophis*) the aforementioned analysis can be performed.

With all of the details ironed out. The fallout of the results, if they occur as expected, would provide great insight into the nature of the natural selection landscape. It would give evidence for the predictability of evolution because it would demonstrate the need to achieve specific evolutionary milestones in a specific order to become better adapted to an environment. The implication of this finding would be that evolution as a whole, can be (at least in part) predicted. Knowing that provides an opportunity to develop tools to measure and even directly shape the course of evolution.

7 Appendix

A Computational Minutia for the Model

A.1 Resting Membrane Potential Estimation

Since the location of the action potential peak is known by the virtue of the data cleaning method ($2ms$ after the start of the experimental data), the minimum of the time series before $2ms$ was taken to be the minimum of the resting membrane potential. From there, a short section of data ($200\mu s$) after the repolarization phase ($5ms$ after the start of the experimental data) was sampled and the average of the maximum and minimum of that section was calculated. This estimates noise in the experimental data for that trace. This noise value was added to the minimum-before-depolarization value to approximate the RMP. This value is chosen since the minimum value before the depolarization phase (in the experimental setup) is the voltage maintained by the membrane in the absence of a stimulus which is precisely what is expected of the RMP. Then the average noise value is added to account for the fact that the minimum data point would have been influenced by sampling noise.

A.2 Action Potential Stimulus

One of the key features in the success of the model depends on the model's ability to accurately model the initial stimulus of the action potential. The feature in the action potential curve that is generated by the initial stimulus is referred to as the foot of the action potential. The foot of an action potential is the initial part of the depolarization phase -- where the voltage across the cell membrane begins to increase. Note that an increase in voltage at this instance is meant to convey the tendency for the voltage across the cellular membrane to return to 0 (resting membrane potential

begins as a negative voltage). It is assumed that the shape of the foot of the action potential from experimental data is developed for the passive propagation of ions proximal to the site of recording. This assumption is modeled by treating the stimulus as that passive ion flow.

When fitting the foot of the experimental trace data, the following methodology is used. The assumption is that each experiential action potential was exposed to the same amount of propagating current. Under this assumption, the integral of the stimulus is fixed. This is because the integral of the stimulus gives a decent approximation of the total amount of charge that is experienced by the stimulus (the total passive flow of sodium ions in the vicinity of the probing electrodes). Maintaining a constant value for the total amount of charge supplied by the stimulus regularizes the model on the assumption that the traces were gathered under the same experimental conditions. With this setup, during the foot fitting process, the model foot is optimized to the trace of the experimental data while maintaining an equivalent integration of the stimulus by varying the height of the stimulus as a function of the time that the stimulus is felt. That is to say that action potentials with a more shallow depolarization phase will have stimuli with lower height over a longer duration.

A.2.1 Explanation for mathematical derivation

First, a function $f(x)$ needs to have the properties $f(0) = 0$ and $f(d) = h$ where d is the duration of the stimulus and h is the height of the stimulus. Secondly, the function should have a property such that the integral, $F(0) = 0$. That is to say that, with A as the stimulus area,

$$A = \int_0^d f(x)dx = F(d) - F(0) = F(d) \tag{33}$$

Functions with this property are (not exhaustively) exponential and monomial functions. So, for some monomial $p(x)$, since $p(1) = 1$, $hp(\frac{x}{d}) = h$ when $x = d$. Thus, $f(x) = hp(\frac{x}{d})$ satisfies the properties listed. Now, choose $p(x) = x^m$. Then, that gives the equation

$$f(x) = h \cdot \left(\frac{x}{d}\right)^m \tag{34}$$

Then, to find the area (the total charge experienced at the site of probing) of the stimulus curve,

$$A = \int_0^d f(x) = \int_0^d h \left(\frac{x}{d}\right)^m dx = \frac{hx^{m+1}}{(m+1)d^m}\Bigg|_0^d = \frac{hd}{m+1} \tag{35}$$

So, $h = \frac{(m+1)A}{d}$. Thus, an optimization can be run over the start time of the stimulus and the duration of the stimulus without recalculating the area of each function by pre-calculating the area found when fitting the TTX-sensitive average data and using the formula for h.

This means that for each novel action potential trace, the parameters for m and d are optimized using the same optimization algorithm as in the main optimization (Nelder-Mead with least-squares) where the area from the TTX-sensitive average is used as a benchmark for the total area to hold constant. From those two variables (and the constant area), h is calculated. With those three variables, the shape of the action potential foot can be derived. In the implementation, however, there was a third variable that controlled the time at which the foot began. This was also used in the optimization to correct for experimental differences across trials where the true ‘‘beginning” of the action potential is vague.

A.3 EMF Calculation

Electromotive potentials of K^+ and Na^+ are approximated from the Nernst equation (Equation 1) and known concentrations of the Kreb's solution used to collect the action potential data. The values of the Nernsts equation are as follows

$$R = 8.3144598 \frac{J}{mol \cdot K}$$
$$F = 96.485 \frac{J}{mol \cdot mV}$$
$$T = 296.15K \text{ (Room temperature)}$$
$$z \text{ is the ionic charge}$$

The molar concentrations in the Kreb's solution were: 145 Na^+, 127.7 Cl^-, 25 HCO_3^-, 5.5 glucose, 5.4 K^+, 1.2 Mg^{2+}, 1.8 Ca^{2+}, 1.2 $H_2PO_4^-$, 1.2 SO_4^{2-} and supplemented with 95% O2 and 5% CO_2 to achieve pH 7.35. The estimated intracellular concentrations (rodent white *gastrocnemius* tissue) (Lindinger and Heigenhauser, 1987) were 9.6 Na^+, 8.7 Cl^-, 143 K^+, 31.2 Mg^{+2}, and 3.5 Ca^{2+}.

A.4 Numerical Estimation Procedure

Recall that the experimental action potential traces that were gathered were a measure of the voltage across the cellular membrane over time. Suppose that $s \in \{n, m, h\}$ is an arbitrary sub-unit particle. Then, the general idea for the computation is as follows:

1. Start with an initial voltage (V_m). It is sensible to use the experimental resting membrane potential.
   (a) Initialize n, m, h with s_∞ equations(Section 2.1).
2. Using Euler's method, numerically integrate each $\frac{ds}{dt}$ over the specified time step.

3. Calculate I_m with the equation in Section 2.2.

4. Using Euler's method, numerically integrate $\frac{dV_m}{dt}$ over the specified time step.

5. Return to step 2.

B Structural Identifiability Example

In a mathematical model this complex, there exists the possibility of non-uniqueness (identifiability issues). Consider the linear system:

$$y = (m_1 + m_2)x + b \tag{36}$$

where m_1, m_2, and b are the parameters of the system that are yet to be determined. Suppose that the true slope of this line is m and its intercept is trivially b. This means that the true solution of $m = m_1 + m_2$. Therefore, there are infinitely many combinations of m_1 and m_2 that will satisfy this system. This is what is meant by an identifiability issue: the parameters for the system cannot be uniquely chosen to fit the system even though a perfect fit will, most likely, be found. In this case, it would be prudent to change the system to:

$$y = mx + b \tag{37}$$

where m and b are the only parameters that are being pursued. Thus, assuming the true nature of the real-world system in question is linear, then the m and b (if found) will be unique -- barring a vertical line, of course. What this all means is that some arbitrary model (Equation 36) may have more parameters than are necessary to describe the ‘‘motion” of the system. In this case, the number of parameters can be reduced from 3 to 2 (Equation 37). In the case of the Hodgkin-Huxley system, the overparameterization is less obvious (Section 3.1.1).

C Practical Identifiability Technique

With the model used in the experimental analysis, by starting with the parameter set that was found to fit the TTX-sensitive average, the trace that was generated by that model was saved and used as a quasi-experimental trace to observe and eliminate the noise in the experimental data. Then using Sobol sequences (Section D) to generate thousands of parameter sets, each sequenced parameter set was optimized against the quasi-experimental trace to see if the parameters can return to their original values, and if they do, will all of the parameter sets be the same once optimized. By varying all of the parameters and performing this procedure, it was found that the parameters do not tend to return to their original values yet, still achieve similar quality fits. By reducing the parameter set that is allowed to be optimized and fixing some parameters to their original values, it is possible to construct a set of parameters that tend to return to their original parameter values after being scattered with the Sobol sequence technique.

To accomplish this, a large set of initial guesses for the free parameters was made (as described), and the fixed parameters were held at their originally optimized values (the values were used to produce the quasi-experimental trace). Then each of those thousands of initial guesses was optimized using the same optimization process as when fitting the traces to experimental data. With this optimization complete, a reasonable representation of the identifiability of the free parameter set in question can be made. However, all of the parameter sets that achieved a fit that was within 20% of the best-fit score (or 10 by count if the score of the fit parameters was not close to the best fit), were taken and randomized within 5% of their returned value to test for convergence problem. Those parameter sets were then rerun through the optimization routine with higher tolerances. With that final set acquired, an analysis of the practical identifiability of the parameters can be made.

D Sobol Sequence Background

When sampling from a higher dimensional space, either the random uniform or Latin hypercube sampling techniques may be used. Random uniform works by randomly taking a point from a one-dimensional uniform distribution for each index of the higher dimensional coordinate. This generates a random point in N-dimensional hypercube. The Latin hypercube improves this technique if a certain number of points are needed and a good spread of the space that is being sampled is required by breaking the hyperspace up into N-dimensional, cuboid, sub-regions. Then the points that are chosen are not allowed to exist in the slice (in any dimension) as another point. This helps more evenly spread out the points. However, because of a distortion that occurs in 4+ dimensional hypercubes (using Euclidean geometry), either of these two methods becomes statistically biased in higher dimensions. There is a technique called Sobol sequencing that can preemptively produce a set of points that more evenly covers higher dimensional space (in a manner that introduces the variability of randomness) as long as the desired number of samples is known beforehand while also eliminating the aforementioned bias (Figure 30) (Renardy et al., 2021). Since the parameter sets that are being used in this analysis exist in 6-dimensional hyperspace or higher, Sobol sequencing was used to randomize parameters whenever needed.

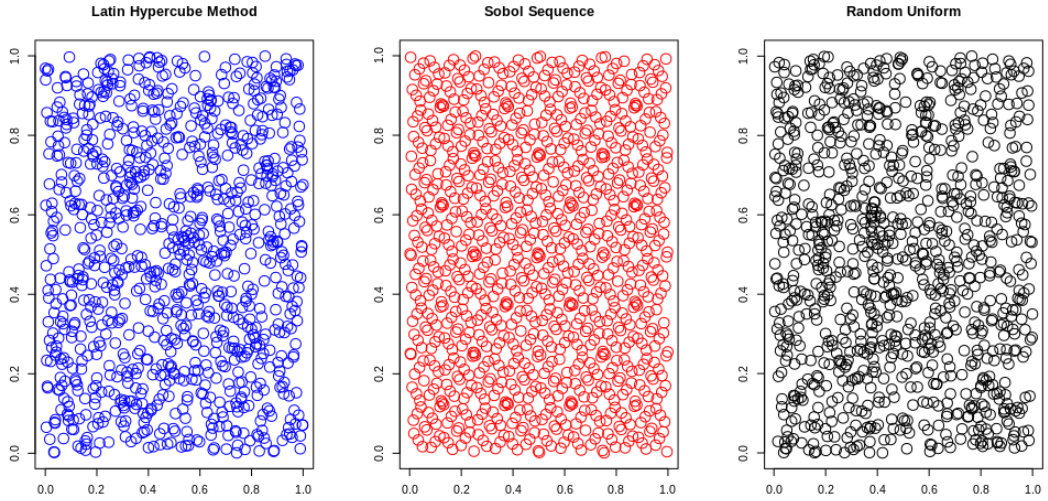

Figure 30: Plot of Latin hypercube method, Sobol sequencing, and a random uniform sample in 2 dimensions with 1000 samples each.

With Sobol sequences, the parameter space that is sampled from is better represented since there are fewer regions of high sampling density that would occur by random chance with one of the other two techniques. This is also true of the Latin hypercube method while also managing to reduce the effect of bias in higher dimensions.

E Group Comparison Details

With the initial set of parameters, take the point-wise average of all the experimental traces in a given group and optimize the sodium parameters to each of those group averages. This gives a parameter set for each group in question. For instance, this could be the TTX-sensitive group and the single-point mutation group. At this stage, simulated experimental data can be generated for each group. This can be done by starting with the group parameters and randomizing the sodium parameters (in a small window -- 10%) in the parameter set for the group in question and then using the model to create a quasi-experimental trace from that parameter set. Now, these quasi-experimental traces can be used to attempt to recover the original values.

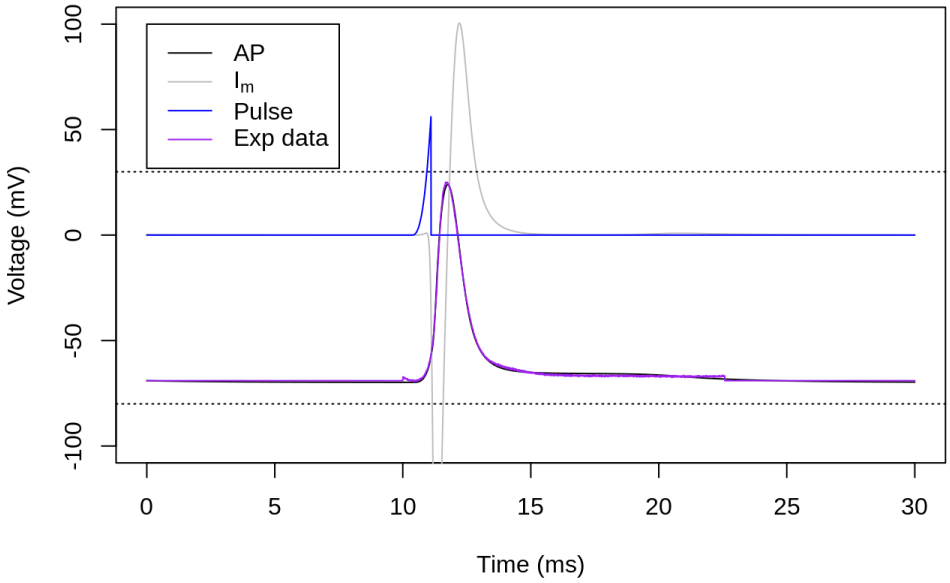

Figure 31: Action potential fit to the TTX-sensitive group average (same color scheme as before).

Now, a set of randomized, non-sodium parameters can be created via Sobol sequences; many of these parameter sets will be created in order to test the hypothesis that the model can accurately discriminate between groups even if the non-sodium parameters are unknown -- which is the case. Consider one of those initial parameter

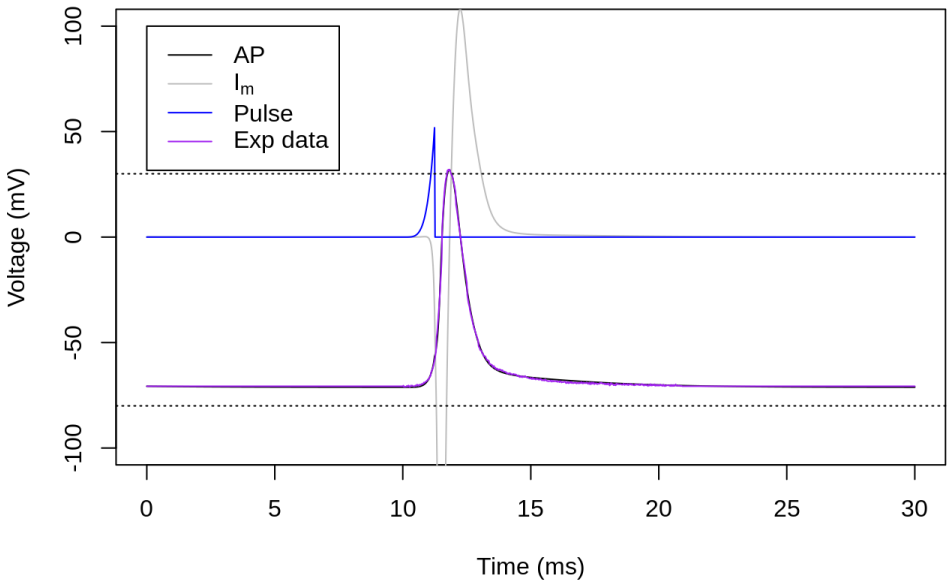

Figure 32: Action potential fit to the P group average (same color scheme as before).

sets and use the optimizer to fit each quasi-experimental trace within the group. This will generate a table of sodium parameter values where every non-sodium parameter value is identical. Repeat this for each set of randomized, non-sodium parameters to achieve many tables for which to deceiver differences between the two groups (Appendix J.2).

At this point, the mean group differences can be used to generate histograms that show the shape of the experimental distribution of change in parameter values (Appendix F). Next, the data from each table that was generated as described before were combined into a large table, and the `LME` (Linear Mixed Effects) method in `R` was used to analyze the data. Figure 28 gives a flowchart of this procedure.

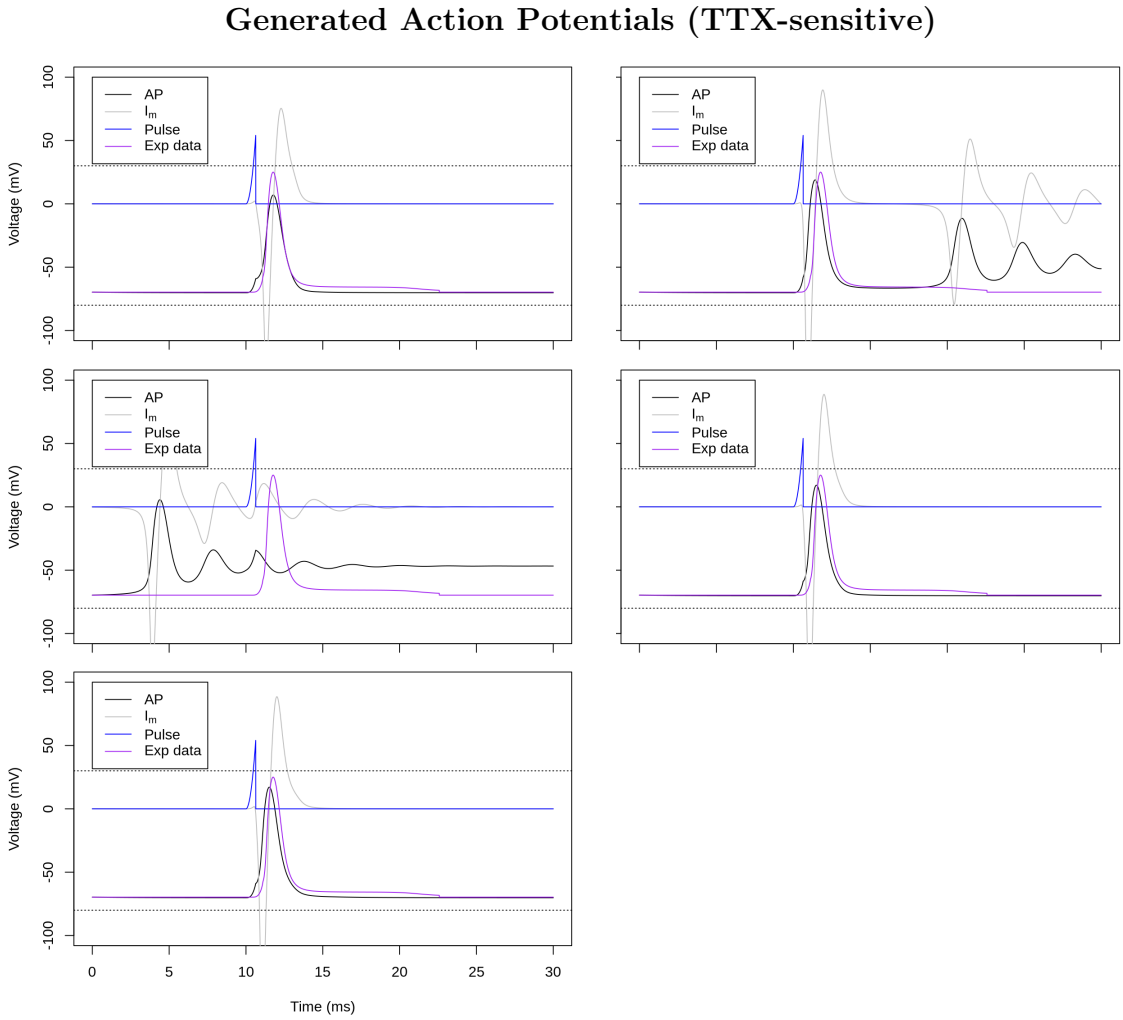

Figure 33: Action potentials randomly generated near the TTX-sensitive average. The purple line represents the TTX-sensitive average from Figure 31 (same in all plots), and the black line is the quasi-experimental action potential.

