University of Nevada, Reno

A Python Library for Ion Channel Modeling

A thesis submitted in partial fulfillment of the
the requirements for the degree Master of Science in
Computer Science and Engineering

by

Gareth B. Ferneyhough

Dr. Frederick C. Harris, Jr. / Thesis Advisor

August, 2013

University of Nevada, Reno
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

GARETH B. FERNEYHOUGH

entitled

A Python Library For Ion Channel Modeling

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

MASTER OF SCIENCE

Dr. Frederick C. Harris, Jr., Ph. D, Advisor

Dr. Sergiu Dascalu, Ph. D, Committee Member

Dr. Normand Leblanc, Ph. D, Graduate School Representative

Marsha H. Read, Ph. D., Dean, Graduate School

August, 2013

Abstract

The creation and simulation of ion channel models using continuous-time Markov processes is a powerful and well-used tool in the field of electrophysiology and ion channel research. While several software packages exist for the purpose of ion channel modeling, none are available as a Python library. In an attempt to provide an easy-to-use, yet powerful Markov model-based ion channel simulator, we have developed ModFossa, a Python library supporting easy model creation and stimulus definition, complete with a fast numerical solver, and attractive vector graphics plotting.

Dedication

For my parents.

Acknowledgments

First, I would like to thank my parents; it is with their love and support over my educational career that I have attained my goal of a master's degree.

Next, I would like to thank my advisor, Dr. Frederick C. Harris, Jr., and committee members Dr. Sergiu Dascalu and Dr. Normand Leblanc for their time and helpful suggestions. Additionally, I would like to acknowledge all of the faculty at the university, especially in the Computer Science department for their dedication.

Lastly, I owe a big thank-you to Dr. Corey Thibeault and Mr. John Kenyon, whose enthusiasm, guidance, and encouragement over the last year has proved tremendously helpful.

This work was partially supported by the U.S. Ofce of Naval Research (N000140110014).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ion channels are trans-membrane proteins that control the passive flow of ions into, and out of a biological cell. They exist in all cell types and play crucial roles in cellular sensing and communication, maintenance of the membrane potential, and regulation of cell volume.

For example, voltage-gated sodium, potassium, and calcium channels (among others) work together to shape the automatically recurring cardiac action potential that give our hearts their sure (and usually steady) beat. Philosophy aside, the very thoughts passing through the consciousness of a student as he reads this page exist as nothing more than electro-chemical signals in the brain. Specifically, voltage-gated sodium and potassium channels integrated in the cell membrane of all neurons allow electrical signals to propagate rapidly down the axon in the form of an action potential. When the action potential reaches the end of the axon, voltage-gated calcium channels regulate the release of special chemical messengers called neurotransmitters into the synapse (the structure that connects neurons to one another). As the neurotransmitters diffuse across the synapse they bind to ligand-gated ion channels on the post-synaptic neuron. Depending on the type of ion channel and the type of neurotransmitter, the channels may open or close, thereby altering the membrane voltage.

If the post-synaptic neuron receives enough excitatory stimulus from the many presynaptic neurons in which it is connected, its membrane potential will depolarize to a point and it too will experience an action potential, thus passing the signal along to other neurons.

As further evidence of their vital importance, ion channels are specifically targeted by several naturally occurring toxins and poisons, including venom from spiders, snakes, scorpions, and fish. It comes as no surprise that the development of new types of drugs which alter the behavior of certain channels is a major motivator behind ion channel research. As in most scientific fields, researchers often develop mathematical models to assist in their understanding of the results. Ion channels are no exception; several classes of ion channel models exist that allow for the study of ion channel behavior under varying conditions. One such generalization involves modeling channel gating using continuous time Markov processes, and solving the underlying ordinary differential equations that arise. A number of commercial as well as open-source software programs exist for this purpose, but it is also common for scientists to implement their own solution using the differential equations solvers in Matlab, for example.

In order to provide an elegant and simple solution for the creation and simulation of Markov model-based ion channels, we present ModFossa, a simulator written in C++ with an easy-to-use Python interface. Several published models of ion channel gating have been reproduced using our simulator; these results, along with performance analysis are presented in the following chapters. In Chapter 2 we present several mathematical, biological, and electrical concepts related to ion channel modeling. In Chapter 3, we present the design and implementation of ModFossa. Results are provided in Chapter 4. Finally, we conclude with a discussion of applications and future work in Chapter 5.