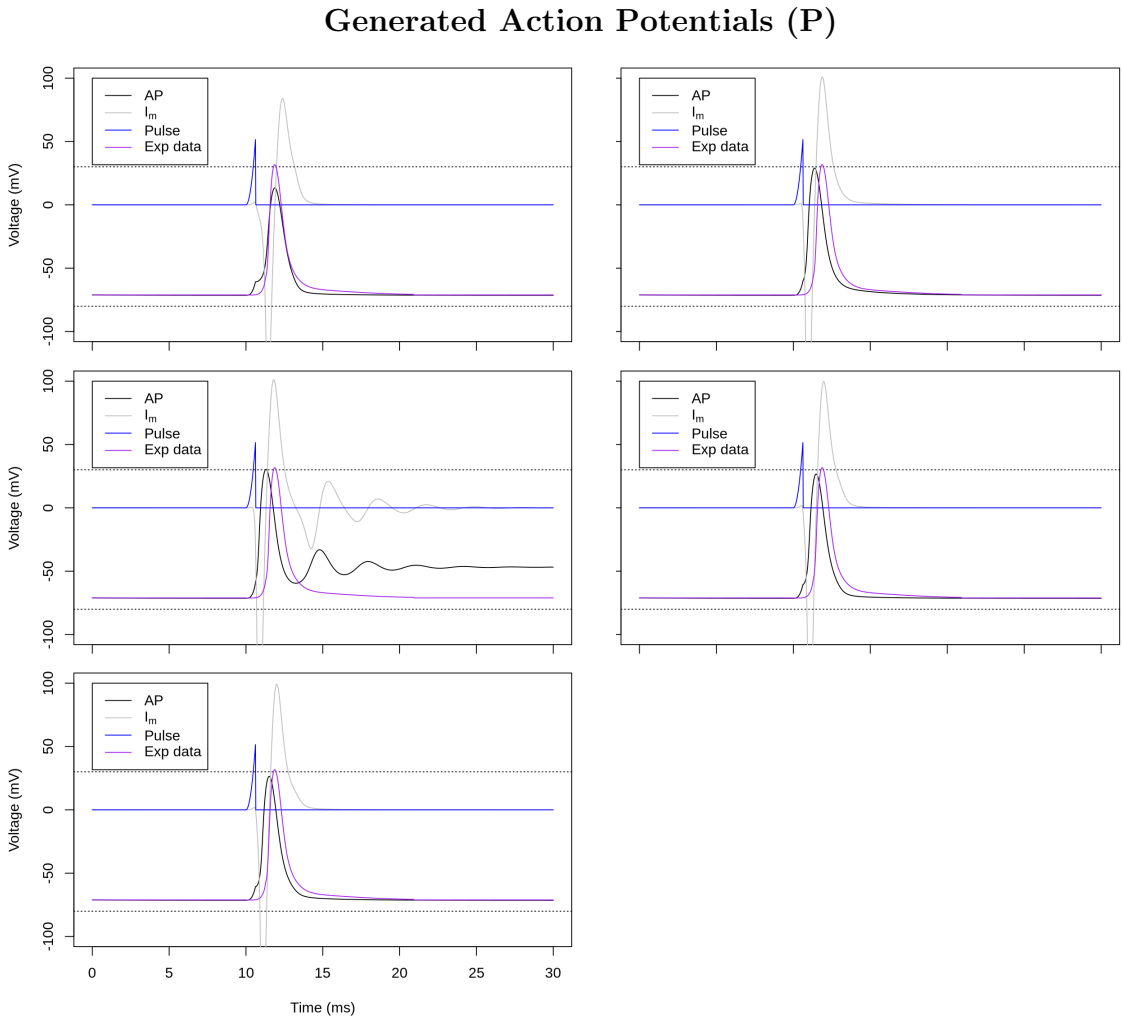

Figure 34: Action potentials randomly generated near the P average. The purple line represents the P average from Figure 32 (same in all plots), and the black line is the quasi-experimental action potential.

F Group Differences: Histogram Visualization

Once a list of tables is generated for each group as mentioned in Section 4, the groups can be compared against each other. To do this, take the average of each sodium parameter associated with quasi-experimental traces. Then take the difference of that average parameter as found in the first group and that same parameter averaged across the second group. This will give a relative measure of the magnitude of each parameter as compared across groups. For each average parameter difference, a histogram can be made to visualize the frequency of differences between the groups (Figure 35). The hope is that each parameter difference will be notably different from 0, especially for the G_{Na}. However, this approach does not easily provide statistical information on which to base a conclusion about the differences between the + and P groups.

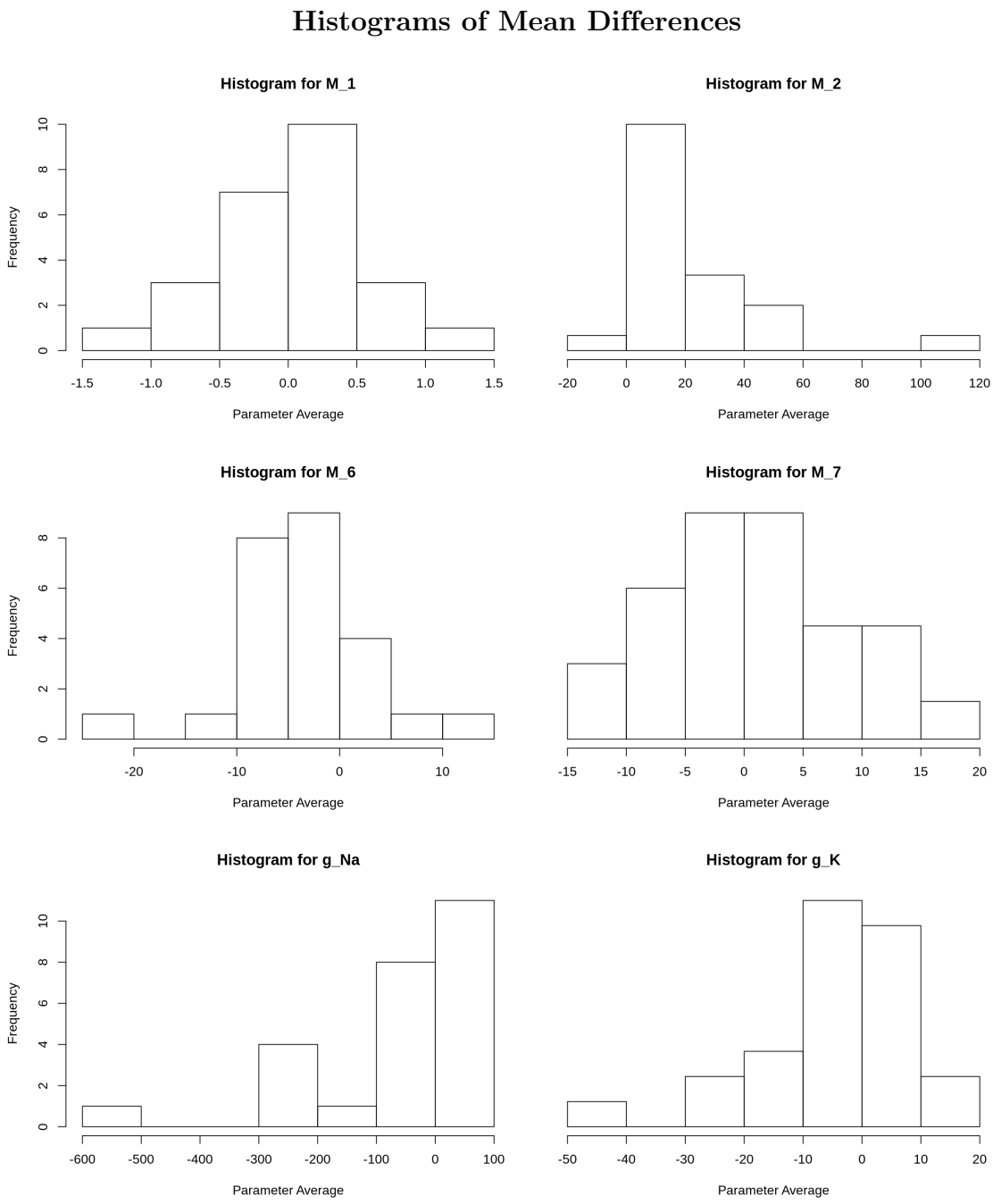

Figure 35: Histograms of the different group means across all tables generated. The difference was calculated with P - TTX-sensitive, so an average (frequency) that is positive number means the P parameter value was larger, and a negative number means the TTX-sensitive parameter value was larger.