Chapter 2

Background

2.1 Biology background

To understand the function and behavior of ion channels, several points need to be discussed. The background comes from a wide disciplinary range, including biology, chemistry, thermodynamics, electricity, probability theory, and differential equations. The most important of these points are presented in the following subsections. For more information, the reader is referred to Hille's extensive book, *Ion Channels of Excitable Membranes* [8], and Keener and Sneyd's work on the applied mathematics of cellular physiology [11].

2.1.1 Salts, ions, and the cellular solutions

Before discussing the biology of cells and ion channels any further, a quick discussion regarding ionic solutions and the *electrochemical gradient* is needed. As any good student should know, ions are atoms or molecules that have a positive or negative charge due to differing numbers of protons and electrons. Specifically, anions have more electrons than protons, so they have a negative charge. Conversely, cations are “missing” electrons, and therefore have a positive charge.

Ions arise from a variety of natural processes, including the dissolution of salts in water. For example, when table salt, or more officially, sodium chloride ($NaCl$) dissolves in water, sodium cations with one positive charge (Na^+) and chloride anions with one negative charge (Cl^-) become dissociated and free to move around independently. Similarly, the salts potassium chloride (KCl) and calcium chloride ($CaCl_2$) dissolve into solutions of free K^+, Cl^-, and Ca^{2+} ions.

Solutions of dissolved salts are of critical importance to a cell’s function. Many intracellular and extracellular environments contain (among other molecules) these four ions in very specific concentrations. Ionic concentration values for three different cell types are provided in Table 2.1.

Now let us discuss the concept of the electrochemical gradient. Consider the situation presented in Figure 2.1 in which a water-filled bath is separated into two halves by a *semi-permeable* membrane. This membrane contains small pores which allow K^+ ions, but nothing else, to pass through. Therefore, K^+ ions can move freely across the membrane from the left to right side of the bath, and vise versa. A more concise description of the membrane is to say that it is perfectly permeable to K^+. As a convention, the left side of the bath represents the inside of a biological cell (intracellular), while the right side is the extracellular environment.

At time $t = 0$, a concentrated solution of KCl is introduced to the left side of the bath, while a weaker solution of KCl is introduced to the right side. This configuration is represented by the larger letters of K^+ and Cl^- on the left side. A voltmeter measures the voltage between the left and right sides, and reads zero volts at $t = 0$, because the number of cations and anions are equal on both sides of the membrane. Because the left side has a higher concentration of K^+, the ions experience a diffusion force which is represented by arrows crossing the membrane.

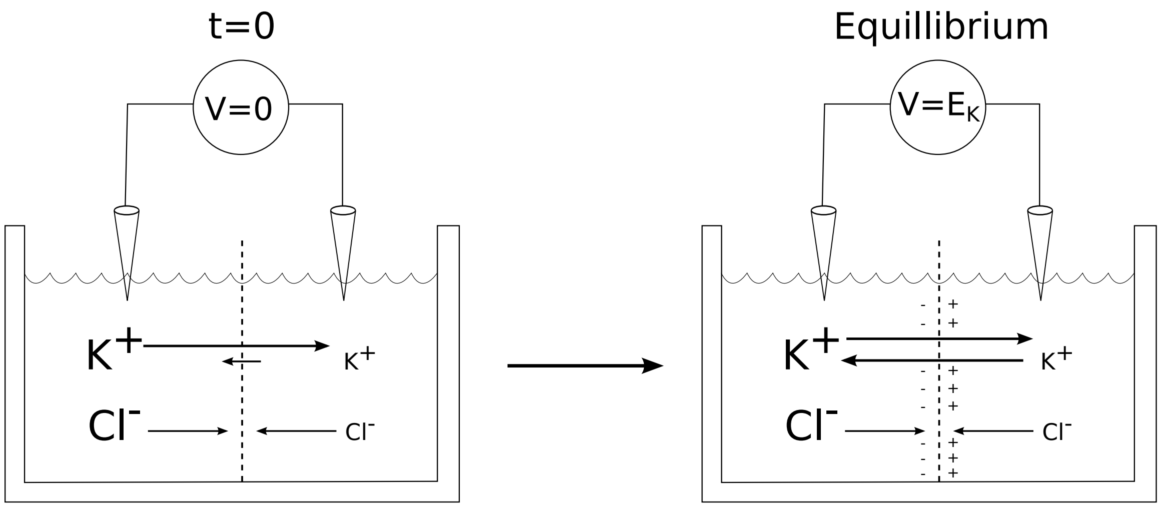


Figure 2.1: A voltmeter measures the potential difference between the two sides of a bath which is separated by a membrane perfectly permeable to K^+. The left side contains a higher concentration of dissolved KCl than the right. At the first instant the solutions are added, no ions have had time to diffuse through the membrane, so the charges are balanced resulting in a voltage of zero. However, K^+ ions immediately begin diffusing through the membrane, down their concentration gradient resulting in a buildup of positive charge of the right side of the membrane. Eventually the electrical and diffusion forces will balance and the system will reach equilibrium. The equilibrium potential is equal to the Nernst potential of potassium, E_K.

Cl^- ions, however, cannot cross the membrane as it is not permeable to them.

The instant after the solutions are added to the bath, K^+ will begin moving across the membrane from left to the right sides, driven by the diffusion force. As this movement of charge continues, the right side will become more positive. Eventually the buildup of positive charge will deter K^+ ions from moving to the right side. At this point, the system is at equilibrium. The chemical gradient pushing K^+ to the right is balanced by the electrical force opposing the movement of charge. The voltage at which the bath reaches equilibrium is called the Nernst potential, and is calculated for a generic ion species S like so:

$$E_S = \frac{RT}{zF} ln \left(\frac{[S]_e}{[S]_i} \right), \tag{2.1}$$

where R is the gas constant, T is the absolute temperature in Kelvins, z is the charge on ion S, F is Faraday's constant, and $[S]_e$ and $[S]_i$ represent the extracellular

Table 2.1: Various ionic concentrations, Nernst potentials, and resting potentials for different cell types. Voltages are in mV and concentrations in mM. Adapted from [11].

	Squid Giant Axon	Frog Sartorius Muscle	Human Red Blood Cell
Intracellular concentrations			
Na^+	50	13	19
K^+	397	138	136
Cl^-	40	3	78
Extracellular concentrations			
Na^+	437	110	155
K^+	20	2.5	5
Cl^-	556	90	112
Nernst potentials			
V_{Na}	+56	+55	+55
V_K	-77	-101	-86
V_{Cl}	-68	-86	-9
Resting potentials	-65	-99	-6 to -10

and intracellular concentrations of ion S. Values for these constants are provided in Table 2.2, while concentration values for select ion species as well as their Nernst potentials are listed in Table 2.1.

In summary, the combination of the diffusion and electrical forces is called the *electrochemical gradient*, and can be thought of as a form of potential energy for a cell. In fact, the energy required for an action potential (discussed briefly in Chapter 1) is provided almost entirely from the electrochemical gradient between the extracellular and intracellular regions of a neuron.

2.1.2 The cell membrane

Biological cells are divided into two main groups: prokaryotic cells which include bacteria, and eukaryotic cells such as plant and animal cells. Prokaryotic and eukaryotic cells are both enclosed in a selectively permeable biological membrane known as the cell (or plasma) membrane. Eukaryotic cells also have a number of internal membranes enclosing their nucleus, mitochondria, and several other organelles. Prokaryotes on the other hand do not have a nucleus or any other membrane-bound organelles.

The cell membrane of both prokaryotes and eukaryotes, as well as the membranes of certain organelles all contain ion channels. From this point on, however, the focus will be on the cell membrane of eukaryotic cells.

The cell membrane separates the intracellular contents from the extracellular environment and is composed of a thin double layer of lipids about 7.5 nM thick [11]. A lipid is a simple molecule with one hydrophilic head and one or more hydrophobic tails. Lipids group together in a double layer to form the phospholipid bilayer of the cell membrane. Dispersed throughout the cell's membrane are an an assortment of integral (also called trans-membrane) proteins, as well as a variety peripheral proteins and other molecules such as carbohydrates. The variety and importance of membrane proteins is evident by a recent study estimating that such proteins account for 30% of genes [4]. Figure 2.2 provides a view of a typical cell membrane.