G Practical Identifiability Plots

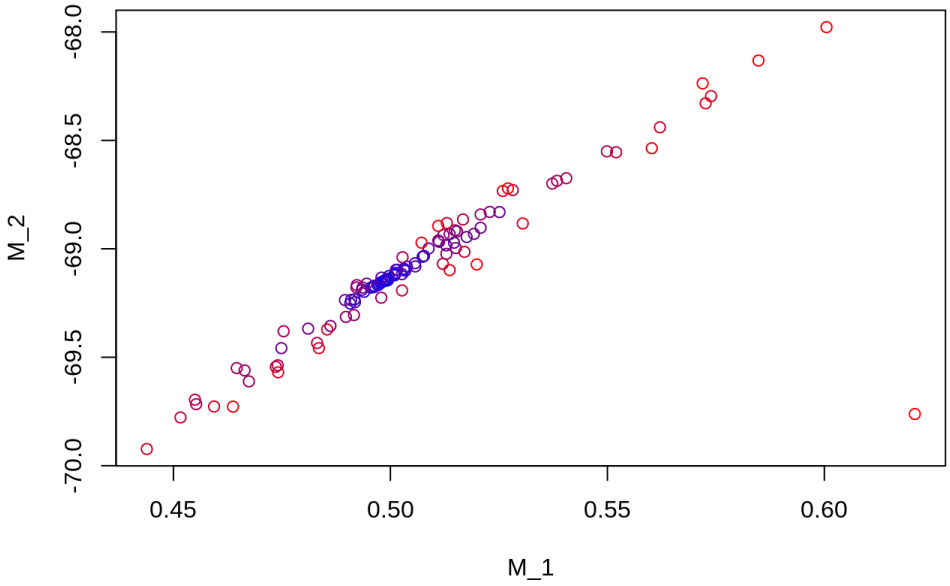

Figure 36: M_1 vs M_2 scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

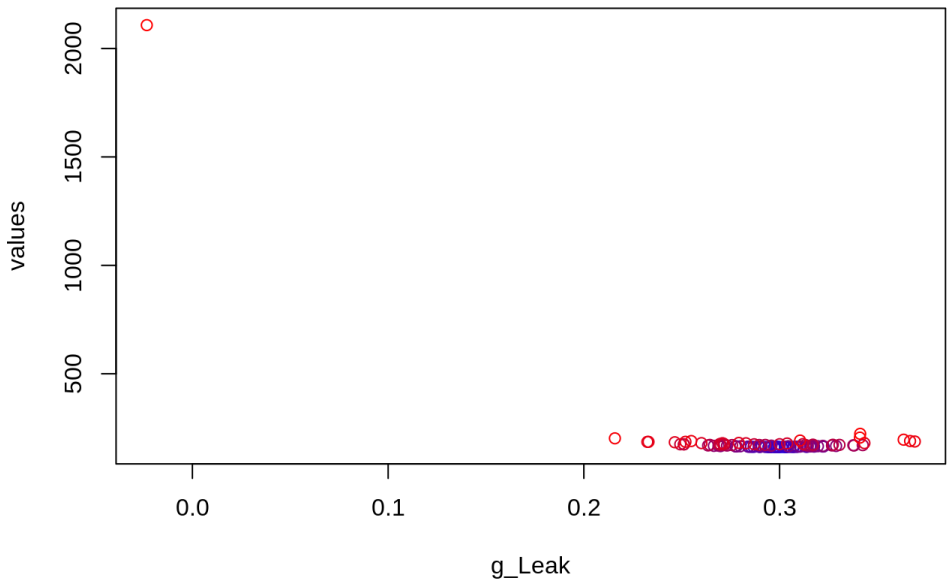

Figure 63: G_{Leak} vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

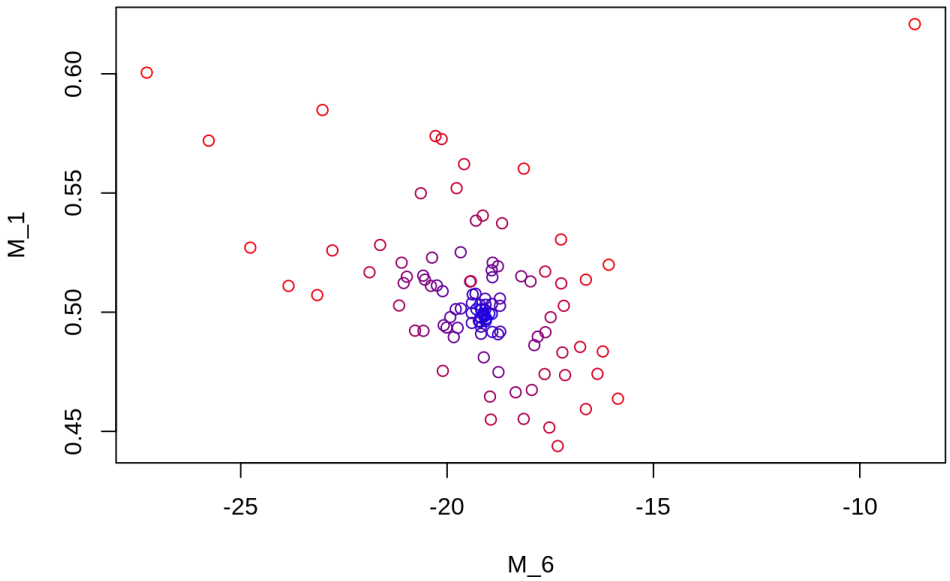

Figure 37: M_1 vs M_6 scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

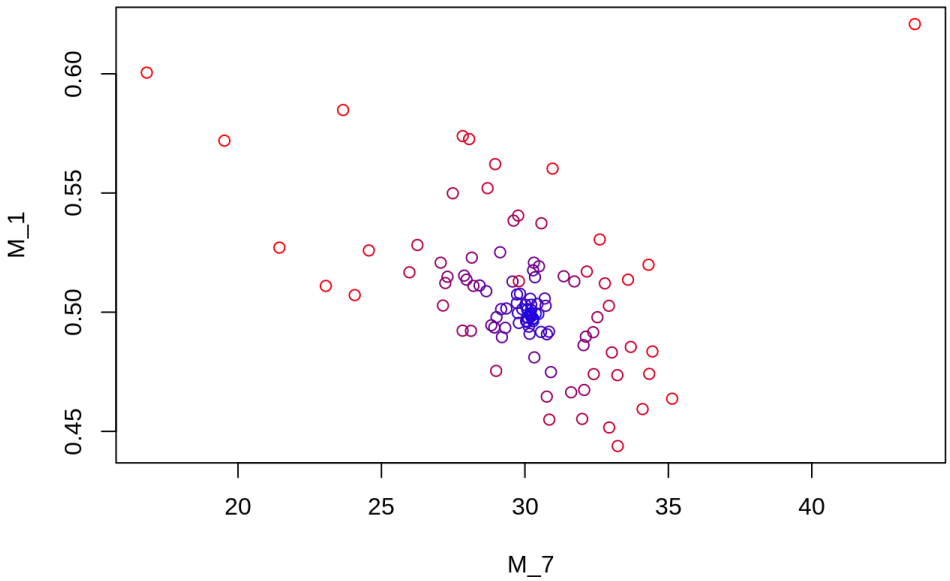

Figure 38: M_1 vs M_7 scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

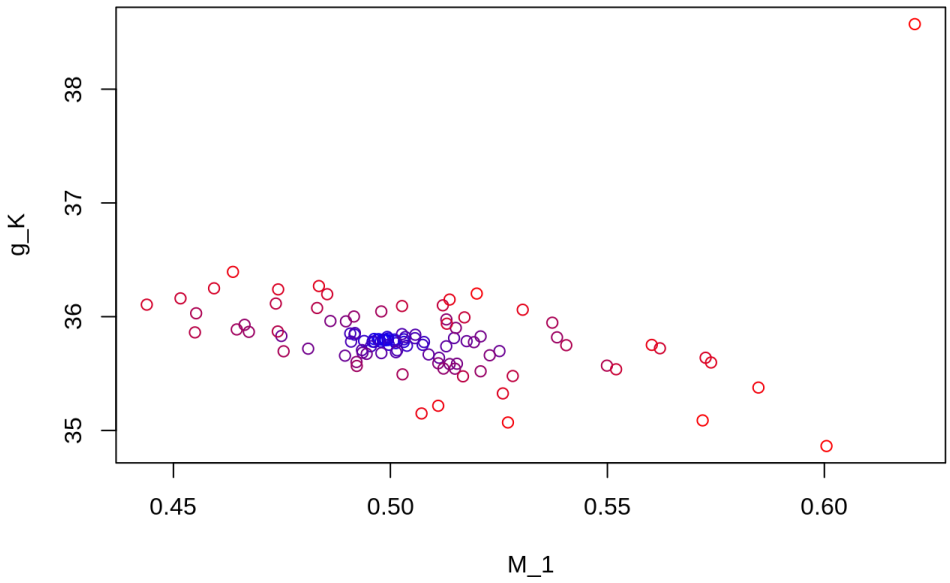

Figure 39: M_1 vs G_K scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

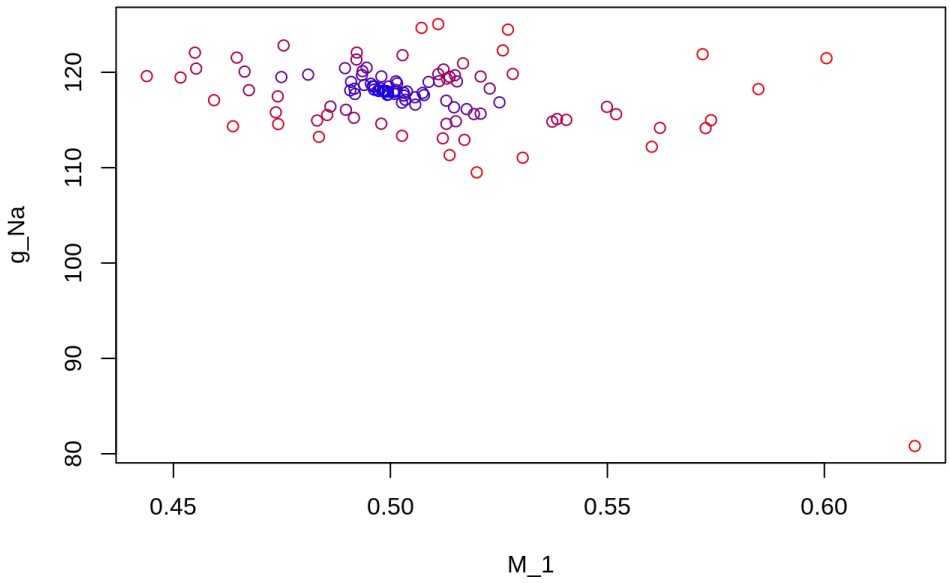

Figure 40: M_1 vs G_{Na} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

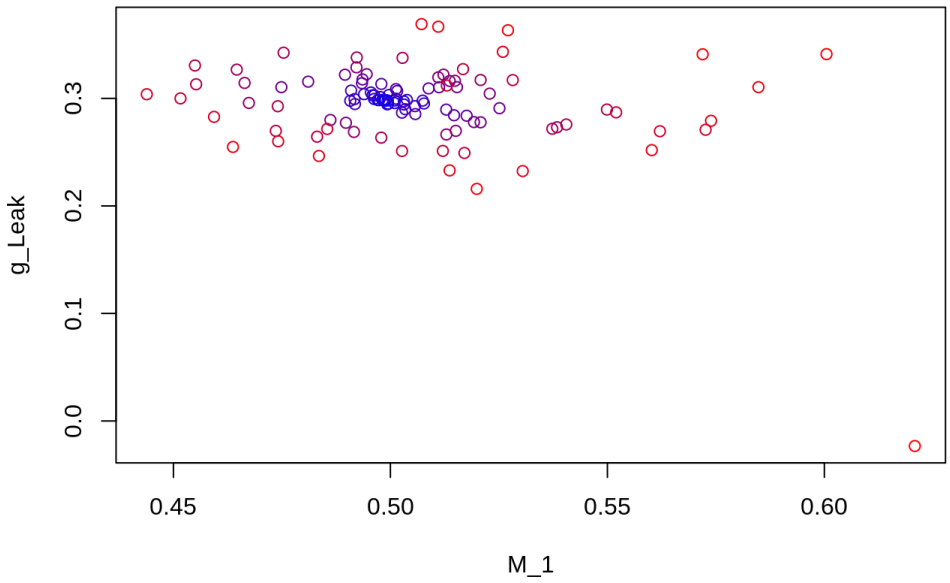

Figure 41: M_1 vs G_{Leak} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

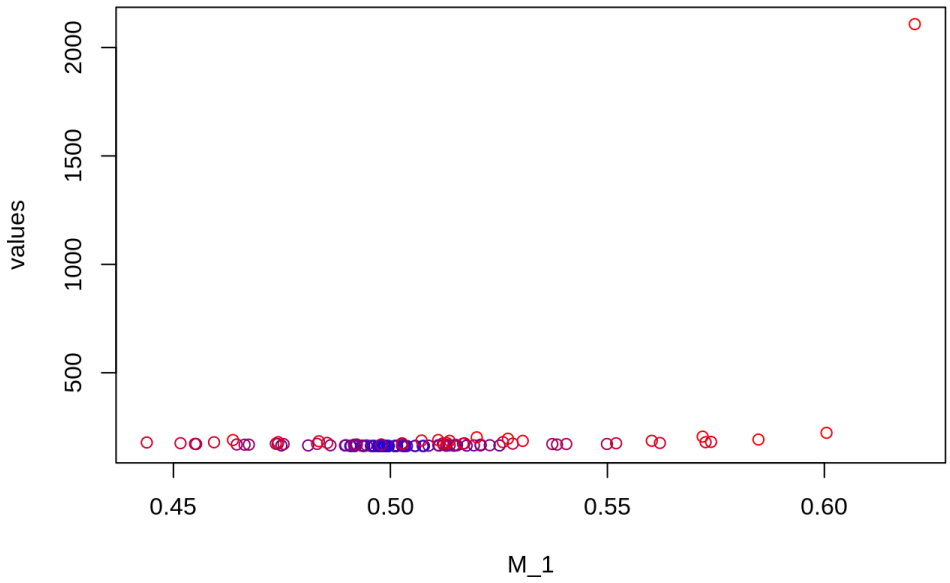

Figure 42: M_1 vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

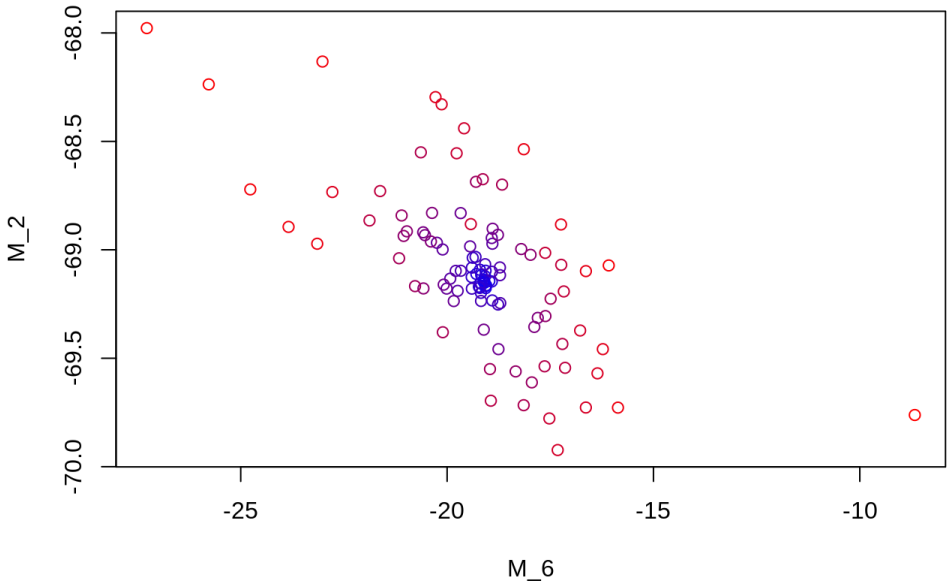

Figure 43: M_2 vs M_6 scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

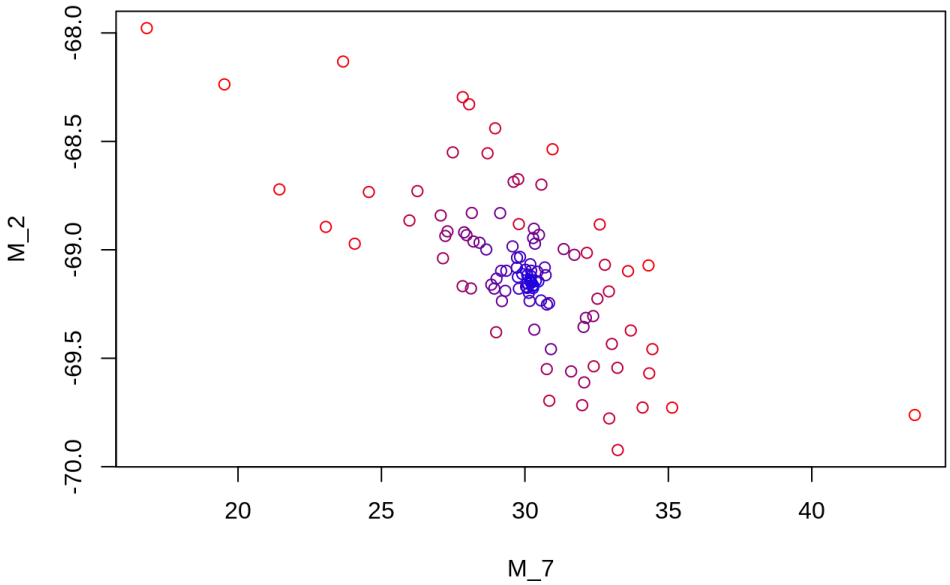

Figure 44: M_2 vs M_7 scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

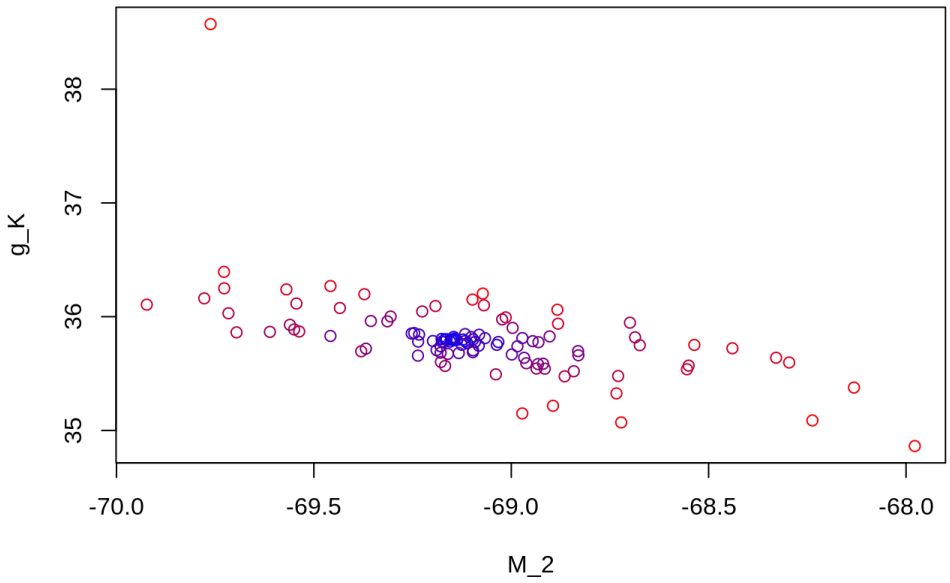

Figure 45: M_2 vs G_K scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

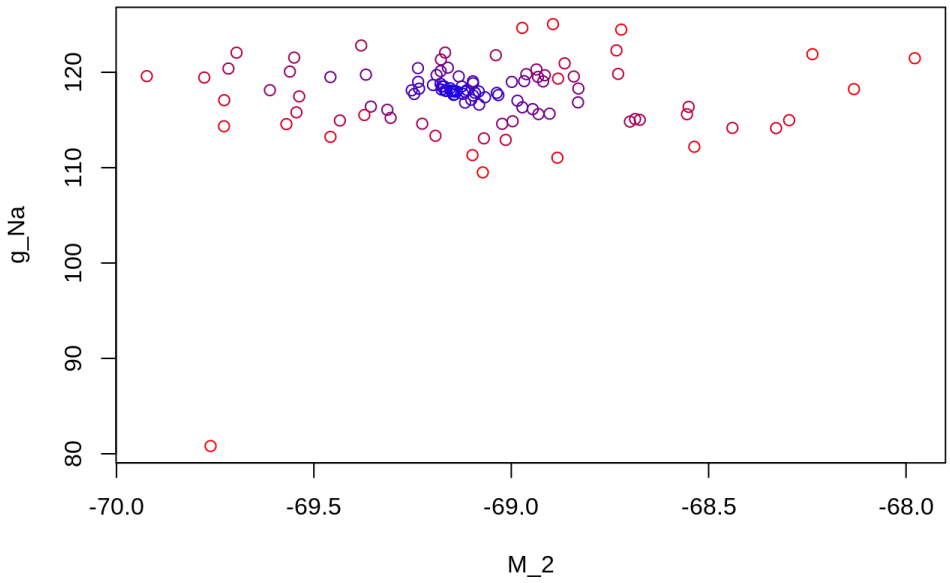

Figure 46: M_2 vs G_{Na} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

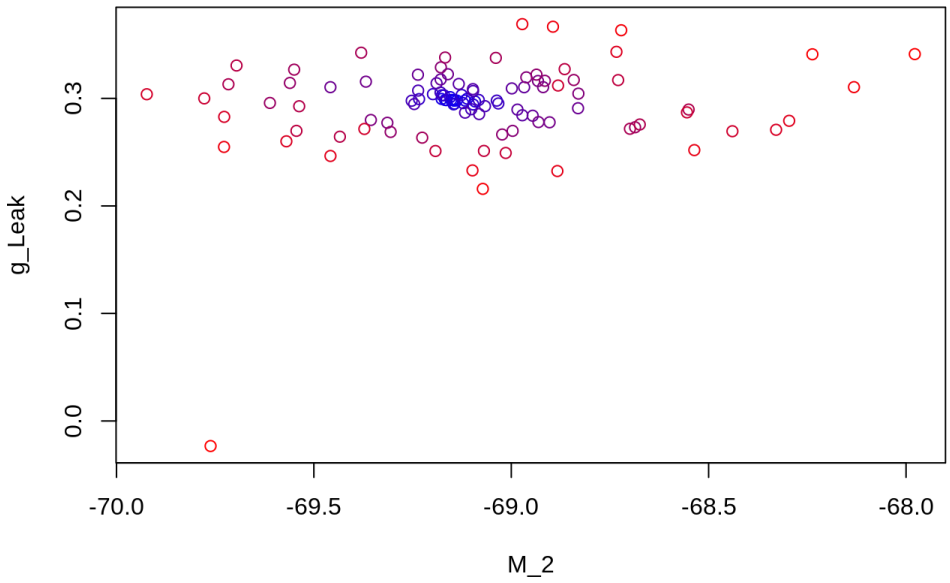

Figure 47: M_2 vs G_{Leak} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

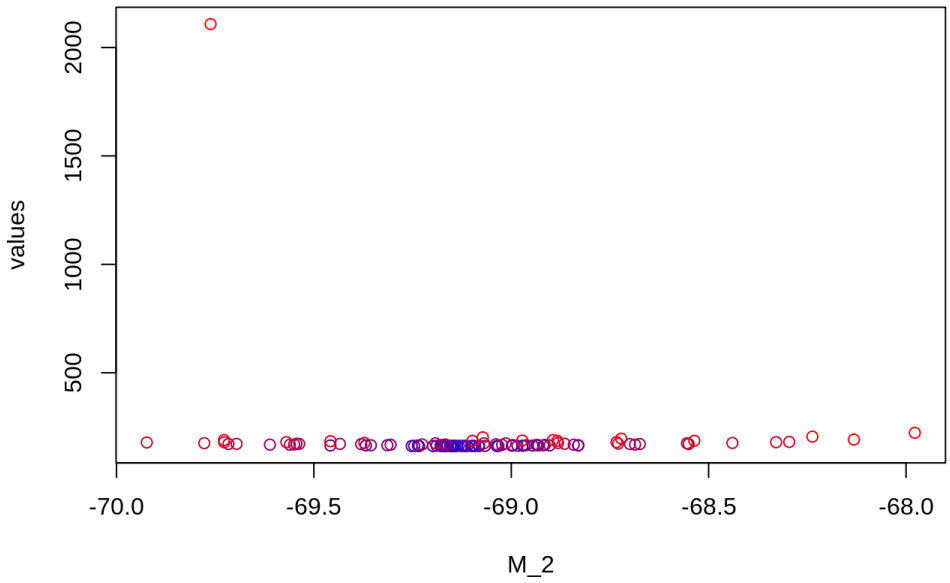

Figure 48: M_2 vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

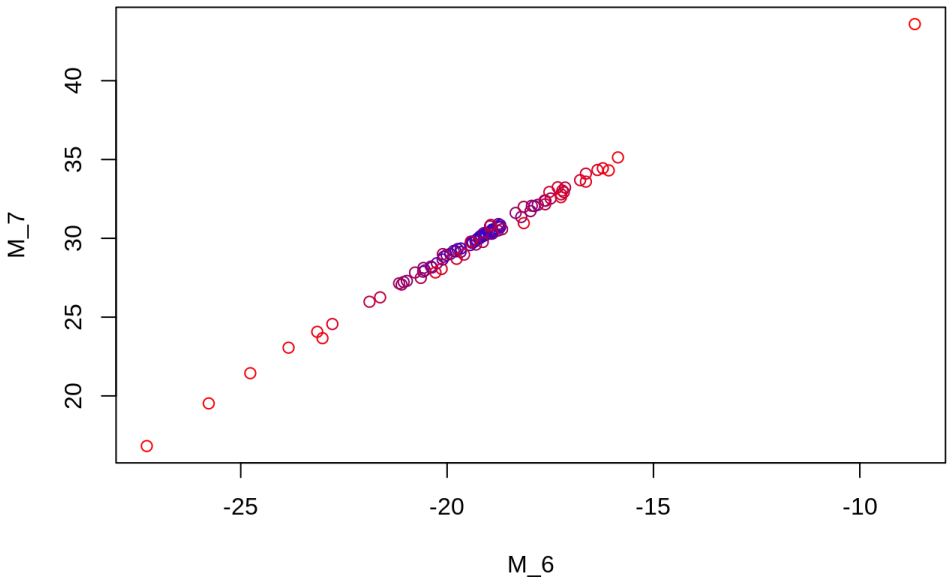