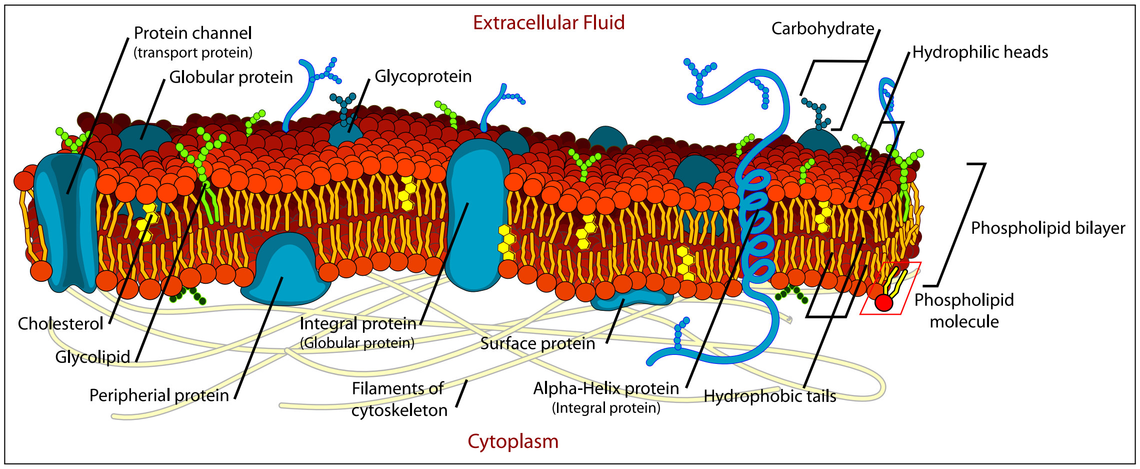


Figure 2.2: Representation of the cell membrane showing the lipid bilayer, integral and peripheral proteins, and carbohydrates. From Wikipedia [21].

An important characteristic of the cell membrane is that it is selectively permeable, meaning that it allows certain molecules to pass though in either direction in controlled quantities. For example, waste products are allowed to leave the cell, and certain sugars, acids, and other molecules can enter. Also affected by the membrane's selective permeability are ions, such as Na^+ (sodium), K^+ (potassium), Cl^- (chloride), and Ca^{2+} (calcium).

The transportation of molecules across the membrane can either be passive (requiring no net energy), or active (requires energy, such as ATP). An important active transport mechanism is the $Na^+/K^+ - ATPase$, which is also called the sodium-potassium pump for the sake of readability. This pump is a trans-membrane protein that uses the energy stored in ATP to pump Na^+ out of the cell and K^+ into the cell [11]. In addition to pumps, integral proteins called exchangers also facilitate active transport across the cell membrane. The $Na^+ - Ca^{2+}$ exchanger uses the energy gained from allowing Na^+ entry into the cell to remove Ca^{2+}. One may question why allowing Na^+ into the cell provides energy, while removing Ca^{2+} requires energy. The reason this is so is due to the electrochemical gradients arising from the ionic concentration levels that the cell maintains through active transport.

At the cell's resting potential, the concentration of Na^+ is much higher outside the cell than inside the cell, so Na^+ wants to enter because the electrochemical gradient is driving it into the cell. Ca^{2+} also a higher extracellular concentration, so it takes energy to remove it from the cell because it must be moved up, or *against* its electrochemical gradient.

The last transport mechanism that deserves mention are ion channels, trans-membrane proteins which facilitate the diffusion of select ion species down their electrochemical gradient. Ion channels are therefore referred to as passive transport mechanisms. As a simplification, an individual channel can be thought of as either open, or closed. In the open configuration, channels allow ions to pass through at speeds of 10^8 ions per second, near diffusion speed [4].

From the perspective of the whole cell, the large number ion channels together can be said to affect the *permeability* of the membrane to a certain ion species.

In summary, the cell membrane uses ion channels, pumps, and exchangers to maintain a certain concentration gradient for a number of ion species. The membrane's permeability to an ion species depends on the types of ion channels in that cell type, as well as their current states (open, or closed).

2.1.3 Membrane potential

Given various intracellular and extracellular ionic concentrations, as well as the membrane's relative permeability to these ions, one can calculate the resting membrane potential of a cell. The resting membrane potential First, some basic concepts in electricity need to be refreshed.

Quantity of charge is measured in coulombs, and is abbreviated C. Current, (I), represents the movement of charge and has units coulombs per second, or amperes (A). Conductance, (G), is the inverse of resistance (R), and measures the ease of current flow between two points. The unit of conductance is siemens (S), while the unit of resistance is ohms (Ω). Voltage, V, measures the potential difference between two points. Note that the terms "potential" and "voltage" can be used interchangeably.

Ohm's law defines the relationship between voltage, current, and resistance:

$$V = IR. \tag{2.2}$$

It may also be written in terms of conductance:

$$V = I/G\ . \tag{2.3}$$

Now let us apply Ohm's law to a biological cell. Considering the simplification that the membrane is permeable only to ion species S, the membrane's conductance, G, is directly related to how easily S passes though its ion channels. This is called channel conductance, and is a function of the opening and closing (referred to as gating from now on) of the channels as well as their type. I_S is the current through the membrane due to the movement of species S. In this case, the voltage, V, is referred to as the *driving force*. The driving force is different for each different ion species in the cell - Na^+ ions, for example, experience a force greater in magnitude and opposite in direction compared with K^+ ions.

It is worth noting that it is difficult to measure the conductance of a cell. Permeability, on the other hand, is easy to measure. Permeability and conductance are related but not equivalent; however, they are treated as such in electrophysiology.

To summarize: the current from ion S is equal to the driving force of S times the membrane conductance to S. Formally:

$$I_S = G_S(DrivingForce_S). \tag{2.4}$$

Driving force is determined by the membrane voltage, V_m, and E_S, the equilibrium potential (calculated using Equation 2.1) for ion S:

$$DrivingForce_S = (V_m - E_S). \tag{2.5}$$

Substituting:

$$I_S = G_S(V_m - E_S). \tag{2.6}$$

Equation 2.6 is used by the software described in Chapter 3 to calculate the ionic currents given a known membrane voltage and simulated channel conductances.

To calculate the resting potential of a cell permeable to more than one ion type,

Table 2.2: Physical constants. From [8].

Faraday's constant	F	9.6485×10^4	$Cmol^{-1}$
Absolute temperature	$T(K)$	273.15	$^{\circ}Celcius$
Gas constant	R	8.3145	$VCmol^{-1}K^{-1}$

we need to take into account the concentrations of the various ion species, as well as the conductance of their respective channels. The equation that does this is the Goldman-Hodgkin-Katz (GHK) voltage equation:

$$E_m = \frac{RT}{F}\, ln \frac{P_K[K]_e + P_{Na}[Na]_e + P_{Cl}[Cl]_i}{P_K[K]_i + P_{Na}[Na]_i + P_{Cl}[Cl]_e}, \tag{2.7}$$

where E_m is the resting membrane potential and P_S represents the relative permeability of the membrane to ion species S.

It is worth stating that unlike the Nernst equation which is universally true, the GHK equation depends on the assumption of a constant electric field.

2.2 Modeling ion channels

The modeling of ion channels dates back over 100 years [13]. As is with many mathematical models, the motivations driving the development of ion channel models include:

- striving to infer fundamental knowledge of the underlying physical processes,
- matching experimental data to the model's predictions,
- using the model to predict behaviors which are difficult to observe, and
- developing common terminology and knowledge among researchers.

The interpretation of ion channels models during the pre-molecular era provided the primary source of information about channel structure [13]. However, current technology such as molecular biology and x-ray diffusion are beginning to prove atomic-resolution detail of bacterial as well as some mammalian channels, though the fraction of channels with known structure remains low [4, 13].

Hodgkin and Huxley's landmark model describing the ionic mechanisms behind the action potential of a squid giant axon provided the foundation for many modern practices in electrophysiology and ion channel modeling [9, 11]. Their model describes the voltage-dependent and time-dependent behavior of Na^+ and K^+ conductances using a coupled set of ordinary differential equations. While the Hodgkin-Huxley model is still used today due to its simplicity and low number of parameters, it does exhibit several shortcomings. These include, for example, a lack of connectivity between activating and inactivating gates in the Na^+ channel, as well as the premise that the inactivation gate can only close after the activation gate opens [7].

2.2.1 Markov models

Markov models for ion channels are an extension of the Hodgkin-Huxley formalism, and are good for modeling single channel data, gating currents, and drug interactions. Unlike Hodgkin-Huxley models, Markov models show the state dependence of activation and inactivation [6].