Figure 49: M_6 vs M_7 scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

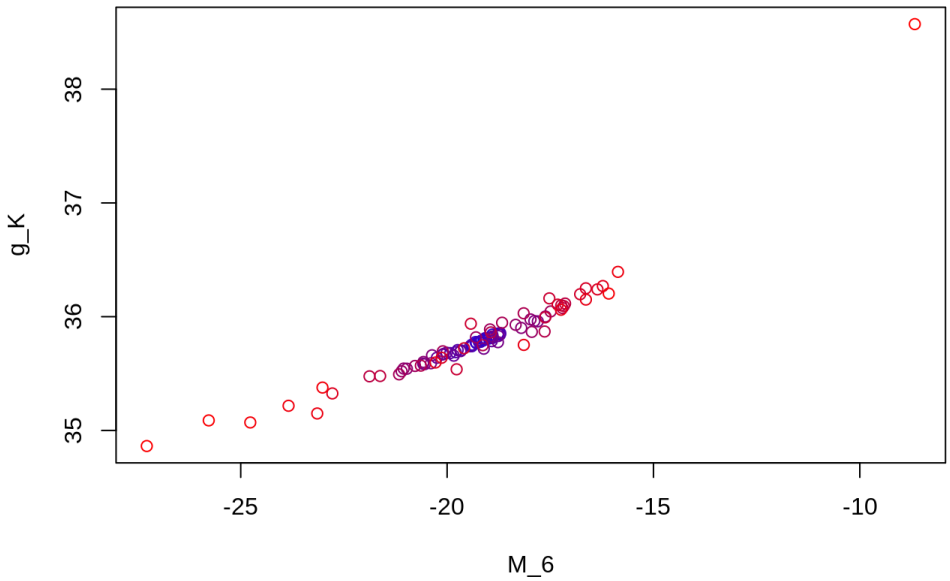

Figure 50: M_6 vs G_K scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

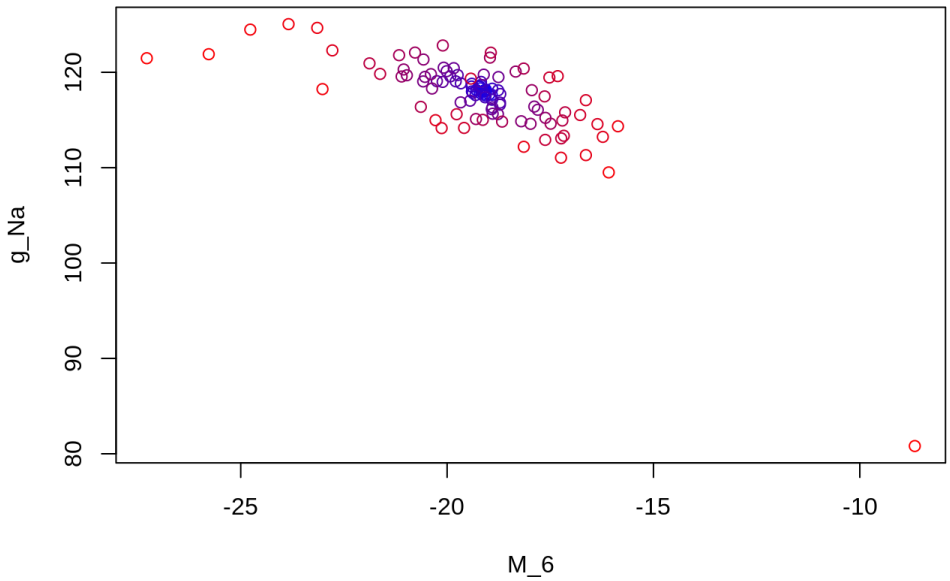

Figure 51: M_6 vs G_{Na} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

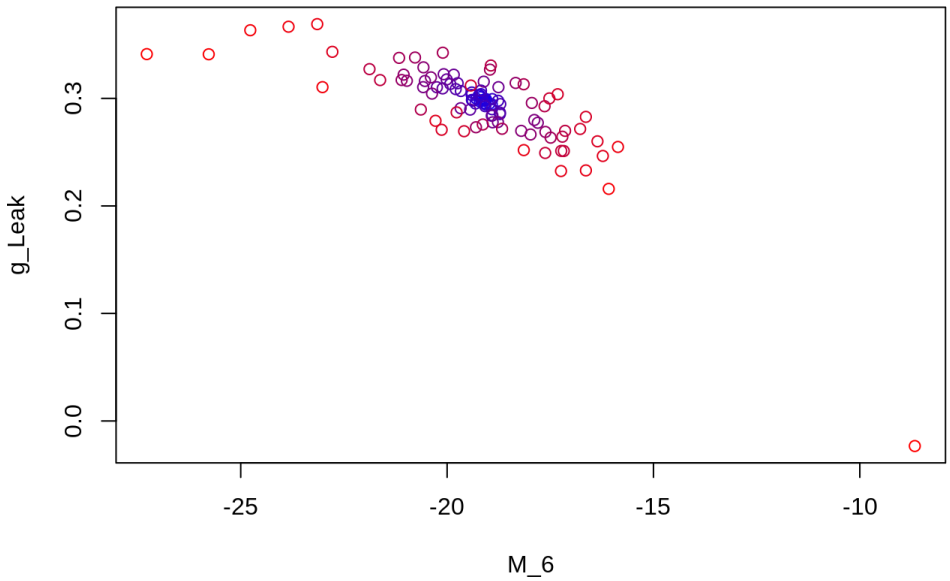

Figure 52: M_6 vs G_{Leak} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

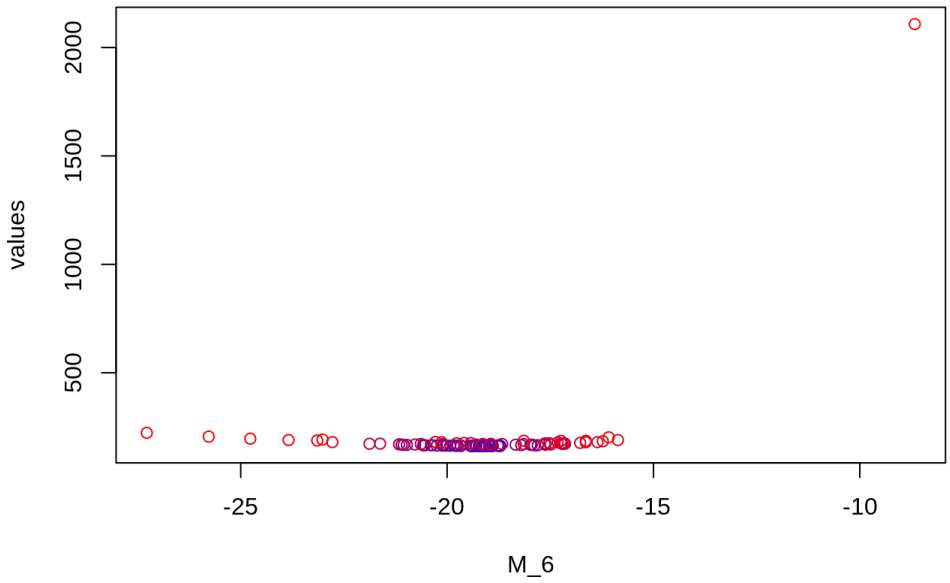

Figure 53: M_6 vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

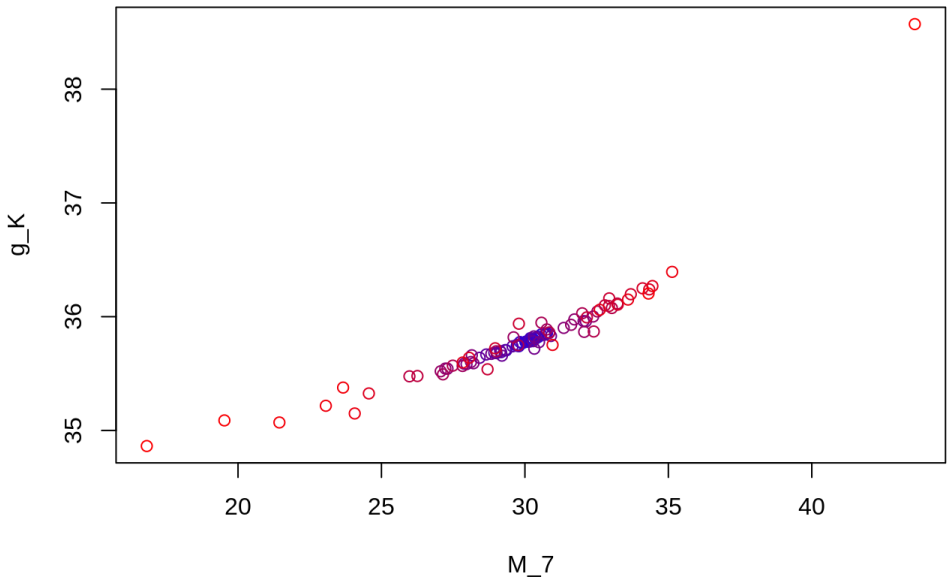

Figure 54: M_7 vs G_K scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

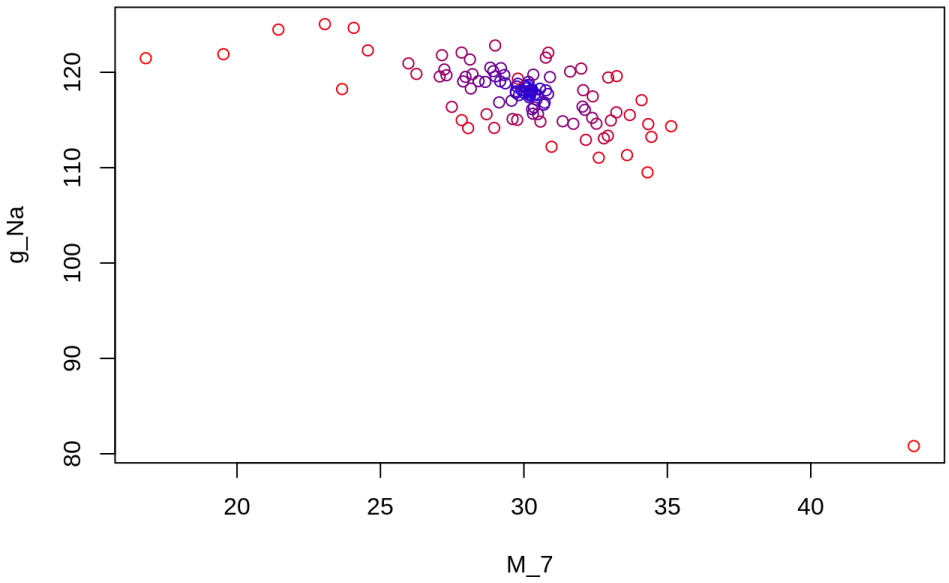

Figure 55: M_7 vs G_{Na} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

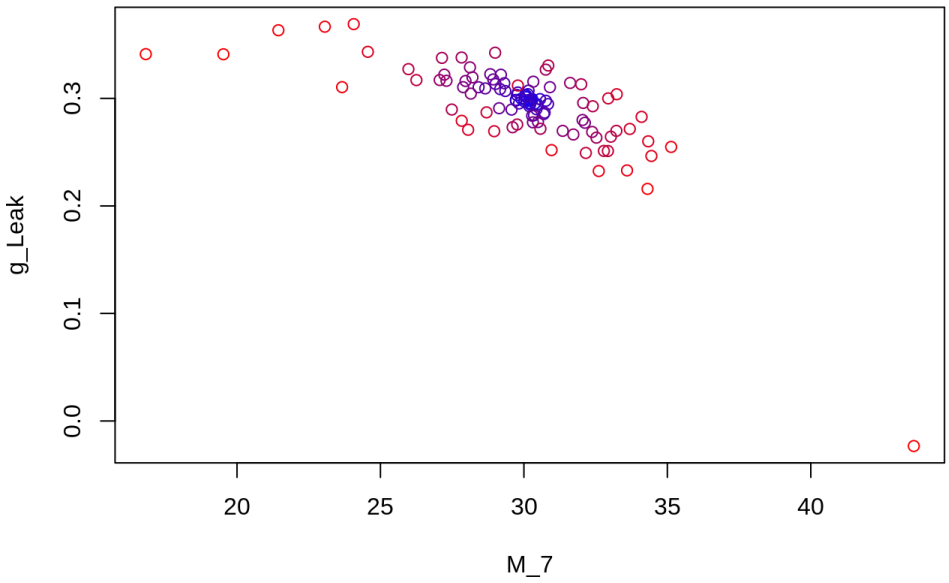

Figure 56: M_7 vs G_{Leak} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

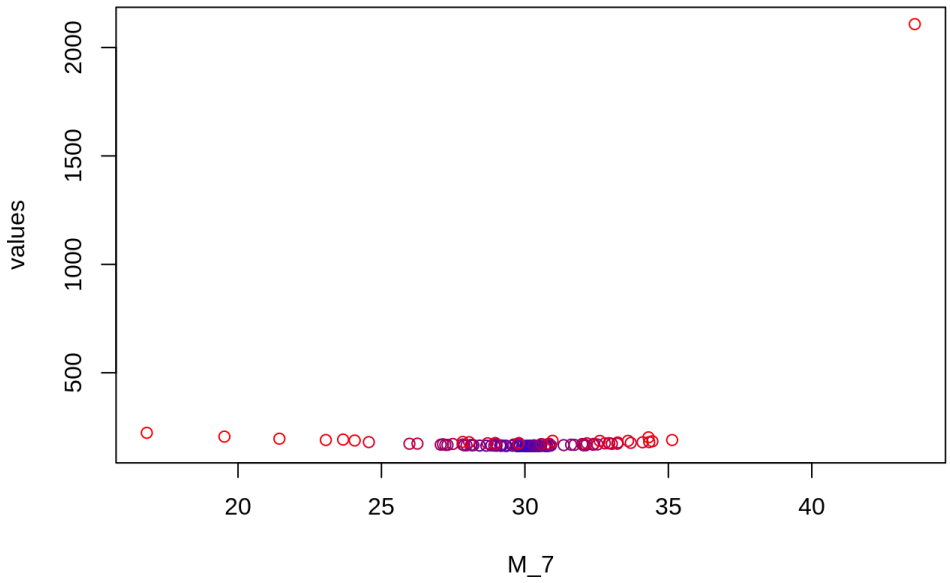

Figure 57: M_7 vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

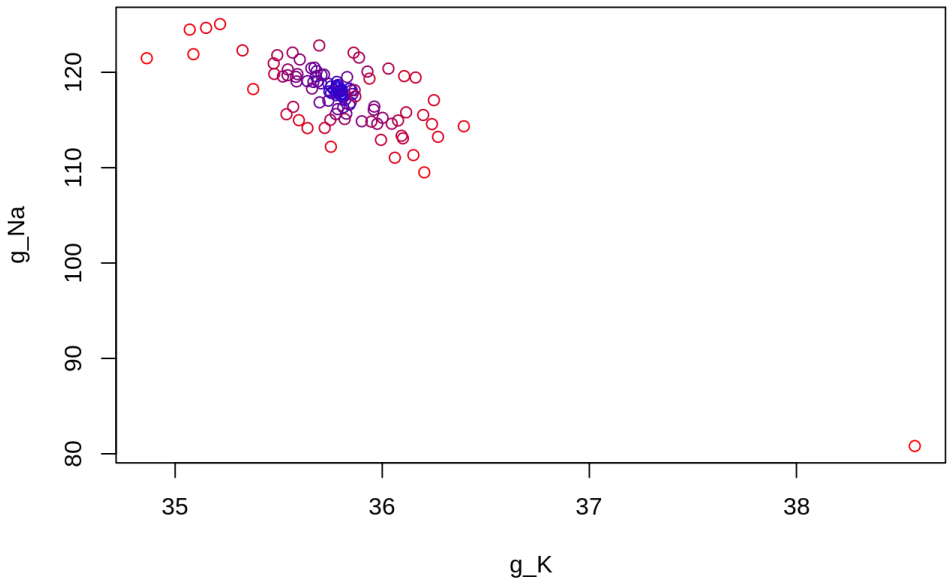

Figure 58: G_K vs G_{Na} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

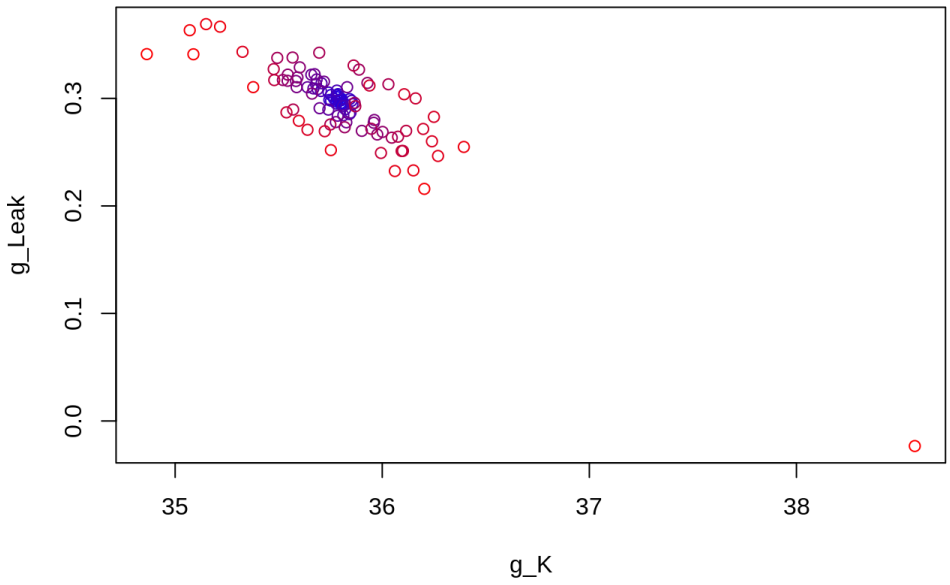

Figure 59: G_K vs G_{Leak} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

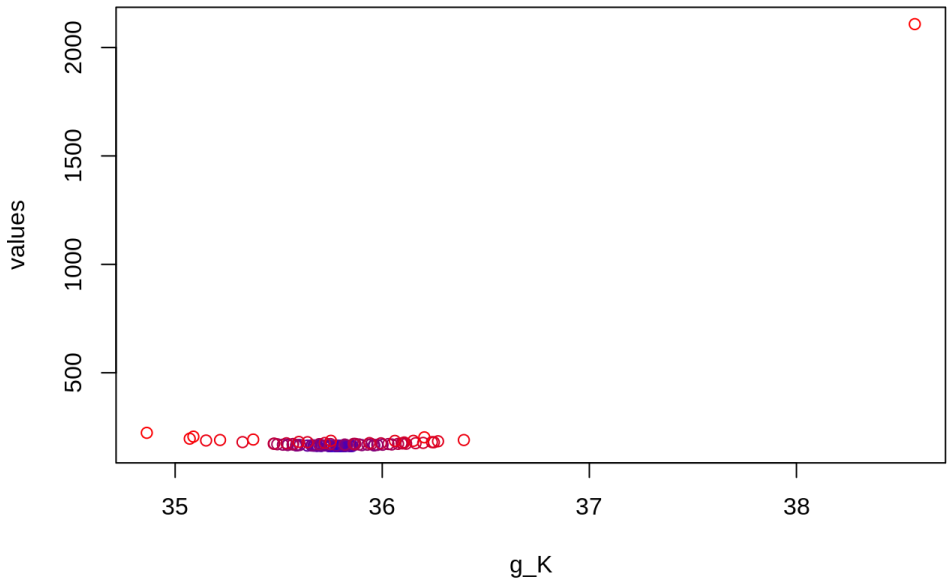

Figure 60: G_K vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

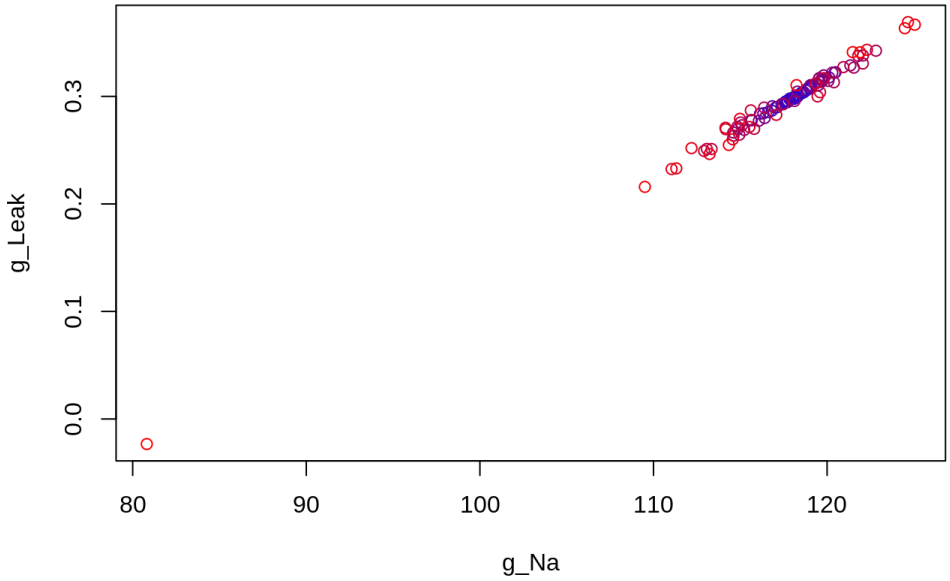

Figure 61: G_{Na} vs G_{Leak} scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

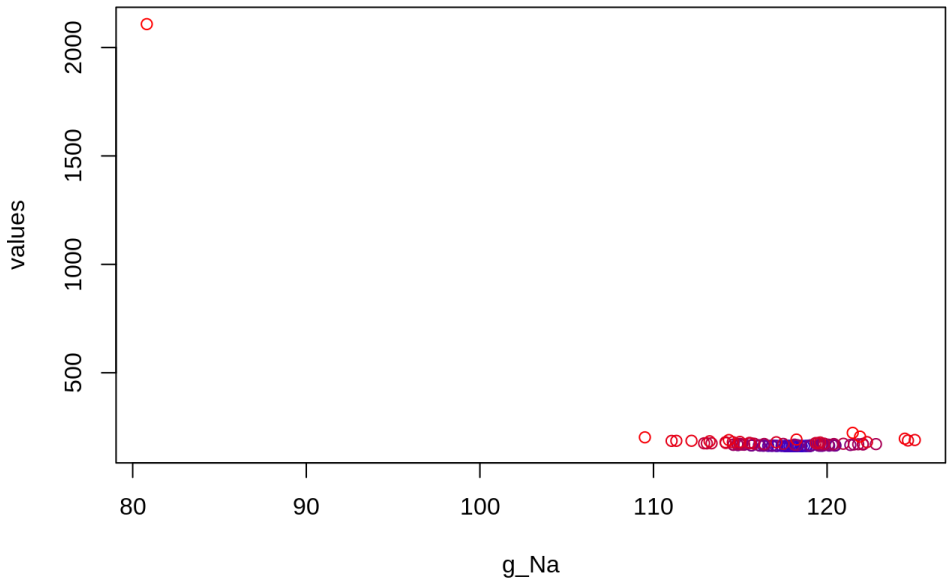

Figure 62: G_{Na} vs Values scatter plot of fits (within 20% of the best-fit value) of optimized parameters used for testing uniqueness while optimizing over M_i and G_i.