Theory

Ion channels can be modeled as continuous time Markov processes (see Section 2.3.1), where a simplification of the channels functional physical shape is represented by states in the Markov chain. A model can have several open, closed, deactivated, and inactivated states, as well as states which represent the binding of ligands. The rates connecting the various states are time independent kinetic rates which can be constant, or dependent on the membrane voltage, or intra- and extra-cellular ionic concentrations.

Criticism

Markov models are not without critics. In a one page perspective titled *Are rate constants constant?* Jones challenges the community to examine the benefits of time-dependent rate constants over Markov models [10]. Jones cites the findings by Uebachs [20] who reports that the rate of recovery from inactivation in a type of Ca^{+2} channel depends on the length of depolarization, which goes against the basic principle of chemical kinetics that rate constants remain constant in a constant environment [10].

Although Markov models are memoryless by definition, the time-dependent behavior described by Uebachs can be reproduced by having a long chain of open or closed states [10, 14]. Nevertheless, Jones concludes that time-dependent rate constants as used by fractal channel models, would provide a more explicit memory than multistate Markov models.

2.2.2 Fractal models

Liebovitch has been the main proponent of fractal ion channel models for over 15 years [14, 15, 16]. One of his main arguments questioning the use of Markov models is founded in the body of evidence depicting ion channel proteins as complex structures with a continuum of states, exhibiting large and small motions over varying time scales. Fractal models as proposed by Liebovitch contain a large number of conformational states with kinetic rates connecting them. The rates, however, are linked and not independent as in Markov models. This has the effect of giving the channel “memory” [14].

An additional concern raised by Liebovitch is the use of exponential rate constants.

He provides a friendly reminder to his colleagues that the fitting of experimental data to sums of exponentials is famously ill-conditioned [1, 14]. As confirmation he provides a quote from a numerical methods text which states that those who try to determine the parameters of such equations from experimental data "must be spanked or counseled. At the very least, keep them from obstructing Progress and the computer!" [1].

In their defense, other researchers have argued that Markov models with exponential rate constants and a small number of states fit the experimental data better than fractal models [17]. Nevertheless, fractal models continue as the number one challenger to Markov models.

2.3 Math Background

2.3.1 Continuous time Markov process

The following information is inspired from David Anderson's notes on biological stochastic processes [2].

Continuous time Markov processes are commonly used to simulate stochasticity in a variety of fields including biology, chemistry, physics, and economics. As the name implies, a continuous time Markov process is similar to a discrete time Markov chain, with the exception that transitions between states can occur at any time, with a exponentially distributed probability. Both discrete and continuous time Markov processes satisfy the Markov property which says that the probability distribution of future states depends only on the current state, and not on the preceding events. This "memoryless" behavior is what gives Markov processes their stochastic behavior. The property is named after the Russian mathematician Andrey Markov.

Continuous time Markov chains are easily described using weighted directed graphs, where nodes represent the states, and the edges are labeled with the *rates* of transitioning between states. Figure 2.3 shows a three state Markov chain in which state a transitions to state b with rate k_1, and c transitions to b with a rate k_4. Note that unlike the discrete case, the transition rates are not equivalent to probabilities and thus are not restricted to values $[0-1]$.

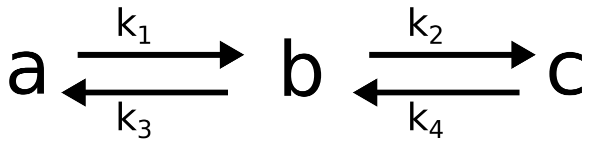


Figure 2.3: A three state Markov chain with states a, b, and c, and transition rates $k_1 - k_4$.

To represent the connections between N states in a continuous time Markov chain, we can use a weighted adjacency matrix, D, where D is size $N \times N$, and the non-diagonal entries, D_{ij} represent the edge weight, or *transition rates* from state j to state i. Self-connections are not allowed, i.e., state b cannot define a rate to itself. Therefore, the diagonal entries of the matrix must be zero.