H Parameter Variation Figures

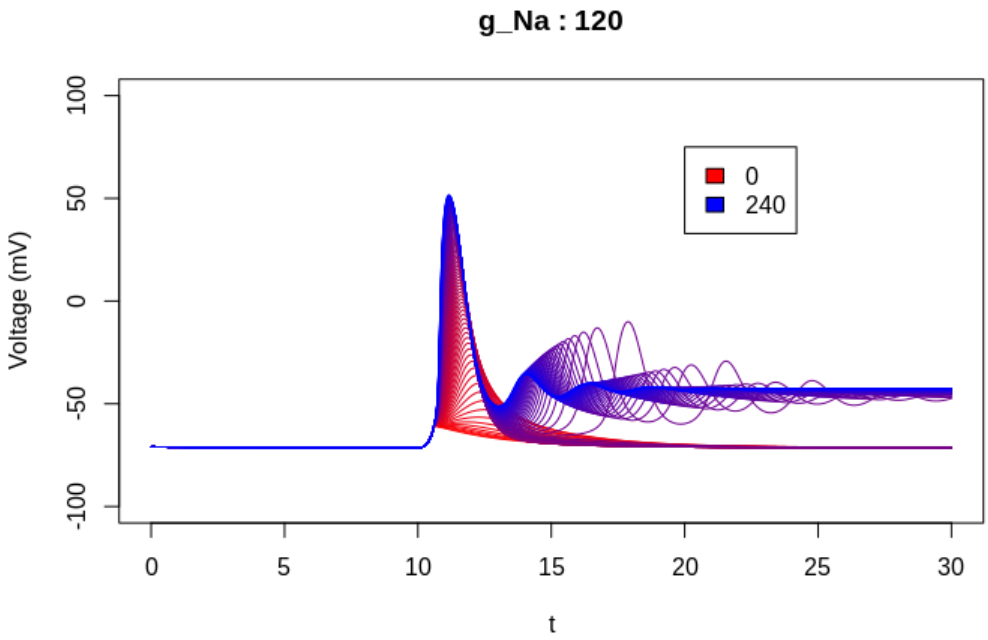

Figure 64: Visualization of the G_{Na} parameter though a range of values

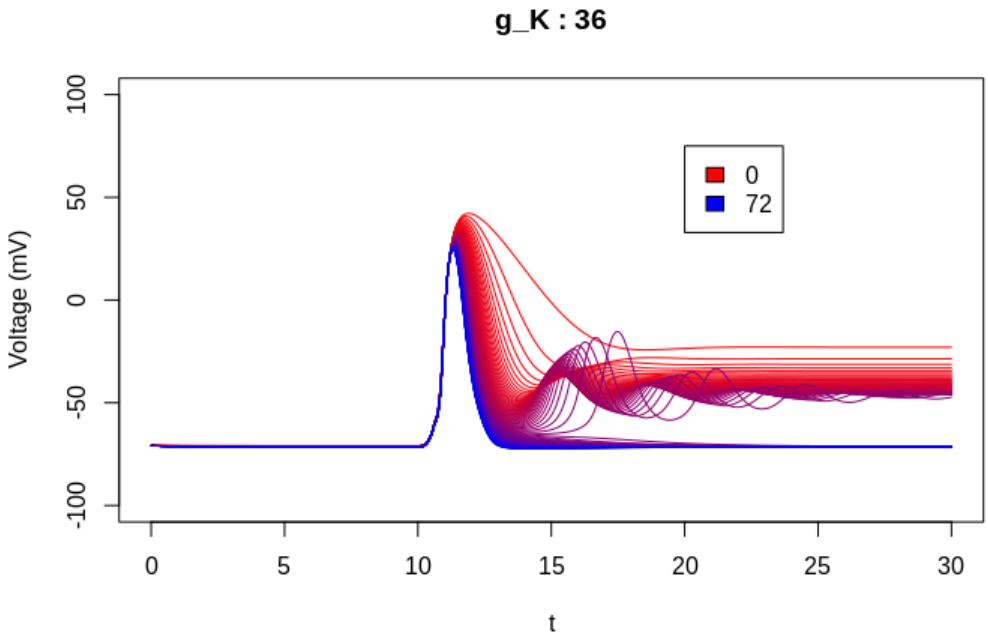

Figure 65: Visualization of the G_K parameter though a range of values

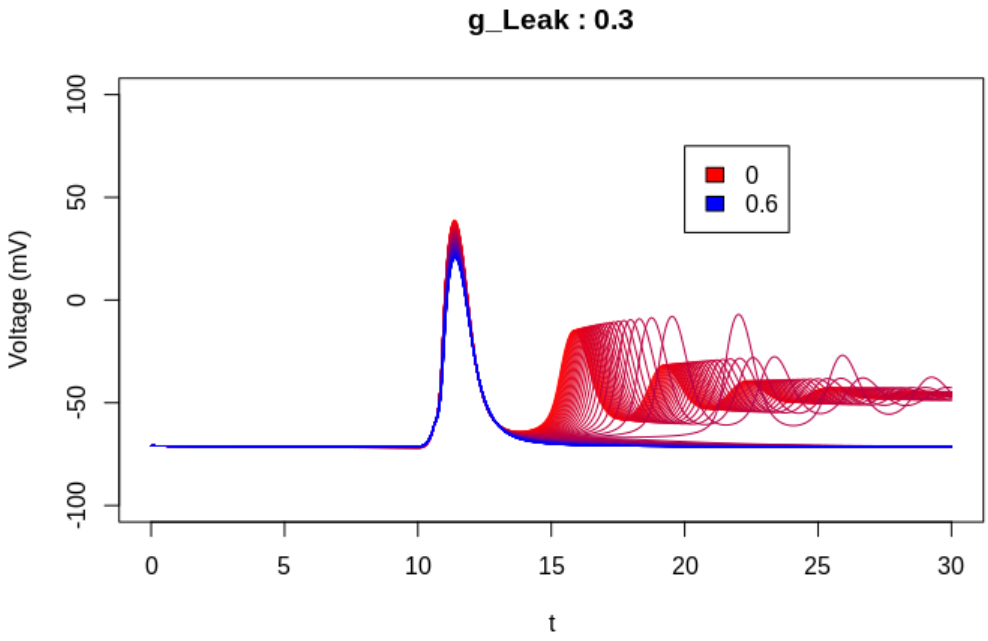

Figure 66: Visualization of the G_{Leak} parameter though a range of values

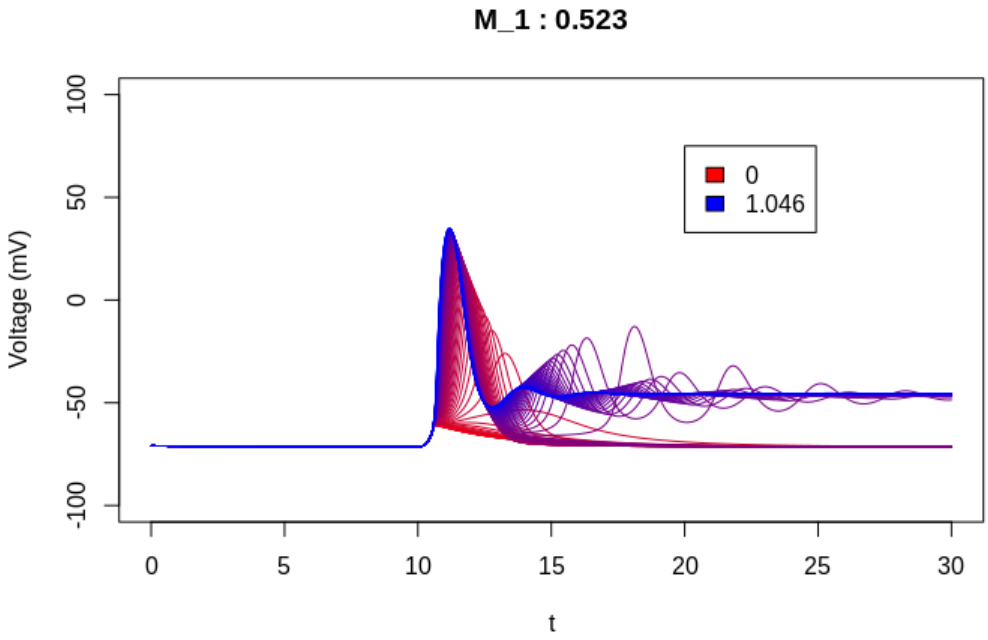

Figure 67: Visualization of the M_1 parameter though a range of values

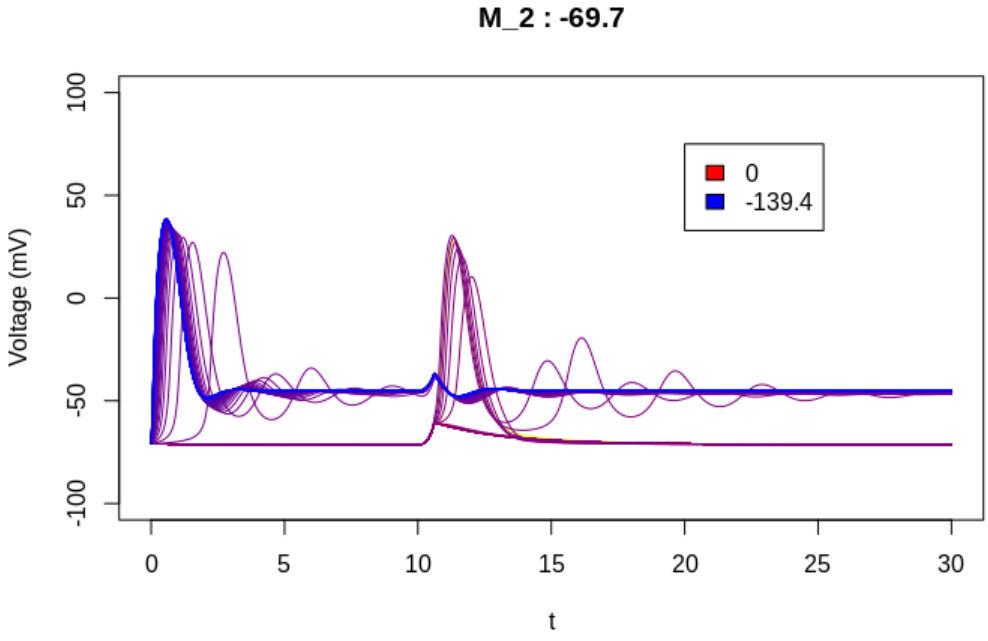

Figure 68: Visualization of the M_2 parameter though a range of values

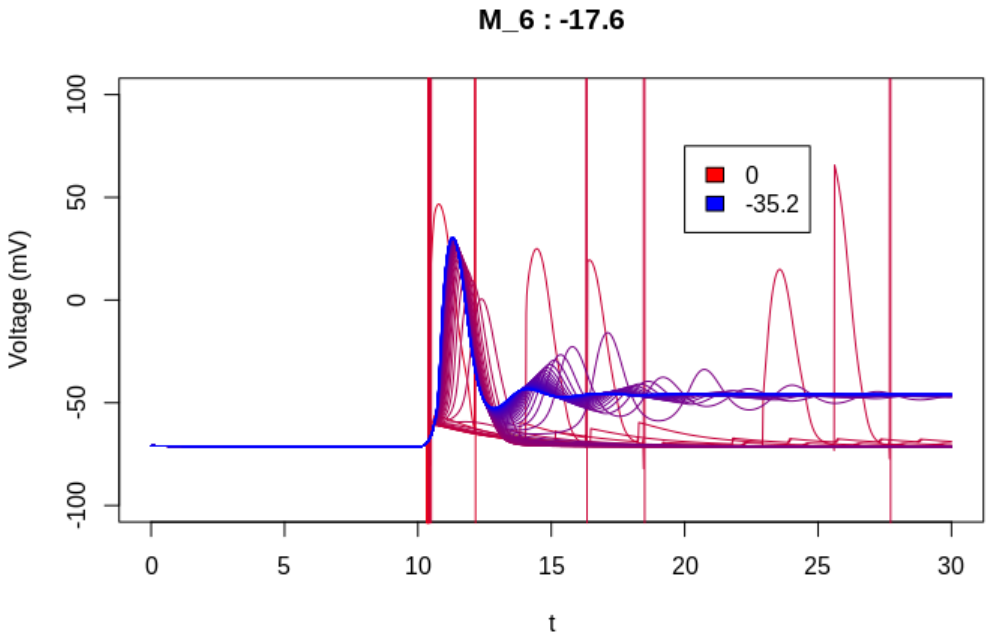

Figure 69: Visualization of the M_6 parameter though a range of values

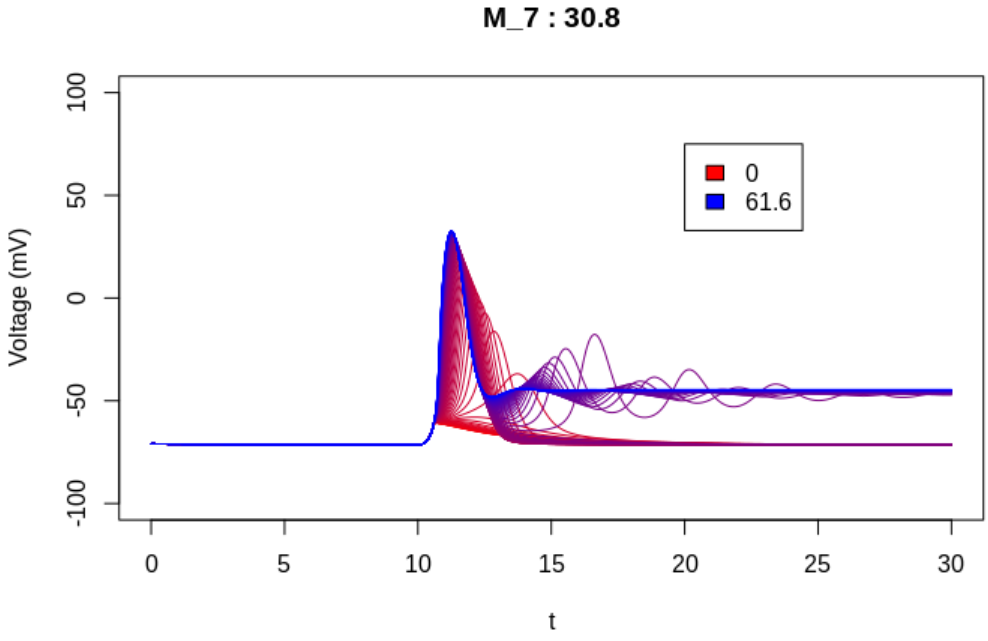

Figure 70: Visualization of the M_7 parameter though a range of values

I Programs in R

I.1 Example Implementation

Given below is a simplified implementation of the `ActionPotential` object in `R`. Implementations used in this thesis can be found in Appendix J.

```
# The ActionPotential(parameters, trace_data, time_data, name="Name
    ")
#   implementation creates an action potetnial object with given
#   data about the trace.
AP = ActionPotential(par_0, trace_data_ex, time_data_ex,"Main_Doc")

# find_foot() finds the start of the action potential and
#   determines the shape of the foot
AP$find_foot()

# optimize(paramters) is a member function that optimizes the
#   given paramters against the trace data supplied
AP$optimize(opt_par_0)

# display_action_potential() shows all of the plots for
#   corresponding to the current paramters
AP$display_all_plots()
```

Listing 1: Example Implementation

I.2 Action Potential Object

This (S4) object contains the set of parameters that are used in the optimization as well as the specific trace that the optimizer will use to fit the parameters. There are a handful of other variables that are stored in the object to keep track of the state of the model such as the RMP value and the values used to calculate the stimulus shape (that stimulus is shaped by the foot finding optimization as described in A.2). This object is called with `ActionPotential(parameters, trace_data, time_data)` where `parameters` is a vector of parameterized used to initialize the object, `trace_data` is the experimental data (in mV), and `time_data` is the times at which each data point was taken.

I.2.1 Main Object Declaration

The `ActionPotential` object contains the data and related information for a single action potential trace and its corresponding parameters. The `ActionPotential` object contains the following member variables:

- `all_params`: a named vector of values for the parameters that correspond to the variables described in Section 2.2. See Appendix I.4 for an example.
- `trace_data`: a vector of voltage values (in mV) that describe the action potential trace which could be experimentally derived or it could be a quasi-experimental trace that is manufactured and used as a reference for optimization.
- `dt`: stores the time between each sample given the experimental data (time-step). In the initialization, the object is given a list of times corresponding to each voltage sample and the time step.
- `AP_name`: an optional parameter that stores the name of the `ActionPotential` object that can be used for clarification or debugging in large programs

- `n`, `m`, and `h`: subunit particle activity percentages corresponding to the n, m, and h values described in Section 2.2. These three members are all of the `Subunit` class which is defined in Appendix I.3.

- `K`, `Na`, and `Leak`: Are `Channel` objects as described in Appendix I.3. These hold the emf values and conductance values for their respective channel proteins.

- `stim_d`: a number that represents the duration of the stimulus (in ms).

- `stim_h`: a number that represents the heights of the stimulus (in mV).

- `stim_dim`: a number that represents the dimension of the stimulus function as described in Appendix A.2.

- `stim_A`: a number that represents the area of stimulus (in $mV \cdot ms$).

- `time`: the duration of the action potential including stabilization time before and after the depolarization/repolarization phases.

- `t`: a sequence of numbers that represent each time step of the action potential. This sequence is inferred from `dt`, and should look like the inputted time data (assuming the time points are all evenly spaced in the source data), but is generated inside the object to ensure consistency with calculations and plotting.

- `stabil_time`: a number that indicates how long the action potential model should wait before applying the stimulus. This allows time for the model to come to its true RMP given the parameters. This is the value that is used before any trace-specific correction is made due to the variability between traces.

- `tot_wait`: a number representing the total amount of time to wait before the stimulus after factoring in the repositioning of the stimulus based on the given experimental data.

- `peak_approx`: stores the approximate amount of time after the stabilization time for which the action potential peak is expected to occur (in ms).
- `pk_dur_est`: stores the expected duration of the depolarization/repolarization phase (in ms).
- `AP_max`: stores the maximum value of the experimental data (in mV).
- `AP_min`: stores the minimum value of the experimental data (in ms).
- `AP_height`: stores the difference between the maximum and minimum values of the experimental data (in mV).
- `RMP`: a numeric value that represents the resting membrane potential. This is calculated by first taking the minimum of the action potential before the peak, then adding the average of the maximum and minimum values of a short section ($200\mu s$) of the data after the repolarization phase. The minimum before the peak is chosen to indicate that the membrane is at its resting membrane potential before it depolarizes, and the addition of the average of a small section of data is used to estimate the noise in the data. The section used for noise analysis is after the repolarization phase because the action potential data was cleaned to begin as the depolarization phase is beginning, so a noise analysis in this region would be unreliable.
- `V_init`: a number representing the initial voltage estimate. This value is set to the RMP once it is calculated
- `Vs`: a list of voltage values (in mV) that represent the model's predicted action potential given the initial conditions and parameters.

- `AP_val`: a vector that represents the trace data with artificial data included before and after the action potential to help with sizing an optimization. The artificial data added is the same value as the RMP.

The `ActionPotential` object has a handful of core member methods that are defined as follows:

- `initialize`: this function is called whenever an action potential object is created and the above member variables are initialized as described.

- `generate_trace`: this function works to compute the model prediction given the initial setup and the current set of parameters. This function has two modes: the first is a streamlined mode that only updates the model's voltage values, and the other saves (and returns) all the information about the state of the model at every time step which is used to plot the data. The latter mode is entered when the user passes `PLOT=TRUE`.

- `dn`, `dm`, and `dh` (`ds` in general): computes the change in `s` (the subunit particle probability open) given the subunit object (`s`), the current state (`s_prob`), and the voltage (`V`).

- `dv`: computes the change in voltage given the magnitude of the stimulus (`I_stim` -- usually 0 unless initiating an action potential) and the present membrane current (`I_m`).

- `gen_trace_ode`: analogous to `generate_trace` but without a plotting option and requires the model parameters to be input to the function. This function uses the `ode` package to solve the model rather than a handwritten version of Euler's method.

- `AP_ode`: handler function for `gen_trace_ode` that is used to hand off to `ode()`.

- `stim_function`: calculates the value of the stimulus (in mV) given a time (`x` -- in ms) as described by A.2.
- `update_subunits`: uses the given model parameters (`params`) to change the local values of the model parameters (`all_params`). This is used by the optimization function to test new values for the model parameters.
- `update_model`: updates the model parameters and updates `Vs`.
- `configure_stim`: functions to update the member variables for the stimulus (`stim_d`, `stim_dim`, `stim_h`, and `tot_wait`). This function is used by the foot optimizer.
- `get_trace_data`: returns the vector containing `Vs`.

```
# Libraries needed for ActionPotential
library(methods)
library(deSolve)
library(compiler)
source("Subunit.R")

# Action Potential object declaration
ActionPotential <- setRefClass("ActionPotential",
  fields = list(
    # Initial Data
    all_params="vector",trace_data="vector",dt="numeric",
    AP_name="character",

    # Sub-unit Declarations
    n="Subunit", m="Subunit",h="Subunit",
    K="Channel", Na="Channel", Leak="Channel",

    # Stimulus Values
    stim_d="numeric", stim_h="numeric", stim_dim="numeric",
    stim_A="numeric",

    # Action Potential Values
    time="numeric", stabil_time="numeric", tot_wait="numeric",
    t="vector", peak_approx="numeric", pk_dur_est="numeric",
    AP_max="numeric", AP_min="numeric", AP_height="numeric",
    RMP="numeric", V_init="numeric", Vs="vector", AP_val="vector"
  )
)
```

```
# Core methods for ActionPotential
ActionPotential$methods(
  # Set up action potential class
  initialize = function(in_all_params, in_trace_data,
                        in_time_data, name="Nameless") {
    # Set ActionPotential name
    AP_name <<- name

    # Default values
    time <<- 30
    stabil_time <<- 10

    # Grab data from input
    all_params <<- in_all_params
    dt <<- round((in_time_data[2]-in_time_data[1]),3)

    # Setup Subunits
    n <<- new("Subunit")
    m <<- new("Subunit")
    h <<- new("Subunit")
    K <<- K_orig
    Na <<- Na_orig
    Leak <<- Leak_orig
    update_subunits(all_params)

    # Stimulus variables
    stim_d <<- 0.64
    stim_h <<- 54.874
    stim_dim <<- 2
    stim_A <<- integrate(.self$stim_function, lower=0,
                         upper=stim_d)$value
    tot_wait <<- stabil_time

    # Approximate information about experimental traces
    peak_approx <<- 2 # Approximate time of peak after stimulus
    pk_dur_est <<- 5 # Approximate duration of the peak

    # Prepare action potential
    trace_data <<- in_trace_data
    AP_max <<- max(trace_data)
    AP_min <<- min(trace_data)
    AP_height <<- AP_max-AP_min
    RMP <<- min(trace_data[1:(peak_approx/dt)]) +
      (max(trace_data[((pk_dur_est+1)/dt):((pk_dur_est+1.2)/dt)])
      -min(trace_data[((pk_dur_est+1)/dt):((pk_dur_est+1.2)/dt)])
      )/2
    V_init <<- RMP
    l = length(trace_data)
    AP_val <<- c(rep(RMP,(stabil_time)/dt+1),
                 trace_data, rep(RMP,(time-(stabil_time))/dt-l))
```

```
  # Set up timing
  t <<- seq(0,time,dt) # time in ms
  Vs <<- rep(V_init, length(t))

  # Preemptively calulate Vs
  generate_trace()
},

# Calculate the trace values
generate_trace = cmpfun(function(PLOT=FALSE) {
  # Initialize empty model data
  Vs_loc = rep(V_init, length(t))

  # Include tracking if plotting
  if (PLOT) {
    Is = rep(0, length(t))
    Istims = rep(0, length(t))
    INas = rep(0, length(t))
    IKs = rep(0, length(t))
    ILeaks = rep(0, length(t))
    Istims = rep(0, length(t))
    ns = rep(infty(n, V_init), length(t))
    ms = rep(infty(m, V_init), length(t))
    hs = rep(infty(h, V_init), length(t))
  }

  # Keep track of iterations
  i = 2

  # Initialize sub-unit conditions
  n_t = infty(n, V_init)
  m_t = infty(m, V_init)
  h_t = infty(h, V_init)
  V_m = V_init
  for (timestep in tail(t,-1)) {
    # Inject pulse after stabilization delay
    if (timestep > tot_wait && timestep < (tot_wait + stim_d)){
      I_stim = stim_function(timestep-tot_wait)
    } else {
      I_stim = 0
    }

    # Compute time step
    n_t = n_t + dt*dn(n, n_t, V_m)
    m_t = m_t + dt*dm(m, m_t, V_m)
    h_t = h_t + dt*dh(h, h_t, V_m)
    I_K = K@g*n_t^4*(V_m-K@E)
    I_Na = Na@g*m_t^3*h_t*(V_m-Na@E)
    I_Leak = Leak@g*(V_m-RMP)
    I_m = I_K + I_Na + I_Leak
    V_m = V_m + dt*dV(I_m, I_stim)
```

```
    # Log time step
    Vs_loc[i] = V_m
    if (PLOT) {
      Is[i] = I_m
      Istims[i] = I_stim
      INas[i] = I_Na
      IKs[i] = I_K
      ILeaks[i] = I_Leak
      Istims[i] = I_stim
      ns[i] = n_t
      ms[i] = m_t
      hs[i] = h_t
    }
    i = i+1
  }

  # Return plotting information if requested
  if (PLOT) {
    return(list(Vs=Vs_loc, Is=Is, INas=INas, IKs=IKs,
                ILeaks=ILeaks, Istims=Istims,
                ns=ns, ms=ms, hs=hs))
  }

  # Update Vs
  Vs <<- Vs_loc
}),

# Define derivative functions
dn = cmpfun(function(n, n_prob, V)
  return(n@alpha(V)*(1-n_prob)-n@beta(V)*n_prob)),
dm = cmpfun(function(m, m_prob, V)
  return(m@alpha(V)*(1-m_prob)-m@beta(V)*m_prob)),
dh = cmpfun(function(h, h_prob, V)
  return(h@alpha(V)*(1-h_prob)-h@beta(V)*h_prob)),
dV = cmpfun(function(I_m, I_stim)
  return(I_stim-I_m)),

# Function to generate trace using ode method
gen_trace_ode = cmpfun(function(params=all_params) {
  trace_soln = ode(func=.self$AP_ode, parms=params,
                   y=c(V=V_init, n=infty(n, V_init),
                       m=infty(m, V_init), h=infty(h, V_init)),
                   times=t, method="ode23")
  Vs <<- unlist(trace_soln[,2])
}),

# Function of time, t, with in_vect = c(V,n,m,h), and params
AP_ode = cmpfun(function(t, in_vect, params) {
  with(as.list(c(in_vect,params)), {
    n_alpha = ifelse(V<N_2, 0, (N_1*V-N_1*N_2))
    n_beta = exp((V+N_7)/N_6)
    m_alpha = ifelse(V<M_2, 0, (M_1*V-M_1*M_2))
```

```
    m_beta = exp((V+M_7)/M_6)
    h_alpha = exp((V+H_6)/H_3)
    h_beta = 1/(1+exp((V+H_4)/H_5))

    dn = n_alpha*(1-n)-n_beta*n
    dm = m_alpha*(1-m)-m_beta*m
    dh = h_alpha*(1-h)-h_beta*h

    I_K = K@g*n^4*(V-K@E)
    I_Na = Na@g*m^3*h*(V-Na@E)
    I_Leak = Leak@g*(V-RMP)
    I_m = I_K + I_Na + I_Leak
    I_stim = ifelse( t > tot_wait && t < (tot_wait + stim_d),
                     stim_function(t-tot_wait), 0)
    dV = I_stim-I_m

    list(c(dV, dn, dm, dh))
  })
}),

# Function for stimulus shape
stim_function = cmpfun(function(x) {
  return(stim_h*(x/stim_d)^stim_dim)
}),

# Update the alpha/beta functions with new parameters
update_subunits = cmpfun(function(params) {
  all_params <<- params
  with(as.list(all_params), {
    n@alpha <<- function(V) ifelse(V<N_2, 0, (N_1*V-N_1*N_2)) #N_1
 *(V+N_2)/(exp((V+N_2)/N_3)-1)
    n@beta <<- function(V) exp((V+N_7)/N_6)
    m@alpha <<- function(V) ifelse(V<M_2, 0, (M_1*V-M_1*M_2)) #M_1
 *(V+M_2)/(exp((V+M_2)/M_3)-1)
    m@beta <<- function(V) exp((V+M_7)/M_6)
    h@alpha <<- function(V) exp((V+H_6)/H_3)
    h@beta <<- function(V) 1/(1+exp((V+H_4)/H_5))
    K@g <<- g_K; Na@g <<- g_Na; Leak@g <<- g_Leak
  })
}),

# Update the model with new parameters
update_model = cmpfun(function(params) {
  update_subunits(params)
  generate_trace() # can use gen_trace_ode()
}),

# Update the values for the stimulus
configure_stim = function(stim_d, stim_dim, stim_h, tot_wait) {
  stim_d <<- stim_d
  stim_dim <<- stim_dim
  stim_h <<- stim_h
```

```
    tot_wait <<- tot_wait
  },

  # Return the trace_data
  get_trace_data = function() {
    return(Vs)
  }
)

# Source Supporting method files of ActionPotential
source("ActionPotentialShow.R")
source("ActionPotentialOpt.R")
source("ActionPotentialStat.R")
```

Listing 2: Action Potential Model in R

I.2.2 Optimization Methods

This file holds the `ActionPotential` methods for optimization:

- `optimize`: the core optimization function for the `ActionPotential` object. This function takes, as an argument, a vector or parameters that should be used in the optimization process (`opt_params`), and the rest are held constant. This optimization is done as described in Section 3.1.2 where the reference data being optimized against is `AP_data`.

- `nudge_optimize`: implements the multi-start method whereby multiple (specifically `num_fits`), similar parameter sets are chosen at random (by Sobol Sequences -- Appendix D), within a chosen `range`. The `opt_params` indicates which model parameters should be optimized.

- `optimize_split`: a different method of optimization that separates the parameter set into sodium- and potassium-related parameters and only optimized those sets of parameters on the portion of the action potential for which is expected those molecules have the most influence (see Section 1.3). This method was not used in the collection of the data presented here.

- `find_foot`: optimizes the variables that are involved with creating the shape of the stimulus (the feature on the action potential trace that arises from the action of the stimulus is sometimes referred to as the foot of the action potential).

- `optim_standard`: a wrapper for the `optim` function with standardized settings across all optimization routines. `fcn` is the objective function being passed and `input` is used in the `parscale` option to normalize the parameter space.

```
# Optimization methods for ActionPotential
ActionPotential$methods(
```

```
# Optimize action potential against data
optimize = cmpfun(function(opt_params_in=all_params) {
  peak_dur = pk_dur_est
  setup_dur = 5 # Used to "calm down" beginning of AP
  Ind = (t>(tot_wait-setup_dur) & t<(tot_wait)) |
    (t>(tot_wait+peak_dur) & t<(tot_wait+length(trace_data)))
  Ind_pk = t>(tot_wait) & t<(tot_wait+peak_dur)

  # Save original paramter set
  orig_param = all_params

  # Parition the paramter set
  opt_params = all_params[names(all_params) %in%
                            names(opt_params_in)] #safeguard
  non_opt_params = all_params[! names(all_params) %in%
                                names(opt_params_in)]

  # Value to return when a optim condition is unwanted
  obj_reject = Inf

  # Penalty check for unresonably sized parameters
  check_penalty <- function(params) {
    max_up = 5 #number represented as a percent
    max_down = 0.1
    par_norm = params/orig_param[names(orig_param) %in%
                                   names(params)]
    return(sum((par_norm < max_down) | (par_norm > max_up)))
  }

  # Objective function
  objective <- function(params) {
    if(check_penalty(params)) {
      return(obj_reject)
    }
    update_model(c(params,non_opt_params))
    # weighted action potential on peak
    score = 5*sum((Vs[Ind_pk]-AP_val[Ind_pk])^2) +
      sum((Vs[Ind]-AP_val[Ind])^2)
    return(ifelse(is.na(score), obj_reject, score))
  }
  out_par = optim_standard(objective, opt_params)

  # Add back in paramters that were not optimized
  out_par$par = c(out_par$par,non_opt_params)
  update_subunits(out_par$par)
  return(out_par)
}),

# Multistart implementation
nudge_optimize = cmpfun(function(opt_params=all_params,
                                 num_fits=1, range=0.1) {
  #could change to "wander" with better fits
```

```
  time_data_loc = c(0,dt)

  init_par = as.data.frame(
    sobolSequence.points(length(opt_params),
                         count=num_fits)*(2*range) + (1-range))
  colnames(init_par) = names(opt_params)
  for(r in 1:nrow(init_par)) {
    init_par[r,] = init_par[r,] * opt_params
  }
  # Add in fixed parameters
  non_opt_params = all_params[! names(all_params) %in%
                                names(opt_params)]
  if(length(non_opt_params) > 0) {
    for(i in 1:length(non_opt_params)) {
      init_par = cbind(init_par, rep(non_opt_params[i],
                                     nrow(init_par)))
      colnames(init_par)[ncol(init_par)] =
        names(non_opt_params[i])
    }
  }

  # Optimize a number of paramter sets
  tmp_data = foreach(i = 1:num_fits, .combine = rbind,
                     .errorhandling = 'remove') %do% {
    cur_pars = unlist(init_par[i,])
    cur_opt_pars = cur_pars[names(cur_pars) %in%
                              names(opt_params)]
    AP_nudge = ActionPotential(cur_pars, trace_data,
                               time_data_loc)
    out_par = AP_nudge$optimize(cur_opt_pars)

    data.frame(t(sapply(out_par$par,c)), values=out_par$value,
               convergence=out_par$convergence)
  }

  # Remove "unruly" parameter sets and return the best
  tmp_data = subset(tmp_data, convergence != 10)
  data = (tmp_data[order(tmp_data$values),])
  data = data[,!(names(tmp_data)%in%
                   c("values","convergence"))]
  update_model(unlist(data[1,]))
  return(optimize(opt_params))
}),

# Optimization routine that splits the trace into multiple
#   sections to optimize independently
optimize_split = cmpfun(function() {
  peak_dur = pk_dur_est
  setup_dur = 5 # Used to "calm down" beginning of AP
  Ind_K = t>(tot_wait-setup_dur) & t<(tot_wait) |
    t>(tot_wait+(peak_approx*0.9)) &
    t<(tot_wait+length(trace_data))
```

```
Ind_Na = t>(tot_wait) & t<(tot_wait+(peak_approx*1.1))
Ind_all = t>(tot_wait-setup_dur) &
  t<(tot_wait+length(trace_data))

# Gather initial parameters
orig_param = all_params
opt_params_K = all_params[names(all_params) %in%
  c('N_1','N_2','N_6','N_7','g_K')]
opt_params_Na = all_params[names(all_params) %in%
  c('M_1','M_2','M_6','M_7','H_3','H_4','H_5','H_6','g_Na')]
opt_params_other = all_params[names(all_params) %in%
  c('g_Leak')]

# Penalty check for unresonably sized parameters
check_penalty <- function(params) {
  max_dev = 0.90 #number represented as a percent
  par_norm = params/orig_param[names(orig_param) %in%
                               names(params)]
  return(sum((par_norm < (1-max_dev)) |
               (par_norm > (1+max_dev))))
}

# Objective functions
objective_all <- function(opt_params) {
  if(check_penalty(opt_params)) {
    return(obj_reject)
  }
  non_opt_params = all_params[! names(all_params) %in%
                              names(opt_params)]
  update_model(c(opt_params, non_opt_params))
  score = sum((Vs[Ind_all]-AP_val[Ind_all])^2)
  return(ifelse(is.na(score), obj_reject, score))
}
objective_K <- function(K_params) {
  if(check_penalty(K_params)) {
    return(obj_reject)
  }
  update_model(c(K_params, opt_params_Na, opt_params_other))
  score = sum((Vs[Ind_K]-AP_val[Ind_K])^2)
  return(ifelse(is.na(score), obj_reject, score))
}
objective_Na <- function(Na_params) {
  if(check_penalty(Na_params)) {
    return(obj_reject)
  }
  update_model(c(Na_params, opt_params_K, opt_params_other))
  score = sum((Vs[Ind_Na]-AP_val[Ind_Na])^2)
  return(ifelse(is.na(score), obj_reject, score))
}

# Rejection return value
orig_score = sum((Vs[Ind_all]-AP_val[Ind_all])^2)
```

```
  obj_reject = Inf #abs(orig_score + 10) * 1000000

  # Optimize the Na and K paramters:
  # Start with a baseline with K
  out_par_K = optim_standard(objective_K, opt_params_K)
  opt_params_K = out_par_K$par
  update_model(c(opt_params_K, opt_params_Na,
                 opt_params_other))

  # Cycle though Na and K
  for(i in 1:15) {
    out_par_Na = optim_standard(objective_Na, opt_params_Na)
    opt_params_Na = out_par_Na$par
    update_model(c(opt_params_K, opt_params_Na,
                   opt_params_other))

    out_par_K = optim_standard(objective_K, opt_params_K)
    opt_params_K = out_par_K$par
    update_model(c(opt_params_K, opt_params_Na,
                   opt_params_other))
  }

  # Finally find a fit over entire trace to show feasibility
  all_params_Na = all_params[names(all_params) %in%
                     c('M_1','M_2','M_6','M_7','g_Na')]
  all_params_not_Na = all_params[! names(all_params) %in%
                     names(all_params_Na)]
  out_par = nudge_optimize(all_params_not_Na, 15, 0.15)
  out_par = optimize()

  return(out_par)
}),

# Find the foot of the action potential
find_foot = cmpfun(function() {
  # Objective function for the foot finder
  objective = function(foot_params) {
    with(as.list(foot_params), {
      if(t_0 < 0 | t_0 > peak_approx | stim_dim < 1 |
         stim_d > peak_approx) return(obj_reject)
      configure_stim(stim_d, stim_dim,
              ((stim_dim+1)*stim_A)/stim_d, stabil_time + t_0)

      # Create an array of integration values to optimize
      #   against
      Ind_stim = t>(tot_wait) & t<(tot_wait+stim_d)
      len_stim = length(AP_val[Ind_stim])
      stim_int = rep(0, len_stim)
      for(i in 1:len_stim) {
        stim_int[i] = integrate(.self$stim_function,
                                lower=0, upper=(i*dt))$value + RMP
      }
```

```
      # Compare the trace to the integral values
      return(stim_h*sum(((stim_int -
              AP_val[Ind_stim])/AP_val[Ind_stim])^2)/len_stim)
    })
  }

  # Get a baseline to use to reject senseless parameters
  obj_reject = (objective(c(t_0=0, stim_d=stim_d,
                            stim_dim=stim_dim)) + 10) * 1000

  # Scan start times to coax convergence
  num_fits = 10
  tmp_data = foreach(i = 1:num_fits, .combine = rbind,
                     .errorhandling = 'remove') %do% {
    out_par = tryCatch(
      {
        optim(fn=objective, par=c(t_0=(i/num_fits),
                            stim_d=stim_d, stim_dim=stim_dim),
              method="Nelder-Mead", control=list(reltol=10^-8,
                                                 maxit=1000))
      },
      error=function(e) {
        stop(paste("Error with optim while finding the foot.\n",e)
 )
      }
    )

    data.frame(t(sapply(out_par$par,c)), values=out_par$value,
               dur=stim_d, dim=stim_dim, ht=stim_h,
               wait=tot_wait)
  }
  tmp_data = (tmp_data[order(tmp_data$values),])
  configure_stim(unlist(tmp_data[1,]["dur"]),
                 unlist(tmp_data[1,]["dim"]),
                 unlist(tmp_data[1,]["ht"]),
                 unlist(tmp_data[1,]["wait"]))
  data = tmp_data[,!(names(tmp_data) %in%
                   c("values", "dur", "dim", "ht", "wait"))]
  return(unlist(data[1,]))
}),

optim_standard = cmpfun(function(fcn, input) {
  opt_set = tryCatch(
    {
      optim(fn=fcn, par=input, method="Nelder-Mead",
            control=list(parscale=input, reltol=10^-12,
                         maxit=1000000))
    },
    error=function(e) {
      stop(paste(e, AP_name))
    }
```

```
    )
    return(opt_set)
  })
)
```

Listing 3: Optimization Methods in R

I.2.3 Visualization Methods

This file defines the methods used in the visualization of the `ActionPotential` object:

- `display_action_potential`: plots the action potential generated by the model parameters as well as the stored experimental data, the stimulus curve, and a graph of the net ion current.
- `display_current`: plots the decomposition of the sodium, potassium, and leak currents alongside the net ion current.
- `display_conductance`: plots the percent of ion channels that are in their permissive state for n, m, and h during the course of the action potential.
- `display_alpha_beta`: plots the α and β functions as described by Section 2.2.
- `display_subunits`: plots a graph like the one shown in Figure 10 where n, m, and h are being graphed while stepping the voltage up and then back down.
- `display_taus`: plots the value of tau for n, m, and h over a range of voltages.
- `display_infinities`: plots the values of the steady-state solutions for n, m, and h over a range of voltages.
- `display_all_plots`: shows all of the plots mentioned in this section so far.
- `plot_param_range`: plots a variety of APs where a single parameter is varied. This is repeated for each parameter. This produced the figures for Appendix H.

```
# Plotting methods for ActionPotential
ActionPotential$methods(
  # Method for displaying the action potential
  display_action_potential = function() {
    update_model(all_params)
    with (generate_trace(PLOT=TRUE), {
      plot(t, Vs,
```

```
          main="Action Potential",
          xlab="Time (ms)",
          ylab="Voltage (mV)",
          type='l',
          ylim=c(-100,100))
    lines(t, Is, col="gray")
    lines(t, Istims, col="blue")
    abline(h=c(30,-80), col="black", lty=3)

    lines(x = t, y=AP_val, col="purple")
    legend(0, 100,
           legend=c("AP", expression(I[m]), "Pulse",
                    "Exp data"),
           col=c('black', 'gray', 'blue', 'purple'),
           lty=1)
  })
},

# Plot the decomposed currents for the model
display_currents = function() {
  with(generate_trace(PLOT=TRUE), {
    y_min = min(c(Is, IKs, INas, ILeaks, Istims))
    y_max = max(c(Is, IKs, INas, ILeaks, Istims))
    plot(t, Vs,
          main="Current Decomposition",
          xlab="Time (ms)",
          ylab="Current (nA)",
          type='l',
          ylim=c(y_min,y_max))
    lines(t, Is, col="gray")
    lines(t, IKs, col="blue")
    lines(t, INas, col="springgreen4")
    lines(t, ILeaks, col="pink")
    lines(t, Istims, col="aquamarine3")

    legend(0, y_max,
           legend=c("AP", expression(I[m]), expression(I[K]),
              expression(I[Na]), expression(I[Leak]), "Pulse"),
           col=c('black', 'gray', 'blue', 'springgreen4',
                 'pink', 'aquamarine3'),
           lty=1)
  })
},

# Plot the decomposed sub-unit conductance
display_conductance = function() {
  with(generate_trace(PLOT=TRUE), {
    plot(t, ns,
          main="Conductance Decomposition",
          xlab="Time (ms)",
          ylab="Particle Proability Active",
          type='l',
```

```
          col="blue",
          ylim=c(0,1))
    lines(t, ms, col="springgreen4")
    lines(t, hs, col="red")

    legend(0, 1,
           legend=c("n", "m", "h"),
           col=c('blue', 'springgreen4', 'red'),
           lty=1)
    abline(v=tot_wait, col="black", lty=3)
  })
},

# Test alpha and beta
display_alpha_beta = function() {
  V = seq(-150,100,1)
  plot_max = 2
  plot(V, n@alpha(V),
       main="Alpha: open; Beta: closed",
       xlab="Voltage (mV)",
       ylab="Rate constant (1/mS)",
       type='l',
       col="blue",
       lty=1,
       ylim=c(0,plot_max))
  lines(V, n@beta(V), col="blue", lty=2)
  lines(V, m@alpha(V), col="springgreen4", lty=1)
  lines(V, m@beta(V), col="springgreen4", lty=2)
  lines(V, h@alpha(V), col="red", lty=1)
  lines(V, h@beta(V), col="red", lty=2)
  legend(50, plot_max,
         legend=c(expression(alpha[n]), expression(beta[n]),
                  expression(alpha[m]), expression(beta[m]),
                  expression(alpha[h]), expression(beta[h])),
         col=c('blue', 'blue', 'springgreen4', "springgreen4",
               "red", "red"),
         lty=c(1,2,1,2,1,2))
},

# Testing Subunit functions
display_subunits = function() {
  #V_init = -80
  ns = c(infty(n,V_init))
  ms = c(infty(m,V_init))
  hs = c(infty(h,V_init))
  for (timestep in tail(t,-1)) {
    n_t = tail(ns, 1)
    m_t = tail(ms, 1)
    h_t = tail(hs, 1)
    V = 0
    if (timestep>(time/2))
      V = -80
```

```
    ns = append(ns, n_t + dt*dn(n, n_t, V))
    ms = append(ms, m_t + dt*dm(m, m_t, V))
    hs = append(hs, h_t + dt*dh(h, h_t, V))
  }
  plot(t, ns,
       main="Subunit Conductance",
       xlab="Time (ms)",
       ylab="Particle Proability Active",
       ylim=c(0,1),
       type='l',
       col="blue",
       lty=1)
  lines(t, ms, col="springgreen4", lty=1)
  lines(t, hs, col="red", lty=1)
  lines(t, ns^4, col="blue", lty=2)
  lines(t, ms^3*hs, col="springgreen4", lty=2)
  legend(time-5, 1,
         legend=c('n', 'm', 'h', "K", "Na"),
         col=c('blue', 'springgreen4', 'red', "blue",
               "springgreen4"),
         lty=c(1,1,1,2,2))
},

# Generate plots for Taus
display_taus = function() {
  V = seq(-100,100,1)
  plot_max = 10
  plot(V, tau(n,V),
       main="Time Constants",
       xlab="Voltage (mV)",
       ylab="Tau (ms)",
       type='l',
       ylim=c(0,plot_max),
       col="blue")
  lines(V, tau(m,V), col="springgreen4")
  lines(V, tau(h,V), col="red")
  legend(50, plot_max,
         legend=c('n', 'm', 'h'),
         col=c('blue', 'springgreen4', 'red'),
         lty=1)
},

# Generate plots for Infinities
display_infinities = function() {
  V = seq(-150,100,1)
  plot(V, infty(n,V),
       main="Steady-Sate Solutions",
       xlab="Voltage (mV)",
       ylab="Particle Probabiliy Active",
       type='l',
       col="blue",
       ylim=c(0,1))
```

```
    lines(V, infty(m,V), col="springgreen4")
    lines(V, infty(h,V), col="red")
    legend(50, 0.9,
           legend=c('n', 'm', 'h'),
           col=c('blue', 'springgreen4', 'red'),
           lty=1)
  },

  display_all_plots = function() {
    display_action_potential()
    display_currents()
    display_conductance()
    display_subunits()
    display_alpha_beta()
    display_taus()
    display_infinities()
  },

  # Plot the AP simulation across a range of values for each
  #   parameter
  plot_param_range = function() {
    for(i in 1:(length(all_params))) {
      update_model(all_params)
      par_tmp = all_params[i]
      max_seq = 2*par_tmp
      plot(t, Vs,
        main=paste(names(par_tmp), ":", par_tmp),
        xlab="Time (ms)",
        ylab="Voltage (mV)",
        type='l',
        ylim=c(-100,100),
        col="yellow")
      #lines(t, Istims, col="green")

      legend(20, 75, legend=c(0, max_seq),
             fill = c(rgb(max_seq,0,0, max=max_seq),rgb(0,0,
                                                        max_seq,
                                                        max=max_seq)))

      par = seq(0,max_seq,length.out=100)
      for(val in par) {
        all_params[i] <<- val
        update_model(all_params)
        lines(t, Vs, col=rgb(max_seq-val,0,val, max=max_seq))
      }
      print(par_tmp)
      all_params[i] <<- par_tmp
    }
  }
)
```

Listing 4: Visualization Methods in R

I.2.4 Uniqueness Determination Methods

This file contains the methods for performing uniqueness calculations on the object `ActionPotential`. In reality, the file defines an object called `ActionPotentialStat` that inherits all of the features from `ActionPotential` and adds some new member variables that are only used in the uniqueness determination methods for file management. Those variables are:

- `stat_file`: a string that holds the filename for a file that stores the data from the uniqueness calculations.
- `og_stat_file`: a string that holds the filename for a file that stores the original values of the parameters before optimization (generated via Sobol Sequences: Appendix D).
- `stat_ref _data`: a string that holds the filename for a file that stores the original action potential object used for the uniqueness computations to be used in the later analysis.
- `opt_ref_data`: a string that holds the filename for a file that stores the parameter subset being optimized.

The methods defined for `ActionPotentialStat` are:

- `initialize`: sets the filename values for the member variables defined above.
- `run_statistical_analysis`: follows the methodology given in Section C. The cores `cores` argument sets the number of cores the function will ask the operating system to use, and `opt_params` is the parameter subset being optimized in the analysis. This function is only the computationally expensive portion which includes the thousands of optimizations (running in parallel). Additionally, the

results of the computations are stored in the files as described by the member variables.