The adjacency matrix representing the continuous time Markov chain given in Figure 2.3 is:

$$D = \begin{bmatrix} 0 & k_3 & 0 \\ k_1 & 0 & k_4 \\ 0 & k_2 & 0 \end{bmatrix} \tag{2.8}$$

The adjacency matrix describes the connections of a Markov chain. To describe the rates in which we move between states, we use the transition matrix (also called the infinitesimal generator matrix), A, which is the adjacency matrix with special entries in the diagonal such that each column sums to zero [2]. Thus the diagonal entries must be equal to the negative sum of all the rates for that column. For the three state example, we get the following transition matrix:

$$A = \begin{bmatrix} -k_1 & k_3 & 0 \\ k_1 & -(k_2+k_3) & k_4 \\ 0 & k_2 & -k_4 \end{bmatrix} \tag{2.9}$$

Now we can describe the evolution of the state probabilities by deriving a set of ordinary differential equations, called the Kolmogorov forward equations [2]. In the case of natural sciences, they are known as the master equation, or chemical master equation. The master equation has the form:

$$\frac{d\vec{P}}{dt} = A\vec{P}, \tag{2.10}$$

where $\frac{d\vec{P}}{dt}$ represents the change of the state probability vector $\vec{P}$, and A is the transition matrix. The master equation in its expanded form for the three state example is:

$$\begin{bmatrix} dP_a/dt \\ dP_b/dt \\ dP_c/dt \end{bmatrix} = \begin{bmatrix} -k_1 & k_3 & 0 \\ k_1 & -(k_2+k_3) & k_4 \\ 0 & k_2 & -k_4 \end{bmatrix} \begin{bmatrix} P_a \\ P_b \\ P_c \end{bmatrix} = \begin{bmatrix} -k_1P_a + k_3P_b \\ k_1P_a - (k_2+k_3)P_b + k_4P_c \\ k_2P_b - k_4P_c \end{bmatrix} \tag{2.11}$$

We now have three differential equations to describe the evolution of our three state system. However, this system is over specified. Instead of explicitly calculating the evolution of the last state (state c in our example), we infer it via the conservation of probabilities like so:

$$P_N = 1 - \sum_{i=1}^{N-1} P_i. \tag{2.12}$$

In other words, the probability of model being in the last state, plus the sum of the other state probabilities must be equal to one.

Using the conservation of probabilities, our system can be re-written as:

$$\begin{bmatrix} dP_a/dt \\ dP_b/dt \\ P \end{bmatrix} = \begin{bmatrix} -k_1 & k_3 & 0 \\ k_1 & -(k_2+k_3) & k_4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_a \\ P_b \\ P_c \end{bmatrix} = \begin{bmatrix} -k_1P_a + k_3P_b \\ k_1P_a - (k_2+k_3)P_b + k_4P_c \\ P_a + P_b + P_c \end{bmatrix}. \tag{2.13}$$

If we wish to solve our system of ordinary differential equations as an initial value problem, we need initial conditions. When simulating the current through ion channels, it is often desired to start solving the system while it is at steady state, that is, when all of the state probability derivatives are equal to zero:

$$\begin{bmatrix} dP_a/dt \\ dP_b/dt \\ P \end{bmatrix} = \begin{bmatrix} -k_1 & k_3 & 0 \\ k_1 & -(k_2+k_3) & k_4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} P_a \\ P_b \\ P_c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, or \tag{2.14}$$

$$\vec{P'} = A\vec{P_{ss}} = \vec{P}. \tag{2.15}$$

Thus,

$$\vec{P_{ss}} = A\Big/\vec{P}\,. \tag{2.16}$$

Equation 2.16 is used in the implementation of our software (see Chapter 3) to set the state probabilities to steady state before beginning integration. During simulation we supply the derivatives of the state probabilities (calculated using Equation 2.10) to the solver during each integration step.

Table 2.3: Feature comparison of several ion channel simulators which use Markov models.

Name	Model	GUI	Scriptable	Plotting	Plot format	Simulation type	OS	License
IonChannelLab	Markov	yes	no	yes, interactive	bitmap	numerical integration, Monte Carlo, Q-Matrix	Win.	Free, no source available
ChannelLab	Markov	yes	no	yes	bitmap	numerical integration, Monte Carlo	Win.	Proprietary
QUB	HMM	yes	yes	yes, interactive	bitmap	numerical integration	Win., Mac, Linux	GPL
ModFossa	Markov	no	yes, from Python	yes	vector	numerical integration	Linux	GPL

2.4 Existing ion channel simulators

We provide a comparison of several Markov model based ion channel simulators. Table 2.3 compares the basic features of each simulator, including our solution, ModFossa.