- `show _statistical_analysis`: the second part of the preceding function that carries out the analysis (using data that was stored in the files as described).
- `display_stat_APs`: handling visualizing the outputs of the statistical analysis done in the preceding function.

```
# Load required libraries
library(SobolSequence)
library(doParallel)
library(foreach)

# Class for ActionPotential statistics
ActionPotentialStat <- setRefClass("ActionPotentialStat",
                                   contains="ActionPotential",
  fields = list(
    opt_params="vector",
    # Files
    stat_file="character",
    og_stat_file="character",
    stat_ref_data="character",
    opt_ref_data="character"
  )
)

# Statistics methods for ActionPotential
ActionPotentialStat$methods(
  initialize = function(...) {
    stat_file <<- "stat_par"
    og_stat_file <<- "stat_par_og"
    stat_ref_data <<- "stat_AP_ref"
    opt_ref_data <<- "opt_par_ref"
    callSuper(...)
  },

  # Perform the statistical analysis
  run_statistical_analysis = cmpfun(function(
      cores=(detectCores() - 1), opt_params=all_params) {
    num_fits = 5000
    range = 2.5
    center = ifelse(range < 1, 1, range + 0.1)

    # Generate trace data to use for comparison
    generate_trace()
```

```
trace_data_gen = Vs[(stabil_time/dt):
                      (stabil_time/dt+length(trace_data))]
time_data_gen = c(0, dt)

# Save reference data to refer to later
AP_ref = ActionPotential(all_params, trace_data_gen,
                         time_data_gen)
saveRDS(AP_ref, file = paste(output_folder, stat_ref_data,
                             ".Rds", sep=""))
saveRDS(opt_params, file = paste(output_folder, opt_ref_data,
                                 ".Rds", sep=""))

print(paste("Fits: ", num_fits, ". Range: ", range,
            ". Center: ", center, ". Cores: ", cores,
            sep = ""))

# Generate quasi-random parameter sets
init_par = as.data.frame(sobolSequence.points(
  length(opt_params), count=num_fits)*(2*range) +
    (center-range))
colnames(init_par) = names(opt_params)
for(r in 1:nrow(init_par)) {
  init_par[r,] = init_par[r,] * opt_params
}
# Add in fixed parameters
non_opt_params = all_params[! names(all_params) %in%
                              names(opt_params)]
if(length(non_opt_params) > 0) {
  for(i in 1:length(non_opt_params)) {
    init_par = cbind(init_par, rep(non_opt_params[i],
                                   nrow(init_par)))
    colnames(init_par)[ncol(init_par)] =
      names(non_opt_params[i])
  }
}
saveRDS(init_par, file = paste(output_folder, og_stat_file,
                               ".Rds", sep=""))

if (file.exists(paste(stat_file, ".Rds", sep=""))) {
  file.remove(paste(stat_file, ".Rds", sep=""))
}

# Run computation for identifiability analysis
clust = makeCluster(cores)
clusterExport(clust, c("init_par", "opt_params",
                       "trace_data_gen", "time_data_gen"),
              envir=environment())
registerDoParallel(clust)
data_one <- foreach(i = 1:num_fits, .combine = rbind,
                    .errorhandling = 'remove') %dopar% {
  source("ActionPotential.R")
  cur_pars = unlist(init_par[i,])
```

```
    cur_opt_pars = cur_pars[names(cur_pars) %in%
                              names(opt_params)]
    AP_stat = ActionPotential(cur_pars, trace_data_gen,
                              time_data_gen)
    out_par = AP_stat$optimize(cur_opt_pars)

    data.frame(t(sapply(out_par$par,c)), values=out_par$value,
               convergence=out_par$convergence)
  }
  stopCluster(clust)

  # take all values 20% away from the min value
  data_one = (data_one[order(data_one$values),])
  min_val = data_one[,"values"][1]
  new_max_val = min_val*1.2
  max_range = length(which(data_one[,"values"] <= new_max_val))
  data_one = data_one[1:ifelse(max_range<10,10,max_range),]
  data_one = data_one[,!(names(data_one) %in%
                           c("values","convergence"))]

  # Set up data for second analysis
  mult = 10
  for (i in 1:nrow(data_one)) {
    for (j in 1:mult) {
      data_one = rbind(data_one, data_one[i,])
    }
  }
  num_fits_two = nrow(data_one)

  # Run computation for identifiability analysis a second time
  clust = makeCluster(cores)
  clusterExport(clust, c("data_one", "opt_params",
                         "trace_data_gen", "time_data_gen"),
                envir=environment())
  registerDoParallel(clust)
  data <- foreach(i = 1:num_fits_two, .combine = rbind,
                  .errorhandling = 'remove') %dopar% {
    source("ActionPotential.R")
    cur_pars_tmp = unlist(data_one[i,])
    cur_non_opt_pars = cur_pars_tmp[! names(cur_pars_tmp) %in%
                                      names(opt_params)]
    cur_opt_pars = cur_pars_tmp[names(cur_pars_tmp) %in%
                                  names(opt_params)]
    cur_opt_pars = cur_opt_pars * runif(length(cur_opt_pars),
                                        min=0.95, max=1.05)
    cur_pars = c(cur_opt_pars, cur_non_opt_pars)
    AP_stat = ActionPotential(cur_pars, trace_data_gen,
                              time_data_gen)
    out_par = AP_stat$optimize(cur_opt_pars)

    data.frame(t(sapply(out_par$par,c)), values=out_par$value,
               convergence=out_par$convergence)
```

```
  }
  stopCluster(clust)

  # Save output data
  saveRDS(data, file = paste(output_folder, stat_file, ".Rds",
                             sep=""))
  saveRDS(data, file = paste(output_folder, stat_file,
                             num_fits, "-", range, ".Rds",
                             sep=""))
}),

# Display the results of the statistical analysis
show_statistical_analysis = cmpfun(function() {
  tmp_data <- subset(readRDS(file = paste(output_folder,
      stat_file, ".Rds", sep="")), convergence != 10)
  data <- (tmp_data[order(tmp_data$values),])
  # add [1:(nrow(tmp_data)*0.2),] to take top 20%

  # Read reference data used to generate the statistics
  AP_ref = readRDS(file = paste(output_folder, stat_ref_data,
                                ".Rds", sep=""))
  print("Reference Data")
  print(AP_ref$all_params)
  print("Optimized paramters")
  print(readRDS(file = paste(output_folder, opt_ref_data,
                             ".Rds", sep="")))
  AP_ref$display_action_potential()

  # Show the plots generated by optimized stats data
  display_stat_APs(data, AP_ref)

  # Calcualte coefficients of variation
  print(paste("n=", nrow(data), " "))
  for (i in 1:ncol(data)) {
    print( paste("CV for", colnames(data)[i],
                 sd(data[,i])/mean(data[,i])) )
  }

  # Plot pairwise comparisons of parameters
  par(mfrow=c(1,1))
  for(i in 1:ncol(data)){
    for(j in 1:ncol(data)) {
      if(i<j){
        plot(data[,i],data[,j],
             main=paste(colnames(data)[i], "vs",
                        colnames(data)[j]),
             xlab=colnames(data)[i],
             ylab=colnames(data)[j],
             col=colorRampPalette(c("blue","red"))(nrow(data)))
        print(summary(lm(data[,i]~data[,j])))
      }
    }
```

```
    }

    pairs(data)
    return(data)
  }),

  # Show action potentials generated for each optimized parameter
  #   set
  display_stat_APs = function(data, AP_ref) {
    AP_ref$update_model(unlist(data[1,]))
    with(AP_ref$generate_trace(PLOT=TRUE), {
      plot(AP_ref$t, AP_ref$Vs,
           main="All Action Potentials",
           ylab="Voltage (mV)",
           xlab="Time (mS)",
           type='l',
           ylim=c(-100,100))
      abline(h=c(30,-80), col="black", lty=3)
    })
    for(i in 2:nrow(data)) {
      AP_ref$update_model(unlist(data[i,]))
      with(AP_ref$generate_trace(PLOT=TRUE), {
        lines(AP_ref$t, AP_ref$Vs, col=rgb(nrow(data)-i,0,i,
                                           max=nrow(data)))
      })
    }
  }
)
```

Listing 5: Uniqueness Determination Methods in R

I.3 Subunit and Channel Object Declaration

This file contains the object declarations for `Subunit` and `Channel`. The `Subunit` object is used to store the α and β functions for n, m, and h. The declaration of this object in `R` was done using the S3 datatype, so it functions slightly differently than the `ActionPotential` object. However, the class members can be described as:

- `alpha`: a function (stored as a member "variable") that corresponds to the α function described in Section 2.2.
- `beta`: a function (stored as a member "variable") that corresponds to the β function described in Section 2.2.

The `Subunit` object contains two methods:

- `tau`: defined as $\frac{1}{\alpha(V)+\beta(V)}$ that takes a subunit object and the present voltage as parameters. This is the τ_s function defined in Section 2.1.
- `infty`: defined as $\frac{\alpha(V)}{\alpha(V)+\beta(V)}$ that takes a subunit object and the present voltage as parameters. This is the s_∞ function defined in Section 2.1.

Additionally, the `Channel` object contains two member variables:

- `g`: stores the maximum conductance value of the given ion channel (in mS).
- `E`: stores the electromotive potential across the membrane for a given ion.

Lastly, this file defines the `nernst` equation and gives a couple of initial values for the `Channel` objects based on Hodgkin and Huxley (1952a).

```
# Create Channel and subunit classes (V in mV)
setClass("Channel", slots=list(g="numeric", E="numeric"))
setClass("Subunit", slots=list(alpha="function", beta="function"))
setGeneric("tau", function(object, V) standardGeneric("tau"))
setGeneric("infty", function(object, V) standardGeneric("infty"))
```

```
setMethod("tau",
          "Subunit",
          function(object, V) {
            return(1/(object@alpha(V)+object@beta(V)))
          })
setMethod("infty",
          "Subunit",
          function(object, V) {
            return(object@alpha(V)/
                     (object@alpha(V)+object@beta(V)))
          })

# Nernst equation
nernst <- function(z, cons_out, cons_in) {
  # Constants
  R <- 8.3144598 # J/mol*K
  F <- 96.485 # J/mol*mV
  T <- 296.15 # Room temperature in Kelvin
  e_chg = 1.602176634e-19 # Elementary charge in Coulombs
  return((R*T)/(z*F)*log(cons_out/cons_in))
}

# Define sub-units and channels
K_orig <- new("Channel", g=36, E=nernst(1,5.4,143))
Na_orig <- new("Channel", g=120, E=nernst(1,145,9.6))
Cl_orig <- new("Channel", g=0.3, E=nernst(-1,127.7,8.7))
Leak_orig <- new("Channel", g=0.3, E=(nernst(-1,127.7,8.7)+
                                        nernst(2,1.8,3.5)) )
```

Listing 6: Subunit object in R

I.4 Global Variables

This file gives the initial parameter set (as optimized for the TTX-sensitive average), and sets the folders for which data can be found/saved. Furthermore, it sources an example trace for testing purposes.

```
# Initial paramter values
par_0 <- c(N_1=0.004,N_2=-119,N_6=-23,N_7=64.5,
           M_1=0.523,M_2=-69.7,M_6=-17.6,M_7=30.8,
           H_3=-21.1,H_4=68.3,H_5=-3.67,H_6=161.5,
           g_K=36, g_Na=120, g_Leak=0.3)

# Set up paramter subset of optimization parameters
with(as.list(par_0), {
  opt_par_0 <<- c(N_6=N_6,N_7=N_7,N_1=N_1,N_2=N_2,
                  M_6=M_6,M_7=M_7,M_1=M_1,M_2=M_2,
                  H_4=H_4,H_5=H_5,H_6=H_6,H_3=H_3,
                  g_Na=g_Na, g_K=g_K, g_Leak=g_Leak)
})

# Set data folders
data_folder = paste(getwd(), "/Cleaned_Data/", sep="")
output_folder = paste(getwd(), "/Parameter_Files/", sep="")
notebook_folder = paste(getwd(), "/Notebook/", sep="")

# Example Values
AP_data_ex = read.csv(paste(data_folder, "AP_trace.csv", sep=""))
trace_data_ex = AP_data_ex$WT_avg_10ms
time_data_ex = AP_data_ex$Time_ms
```

Listing 7: Global variables in R

J Programs in R Markdown

J.1 Functions on Experimental Traces

This file is the file that was used to gather the experimental data and subsequently optimize a set of parameters against each experimental trace (visualization in Section 1.4). Details for each code chunk are outlined:

- `setup`: gives the required libraries and defines where files produced from this RMD file will be placed.
- `header`: lists which trace files to use and determines the correct number of cores to use for the parallelism
- `functions`: defines the functions that will be used. Those functions are as follows:
  - `plot_trace_file_data`: given the name of a `file` containing action potential trace data, all of the traces are plotted on a single plot.
  - `opt_trace_file`: creates an `ActionPotential` object for each action potential from the given `file` and optimizes a parameter set against each trace separately. It then shows the resulting fit.
  - `opt_trace_file_parallel`: performs the same operation as those found in `opt_trace_file` but uses multi-threading (with `cores` number of cores). This implementation also saves the parameters for each fit to a file.
  - `show_opt_trace`: reads in the parameter values from the file mentioned in the previous function, and displays the result. It uses `file` to determine which action potential data to use. Furthermore, this function also calculates the mean, median, and standard deviation for all traces in `file`, and saves that data to a `csv`.

- `opt`: runs the optimization function if the `RMD` file is asked to do so.
- `main`: begins the computation execution.

```
---
title: "Action Potential - Individual Traces"
output:
  html_document: default
  pdf_document:
    latex_engine: xelatex
    pandoc_args: --listings
date: "`r Sys.Date()`"
params:
  run: FALSE
  cores: 1
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

library(doParallel)
library(foreach)
library(knitr)
source("ActionPotential.R")
source("vars.R")

loc_dir = paste("/Trace/", Sys.Date(), "/", sep="")
output_folder_trc = paste(output_folder, loc_dir, sep="")
dir.create(output_folder_trc)
```

```{r header, echo=FALSE}
trace_files = c("Atratus_WT.csv", "Atratus_P.csv",
                "Atratus_EPN.csv")
#trace_files = c("Atratus_WT.csv")
#trace_files = c("Atratus_P.csv")
#trace_files = c("Atratus_EPN.csv")

num_traces = 1
for (file in trace_files) {
  data = read.csv(paste(data_folder, file, sep=""), skip=3)
  num_traces_cur = (ncol(data)/2)

  if (num_traces_cur > num_traces) {
    num_traces = num_traces_cur
  }
}

num_cores = params$cores
if (num_traces < num_cores) {
```
```

```
  num_cores = num_traces
}
```

```{r functions, echo=FALSE}
# Functions for reading traces
trace_file = "trace_par"

# Plot all of the traces from a given file
plot_trace_file_data <- function(file) {
  data = read.csv(paste(data_folder, file, sep=""), skip=3)
  plot(data[,1], data[,2],
       main="Action Potentials",
       ylab="Voltage (mV)",
       xlab="Time (ms)",
       type='l',
       ylim=c(-100,100),
       xlim=c(0,20))
  labs = c(names(data[2]))
  cols = c(rgb(0,0,0))
  for(i in 2:(ncol(data)/2)) {
    cur_col = rgb(0,0,i, max=(ncol(data)/2))
    lines(data[,(2*i-1)], data[,(2*i)], col=cur_col)
    labs = append(labs, names(data[(2*i)]))
    cols = append(cols, cur_col)
  }
  legend(10, 100,
         legend=labs,
         col=cols,
         lty=1)
}

# Optimize a model to the trace data in a given file
opt_trace_file <- function(file) {
  data = read.csv(paste(data_folder, file, sep=""), skip=3)
  for(i in 1:(ncol(data)/2)) {
    read_trace_data = as.numeric(na.omit(data[,(2*i)]))
    read_time_data = as.numeric(na.omit(data[,(2*(i-1)+1)]))
    AP = ActionPotential(par_0, read_trace_data, read_time_data)
    AP$find_foot()
    AP$optimize()
    AP$display_action_potential()
  }
}

# Optimize a model to the trace data in a given file using
#   parallelism
opt_trace_file_parallel <- function(file,
                                    cores=(detectCores() - 1)) {
  data = read.csv(paste(data_folder, file, sep=""), skip=3)

  clust = makeCluster(cores)
```

```
  clusterExport(clust, c("par_0", "opt_par_0", "data"),
                envir=environment())
  registerDoParallel(clust)
  fits <- foreach(i = 1:(ncol(data)/2),
                  .combine = rbind) %dopar% {
    #for(i in 1:(ncol(data)/2)) {
    source("ActionPotential.R")

    read_trace_data = as.numeric(na.omit(data[,(2*i)]))
    read_time_data = as.numeric(na.omit(data[,(2*(i-1)+1)]))
    AP = ActionPotential(par_0, read_trace_data, read_time_data,
                         paste(file, i))
    AP$find_foot()
    #out_par = AP$optimize_split()
    out_par = AP$nudge_optimize(opt_par_0, 15, 0.25)
    #out_par = AP$optimize(opt_par_0)

    # Added RMP -- Could be a source or error
    data.frame(t(sapply(out_par$par,c)), values=out_par$value,
               convergence=out_par$convergence,
               RMP_val=AP$RMP, dur=AP$stim_d, dim=AP$stim_dim,
               ht=AP$stim_h, wait=AP$tot_wait)
  }
  stopCluster(clust)
  saveRDS(fits, file = paste(output_folder_trc, file, trace_file,
                             ".Rds", sep=""))
}

# Visualize the optimized trace
show_opt_trace <- function(file) {
  data = read.csv(paste(data_folder, file, sep=""), skip=3)
  fits = readRDS(file = paste(output_folder_trc, file, trace_file,
                              ".Rds", sep=""))
  header_names = c()
  to_csv = data.frame()

  for(i in 1:nrow(fits)) {
    read_trace_data = as.numeric(na.omit(data[,(2*i)]))
    read_time_data = as.numeric(na.omit(data[,(2*(i-1)+1)]))
    AP = ActionPotential(unlist(fits[i,]), read_trace_data,
                         read_time_data, names(data[2*i]))
    AP$configure_stim(unlist(fits[i,]["dur"]),
                      unlist(fits[i,]["dim"]),
                      unlist(fits[i,]["ht"]),
                      unlist(fits[i,]["wait"]))
    AP$display_action_potential()
    print(names(data[2*i]))
    print(kable(AP$all_params, caption="List of all parameters"))

    header_names = append(header_names, names(data[2*i]))
    if (i == 1) {
      to_csv = as.data.frame(AP$all_params)
```

```
    } else {
      to_csv = cbind(to_csv, as.data.frame(AP$all_params))
    }
  }

  means = c()
  sds = c()
  medians = c()
  for(i in 1:ncol(fits)){
    means = append(means, mean(fits[,i]))
    sds = append(sds, sd(fits[,i]))
    medians = append(medians, median(fits[,i]))
  }
  mean_df = data.frame(names(fits), means)
  sd_df = data.frame(names(fits), sds)
  median_df = data.frame(names(fits), medians)
  print(kable(mean_df, caption="Means of Parameters"))
  print(kable(median_df, caption="Medians of Parameters"))
  print(kable(sd_df,caption="Standard Deviations of Parameters"))

  names(to_csv) = header_names
  to_csv = cbind(to_csv, means)
  to_csv = cbind(to_csv, medians)
  to_csv = cbind(to_csv, sds)

  write.csv(to_csv, paste(output_folder_trc, file, trace_file,
                          ".csv", sep=""), row.names=TRUE)
}
```

```{r opt, echo=FALSE}
if(params$run) {
  for(file in trace_files) {
    opt_trace_file_parallel(file,cores=num_cores)
  }
}
```

Optimizing and Visualization of Individual Traces

```{r main, results="asis", message=FALSE}
print("Optomized paramters")
print(opt_par_0)
for(file in trace_files) {
    plot_trace_file_data(file)
    show_opt_trace(file)
}
```

Listing 8: Functions on Experimental Traces in RMD

J.2 Functions for Group Analysis

This file contains the group analysis execution as described in Section 4. Details for each code chunk are outlined:

- `setup`: includes necessary libraries.
- `header`: declares what the optimization subset of parameters will be, defines which trace files to use, and describes where data produced by the code should be stored.
- `functions`: defines the functions that will be used. Those functions are as follows:
  - `get_group_pars`: takes the point wise average of each trace in `file`, then creates an `ActionPotential` object with the average data and `pars` as the parameter set. It then optimizes the parameters against the calculated trace using `free_pars` as the optimization parameters. The resulting `ActionPotential` object with optimized parameters is returned.
  - `gen_exp_traces`: returns a `data.frame` of `num_fits` traces where each row is quasi-experimental trace data. The `pars`, `group_trace`, and `group_time` arguments are used to create an `ActionPotential` object. The `fixed_pars` and `free_` arguments are used to generate `num_fits` sets of parameter values via Sobol Sequences (Appendix D).
  - `gen_fixed_set`: returns `num_fits` sets of parameter sets where `fixed_pars` are randomized via Sobol Sequences around the `pars` parameter set.
- `main`: performs the calculated needed to do the analysis described in Section 4, and saves the tables of values to separate files.
- `linear`: performs the analysis described in Section 4.2.

- `group`: performs the analysis described in Appendix F.

````
---
title: "Action Potential - Group Analysis"
output:
  html_document: default
  pdf_document:
    latex_engine: xelatex
    pandoc_args: --listings
date: "`r Sys.Date()`"
params:
  run: FALSE
  cores: 1
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

library(nlme)
library(doParallel)
library(foreach)
library(SobolSequence)
library(knitr)
source("ActionPotential.R")
source("vars.R")
```

```{r header, echo=FALSE}
trace_files = c("Atratus_WT.csv", "Atratus_P.csv",
                "Atratus_EPN.csv")
#trace_files = c("Atratus_WT.csv", "Atratus_EPN.csv")
group_file = "group_par"
loc_dir = paste("/Group/", "2023-05-01All/", sep="")
#loc_dir = paste("/Group/", Sys.Date(), "All/", sep="")
output_folder_grp = paste(output_folder, loc_dir, sep="")
dir.create(output_folder_grp)

with(as.list(par_0), {
  opt_par_group <<- c(N_6=N_6,N_7=N_7,N_1=N_1,N_2=N_2,
                  M_6=M_6,M_7=M_7,M_1=M_1,M_2=M_2,
                  #H_4=H_4,H_5=H_5,H_6=H_6,H_3=H_3,
                  g_Na=g_Na, g_Leak=g_Leak, g_K=g_K
                  )
})
```

```{r functions, echo=FALSE}
get_group_pars <- function(pars, fixed_pars, free_pars, file) {
  data = read.csv(paste(data_folder, file, sep=""), skip=3)
  N = (ncol(data)/2)
  dt = 0
````
````

```
  time = Inf
  for(i in 1:N) {
    Time = na.omit(data[,(2*i-1)])
    dt_loc = round((Time[2]-Time[1]),3)
    time_loc = Time[length(Time)]
    if (dt_loc > dt) {
      dt = dt_loc
    }
    if (time_loc < time) {
      time = time_loc
    }
  }

  t = seq(0, time, dt)
  Vs = c()
  for(timestep in t) {
    v_tmp = c()
    for (i in 1:N) {
      Time = round(data[,(2*i-1)], 3)
      Volts = data[,(2*i)]
      v_tmp = append(v_tmp, Volts[which(Time==timestep)])
    }
    Vs = append(Vs, mean(v_tmp))
  }
  Vs = na.omit(Vs)

  AP_tmp = ActionPotential(pars, Vs, t, "Group Average")
  AP_tmp$find_foot()
  AP_tmp$optimize(free_pars)
  AP_tmp$display_action_potential()
  return(AP_tmp)
}

gen_exp_traces <- function(pars, fixed_pars, free_pars,
                           group_trace, group_time, num_fits) {
  #num_fits = 5
  range = 0.05

  # Create uniformly distributed set of free paramters
  init_par = as.data.frame(sobolSequence.points(length(free_pars),
                          count=num_fits)*(2*range) + (1-range))
  colnames(init_par) = names(free_pars)
  for(r in 1:nrow(init_par)) {
    init_par[r,] = init_par[r,] * free_pars
  }
  # Add in fixed paramters
  if(length(fixed_pars) > 0) {
    for(i in 1:length(fixed_pars)) {
      init_par = cbind(init_par,
                       rep(fixed_pars[i], nrow(init_par)))
      colnames(init_par)[ncol(init_par)] = names(fixed_pars[i])
    }
```

```
  }

  # Get varied experimental traces
  varied_traces = foreach(i = 1:num_fits,
                          .combine = rbind) %do% {
    tmp_par = unlist(init_par[i,])
    AP_group = ActionPotential(tmp_par, group_trace, group_time)
    AP_group$display_action_potential()
    Vs_gen = AP_group$Vs

    # Trim atrificially implied data generated from trace
    st = AP_group$stabil_time
    dt = AP_group$dt
    l = length(AP_group$trace_data)
    Vs_gen[(st/dt):(st/dt+l)]
  }
  return(varied_traces)
}

get_fixed_set <- function(pars, fixed_pars, free_pars,
                          num_fits) {
  #num_fits = 10
  range = 0.25

  init_par = as.data.frame(sobolSequence.points(length(fixed_pars),
                          count=num_fits)*(2*range) + (1-range))
  colnames(init_par) = names(fixed_pars)
  for(r in 1:nrow(init_par)) {
    init_par[r,] = init_par[r,] * fixed_pars
  }
  # Add in free paramters
  if(length(free_pars) > 0) {
    for(i in 1:length(free_pars)) {
      init_par = cbind(init_par, rep(free_pars[i],
                                     nrow(init_par)))
      colnames(init_par)[ncol(init_par)] = names(free_pars[i])
    }
  }

  return(init_par)
}

```

```{r main, echo=FALSE, message=FALSE}
num_tables = 25
num_gen_trace = 10
num_nudge = 10

# Start with initail parameters
free = par_0[names(par_0) %in% names(opt_par_group)]
fixed = par_0[! names(par_0) %in% names(opt_par_group)]
```

```
if (params$run) {
  # Generate groups (as type ActionPotential for data management)
  num_groups = length(trace_files)
  groups = foreach(i = 1:num_groups, .combine = c) %do% {
    print(paste("AP ave for group", i))
    AP = get_group_pars(par_0, fixed, free, trace_files[i])
    message(paste("Group", i, "Created"))
    saveRDS(AP, file = paste(output_folder_grp, group_file, i,
                             "groupAP.Rds", sep=""))
    AP
  }
  saveRDS(groups, file = paste(output_folder_grp, group_file,
                               "AllgroupAPs.Rds", sep=""))
  message("Groups Created")

  # Generate quasi-experimental data (as a list of tables with
  #   each row as a trace)
  #   eg.: traces_by_group[[group]][trace,]
  traces_by_group = vector("list", length(groups))
  traces_by_group_comb = foreach(i = 1:length(groups),
                                 .combine = list) %do% {
    AP_tmp = groups[[i]]

    # Trim atrificially implied data generated from trace
    trace_Vs = AP_tmp$Vs
    st = AP_tmp$stabil_time
    dt = AP_tmp$dt
    l = length(AP_tmp$trace_data)
    trace_Vs = trace_Vs[(st/dt):(st/dt+l)]
    print(paste("Generated APs for group", i))
    traces = gen_exp_traces(AP_tmp$all_params, fixed, free,
                            trace_Vs, AP_tmp$t, num_gen_trace)
    saveRDS(traces, file = paste(output_folder_grp, group_file,
                                 i, "groupTraces.Rds", sep=""))
    message(paste("Traces for group", i, "Created"))
    traces_by_group[[i]] = traces
    traces
  }
  saveRDS(traces_by_group, file = paste(output_folder_grp,
                        group_file, "AllgroupTraces.Rds", sep=""))
  message("Traces Generated")

  # Get set with ransomized fixed parameters
  sobol_fixed = get_fixed_set(par_0, fixed, free, num_tables)
  saveRDS(sobol_fixed, file = paste(output_folder_grp,
                              group_file, "Sobols.Rds", sep=""))
  message("Sobol fixed set created")

  # Generate list of tables of optimized free parameters
  clust = makeCluster(ifelse(num_tables<params$cores, num_tables,
                             params$cores))
```

```
clusterExport(clust, c("sobol_fixed", "groups",
                       "traces_by_group", "output_folder_grp",
                       "group_file", "opt_par_group",
                       "num_nudge", "free"),
              envir=environment())
registerDoParallel(clust)
foreach(s = 1:nrow(sobol_fixed),
        .errorhandling = 'remove') %dopar% { # par loop
  source("ActionPotential.R")
  # Use the current parameter set for each optimization in this
  #   loop
  message(paste("Starting table", s))
  seed_param = unlist(sobol_fixed[s,])

  # Fuse all optimized parameters into one large df (with group
  #   id in one column)
  data_table = foreach(g = 1:length(groups),
                       .combine = rbind) %do% {
    AP_group = groups[[g]]
    time = AP_group$t

    # Get all parameter sets for each group as df
    group_pars = foreach(t = 1:nrow(traces_by_group[[g]]),
                         .combine = rbind) %do% {
      trace = traces_by_group[[g]][t,]
      AP = ActionPotential(seed_param, trace, time, "Group")

      # Search for the best of many optimizations
      out_par = AP$nudge_optimize(opt_par_group, num_nudge)
      out_par_free = out_par$par[names(out_par$par) %in%
                                   names(free)]

      # Make df of free parameters
      message(paste("Fit trace", t, "for group", g,
                    "on table", s))
      #data.frame(tbl=s, group=g, indiv=t, t(sapply(out_par_free,c
 )))
      data.frame(tbl=s, group=g, indiv=t,
                 t(sapply(out_par$par,c)),
                 values=out_par$value,
                 convergence=out_par$convergence)
    }
    group_pars
  }
  saveRDS(data_table, file = paste(output_folder_grp,
                                   group_file, s,
                                   "data_table.Rds", sep=""))
  message(paste("Done with table", s))
  data_table
}
stopCluster(clust)
message("Done computing tables")
```

```
} else {
  num_groups = length(trace_files)
  groups = foreach(i = 1:num_groups, .combine = c) %do% {
    AP_view = readRDS(paste(output_folder_grp, group_file, i,
                            "groupAP.Rds", sep=""))
    print(paste("Group", i, "Ave"))
    AP_view$display_action_potential()
    AP_view
  }
  for (i in 1:num_groups) {
    AP_tmp = groups[[i]]
    trace_Vs = AP_tmp$Vs
    st = AP_tmp$stabil_time
    dt = AP_tmp$dt
    l = length(AP_tmp$trace_data)
    trace_Vs = trace_Vs[(st/dt):(st/dt+l)]
    print(paste("Traces for group", i))
    gen_exp_traces(AP_tmp$all_params, fixed, free, trace_Vs,
                   AP_tmp$t, num_gen_trace)
  }
}

all_tables = vector("list", num_tables)
output_catcher = foreach(s = 1:num_tables,
                         .errorhandling = 'remove') %do% {
  all_tables[[s]] = readRDS(file = paste(output_folder_grp,
                                         group_file, s,
                                         "data_table.Rds",
                                         sep=""))
}
```

```{r linear, echo=FALSE, message=FALSE}
sim_data = foreach (d = 1:length(all_tables), .combine = rbind,
                    .errorhandling = 'remove') %do% {
  all_tables[[d]]
}
saveRDS(sim_data, file = paste(output_folder_grp, group_file,
                               "Allsim_data.Rds", sep=""))

for (param in names(opt_par_group)) {
  print(param)
  grp_comp = lme(reformulate("group", param),
                 random = ~ 1 + group | tbl, data=sim_data,
                 control = lmeControl(opt = 'optim'))
  print(summary(grp_comp))
}
```

```{r group, echo=FALSE, message=FALSE, results="asis"}
for (comp in 2:length(trace_files)) {
```

```
  print(paste("Comparision", comp-1))
  print(paste(trace_files[1], trace_files[comp]))
  table_means = foreach (d = 1:length(all_tables),
                         .combine = rbind,
                         .errorhandling = 'remove') %do% {
    dt = all_tables[[d]]
    group_data1 = dt[dt$group==1,]
    group_data2 = dt[dt$group==comp,]

    mean_diff = colMeans(group_data2) - colMeans(group_data1)
  }
  message("Generated table means A")

  for (col in 1:ncol(table_means)) {
    col_values = table_means[,col]
    num_zeros = sum(col_values == 0)
    if (num_zeros != nrow(table_means)) {
      hist(col_values, main=paste("Histogram for",
                                  colnames(table_means)[col]),
           xlab="Parameter Average")
    }
  }

  print(kable(table_means, caption="Means of Parameters"))
  print(kable(colMeans(table_means), caption="Means of Means"))
  print(kable(sim_data, caption="All Simulated Data"))
}
```

Listing 9: Functions for Group Analysis in RMD

J.3 Functions for Uniqueness Evaluation

This file is responsible for running the identifiability analysis as described in Section C. Details for each code chunk are outlined:

- `setup`: includes necessary libraries.
- `header`: declares what the optimization subset of parameters will be.
- `opt`: runs the `run_statistical_analysis` function as described in Appendix I.2.4.
- `main`: formats and shows the raw data form `run_statistical_analysis`.

````
---
title: "Action Potential - Statisitcal Analysis"
output:
  html_document: default
  pdf_document:
    latex_engine: xelatex
    pandoc_args: --listings
date: "`r Sys.Date()`"
params:
  run: FALSE
  cores: 1
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

library(doParallel)
library(foreach)
library(knitr)
source("ActionPotential.R")
source("vars.R")
```

```{r header, echo=FALSE}
with(as.list(par_0), {
  opt_par_stat <<- c(N_6=N_6,N_7=N_7,N_1=N_1,N_2=N_2,
                     M_6=M_6,M_7=M_7,M_1=M_1,M_2=M_2,
                     #H_4=H_4,H_5=H_5,H_6=H_6,H_3=H_3,
                     g_K=g_K, g_Na=g_Na, g_Leak=g_Leak)
})
```
````

```
```{r opt, echo=FALSE}
AP_stat = ActionPotentialStat(par_0, trace_data_ex, time_data_ex, "
    Stat")
if(params$run)
  AP_stat$run_statistical_analysis(cores=params$cores, opt_param=opt
    _par_stat)
stats = AP_stat$show_statistical_analysis()
```

\newpage

Plots of the Relationship Between Each Parameter for Multiple
    Optimizations

```{r main, echo=TRUE, results="asis"}
chunk_size = 4
for(i in 1:(ncol(stats)/chunk_size+1)) {
  end = i*chunk_size
  if(ncol(stats)<end) {
    end = ncol(stats)
  }
  print(kable(stats[((i-1)*chunk_size+1):end], caption="Statistical
    Analysis"))
}
```
```

Listing 10: Functions for Uniqueness Evaluation in RMD

References

E. D. Brodie and E. D. Brodie. Predator-prey arms races. *BioScience*, 49(7):557--568, jul 1999. doi: 10.2307/1313476.

E. D. Brodie, J. L. Hensel, and J. A. Johnson. Toxicity of the urodele amphibians taricha, notophthalmus, cynops and paramesotriton (salamandridae). *Copeia*, 1974 (2):506, jun 1974. doi: 10.2307/1442542.

E. D. Brodie, B. J. Ridenhour, and E. D. Brodie. THE EVOLUTIONARY RESPONSE OF PREDATORS TO DANGEROUS PREY: HOTSPOTS AND COLDSPOTS IN THE GEOGRAPHIC MOSAIC OF COEVOLUTION BETWEEN GARTER SNAKES AND NEWTS. *Evolution*, 56(10):2067, 2002. doi: 10.1554/0014-3820(2002)056[2067:teropt]2.0.co;2.

A. C. Daly, D. J. Gavaghan, C. Holmes, and J. Cooper. Hodgkin–huxley revisited: reparametrization and identifiability analysis of the classic action potential model with approximate bayesian methods. *Royal Society Open Science*, 2(12):150499, dec 2015. doi: 10.1098/rsos.150499.

R. E. del Carlo. *Steps along the evolutionary trajectories of skeletal muscle voltage-gatedsodium channels are guided by biophysical tradeoffs*. PhD thesis, Univeristy of Nevada, Reno, 2020.

R. E. del Carlo, J. S. Reimche, M. T. Hague, E. D. Brodie, N. Leblanc, and C. R. Feldman. Physiological tradeoffs of TTX resistance in $Na_V1.4$: Whole cell electrophysiology and tissue myography reveal reduced tetrodotoxicity at the cost of channel function. *Biophysical Journal*, 114(3):632a, feb 2018. doi: 10.1016/j.bpj.2017.11.3415.

C. R. Feldman, E. D. Brodie, E. D. Brodie, and M. E. Pfrender. The evolutionary origins of beneficial alleles during the repeated adaptation of garter snakes to deadly prey. *Proceedings of the National Academy of Sciences*, 106(32):13415--13420, aug 2009. doi: 10.1073/pnas.0901224106.

C. R. Feldman, E. D. Brodie, E. D. Brodie, and M. E. Pfrender. Genetic architecture of a feeding adaptation: garter snake *Thamnophis* resistance to tetrodotoxin bearing prey. *Proceedings of the Royal Society B: Biological Sciences*, 277(1698):3317--3325, jun 2010. doi: 10.1098/rspb.2010.0748.

C. R. Feldman, E. D. Brodie, E. D. Brodie, and M. E. Pfrender. Constraint shapes convergence in tetrodotoxin-resistant sodium channels of snakes. *Proceedings of the National Academy of Sciences*, 109(12):4556--4561, mar 2012. doi: 10.1073/pnas.1113468109.

S. L. Geffeney, E. Fujimoto, E. D. Brodie, E. D. Brodie, and P. C. Ruben. Evolutionary diversification of TTX-resistant sodium channels in a predator–prey interaction. *Nature*, 434(7034):759--763, apr 2005. doi: 10.1038/nature03444.

W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski. *Neuronal Dynamics: From Single Neurons toNetworks and Models of Cognition*. Cambridge Univ. Press, 2014.

M. T. J. Hague, G. Toledo, S. L. Geffeney, C. T. Hanifin, E. D. Brodie, and E. D. Brodie. Large-effect mutations generate trade-off between predatory and locomotor ability during arms race coevolution with deadly prey. *Evolution Letters*, 2(4): 406--416, aug 2018. doi: 10.1002/evl3.76.

C. T. Hanifin. The chemical and evolutionary ecology of tetrodotoxin (TTX) toxicity in terrestrial vertebrates. *Marine Drugs*, 8(3):577--593, mar 2010. doi: 10.3390/md8030577.

C. T. Hanifin, E. D. Brodie, and E. D. Brodie. Phenotypic mismatches reveal escape from arms-race coevolution. *PLoS Biology*, 6(3):e60, mar 2008. doi: 10.1371/journal.pbio.0060060.

P. Heistracher and C. C. Hunt. The relation of membrane changes to contraction in twitch muscle fibres. *The Journal of Physiology*, 201(3):589--611, may 1969. doi: 10.1113/jphysiol.1969.sp008774.

A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500--544, aug 1952a. doi: 10.1113/jphysiol.1952.sp004764.

A. L. Hodgkin and A. F. Huxley. The components of membrane conductance in the giant axon of *Loligo*. *The Journal of Physiology*, 116(4):473--496, apr 1952b. doi: 10.1113/jphysiol.1952.sp004718.

A. L. Hodgkin and A. F. Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of *Loligo*. *The Journal of Physiology*, 116 (4):449--472, apr 1952c. doi: 10.1113/jphysiol.1952.sp004717.

A. L. Hodgkin and A. F. Huxley. The dual effect of membrane potential on sodium conductance in the giant axon of *Loligo*. *The Journal of Physiology*, 116(4):497--506, apr 1952d. doi: 10.1113/jphysiol.1952.sp004719.

A. L. Hodgkin, A. F. Huxley, and B. Katz. Measurement of current-voltage relations in the membrane of the giant axon of *Loligo*. *The Journal of Physiology*, 116(4): 424--448, apr 1952. doi: 10.1113/jphysiol.1952.sp004716.

L. Y. Jan and Y. N. Jan. Voltage-gated potassium channels and the diversity of electrical signalling. *The Journal of Physiology*, 590(11):2591--2599, apr 2012. doi: 10.1113/jphysiol.2011.224212.

C. H. Lee, D. K. Jones, C. Ahern, M. F. Sarhan, and P. C. Ruben. Biophysical costs associated with tetrodotoxin resistance in the sodium channel pore of the garter snake, thamnophis sirtalis. *Journal of Comparative Physiology A*, 197(1):33--43, sep 2011. doi: 10.1007/s00359-010-0582-9.

M. I. Lindinger and G. J. Heigenhauser. Intracellular ion content of skeletal muscle measured by instrumental neutron activation analysis. *Journal of Applied Physiology*, 63(1):426--433, jul 1987. doi: 10.1152/jappl.1987.63.1.426.

H. A. Moniz, M. A. Richard, C. M. Gienger, and C. R. Feldman. Every breath you take: assessing metabolic costs of toxin resistance in garter snakes (*Thamnophis*). *Integrative Zoology*, 17(4):567--580, aug 2022. doi: 10.1111/1749-4877.12574.

A. L. Morris, M. W. MacArthur, E. G. Hutchinson, and J. M. Thornton. Stereochemical quality of protein structure coordinates. *Proteins: Structure, Function, and Genetics*, 12(4):345--364, apr 1992. doi: 10.1002/prot.340120407.

S. Ohki. The electrical capacitance of phospholipid membranes. *Biophysical Journal*, 9(10):1195--1205, oct 1969. doi: 10.1016/s0006-3495(69)86445-3.

X. Pan, Z. Li, Q. Zhou, H. Shen, and K. Wu. Structure of the human voltage-gated sodium channel $Na_v1.4$ in complexwith $\beta 1$. *Science*, 362, Oct. 2018.

J. S. Reimche, E. D. Brodie, A. N. Stokes, E. J. Ely, H. A. Moniz, V. L. Thill, J. M. Hallas, M. E. Pfrender, E. D. Brodie, and C. R. Feldman. The geographic mosaic in parallel: Matching patterns of newt tetrodotoxin levels and snake resistance in multiple predator–prey pairs. *Journal of Animal Ecology*, 89(7):1645--1657, apr 2020. doi: 10.1111/1365-2656.13212.

M. Renardy, L. R. Joslyn, J. A. Millar, and D. E. Kirschner. To sobol or not to sobol? the effects of sampling schemes in systems biology applications. *Mathematical Biosciences*, 337:108593, jul 2021. doi: 10.1016/j.mbs.2021.108593.

O. J. Walch and M. C. Eisenberg. Parameter identifiability and identifiable combinations in generalized hodgkin–huxley models. *Neurocomputing*, 199:137--143, jul 2016. doi: 10.1016/j.neucom.2016.03.027.

F. H. Yu and W. A. Catterall. Overview of the voltage-gated sodium channel family. *Genome Biology*, 4(3):207, 2003. doi: 10.1186/gb-2003-4-3-207.

H. H. Zakon. Adaptive evolution of voltage-gated sodium channels: The first 800 million years. *Proceedings of the National Academy of Sciences*, 109(supplement_1): 10619--10625, jun 2012. doi: 10.1073/pnas.1201884109.