2.4.1 IonChannelLab

IonChannelLab is a standalone Windows application targeted towards the non-programmer, and is therefore focused on ease of use [5]. It is available free of charge; however, source code is not available.

General observations

The software provides a nice looking interface which is uncluttered, intuitive, and easy to learn; however, the lack of online help (especially context-sensitive help) proves somewhat disappointing. Several example projects are available, including Hodgkin-Huxely type models, as well implementations of the author's own research on chloride channels.

IonChannelLab allows users to save projects in an XML format for later use. Lastly, the software is not scriptable. If it were, IonChannelLab would no doubt be more attractive to advanced users.

Model definition

IonChannel lab facilitates easy Markov model creation by allowing the user to drag and drop states into arbitrary locations on a 2D grid. A transition is created by connecting two states using the mouse. Each transition specifies two rate constants, one for the rate from state A to state B, and another from B to A. Rate constants are specified by a unique string name, and are defined using the function editor window (Figures 2.5 and 2.6). The function editor allows the user to specify the parameters of several available rate constant types, including exponential voltage; exponential voltage with temperature dependence; double exponential with temperature dependence; ligand gated; exponential ligand gated with temperature dependence; Hodgin-Huxely types A, B, and C; constant; custom voltage; and custom ligand. The equation used to calculate each rate constant type is clearly shown in the function editor window, a useful feature not found in other simulators reviewed here.

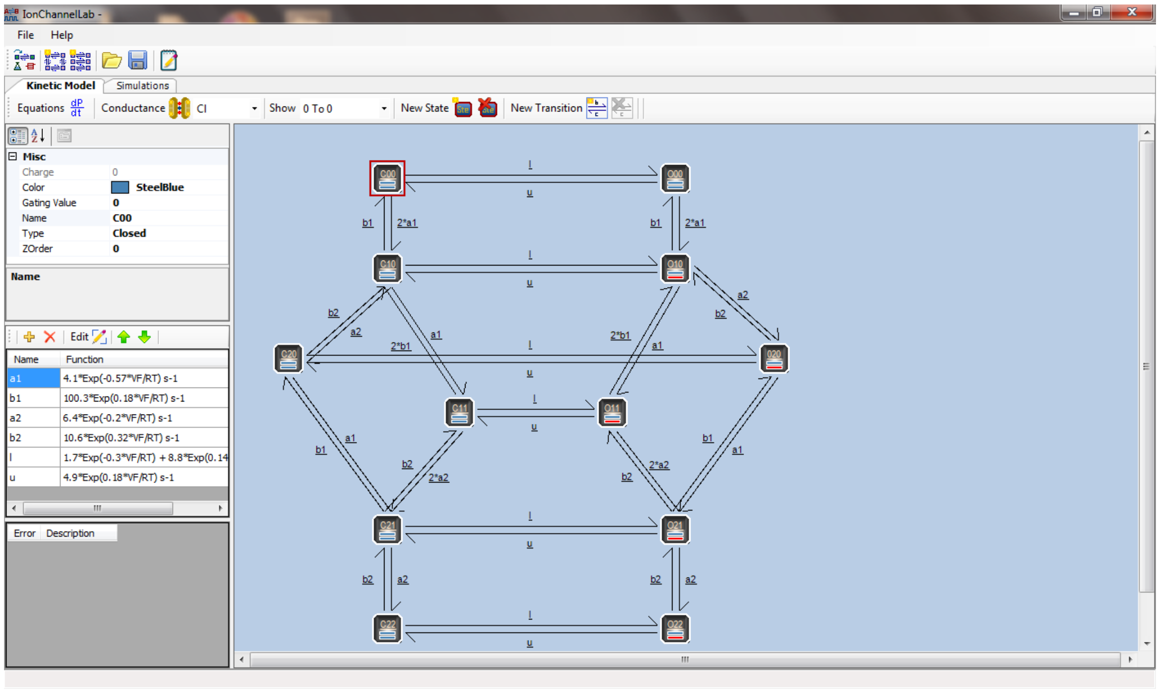
IonChannelLab's model definition screen also provides the user with two choices for the permeation model, Ohm's law and Goldman-Hodgin-Katz.

Stimulus definition

Voltage and concentration protocols are defined in the stimulus window as shown in Figure 2.7. Constant, stepped, and ramped stages are supported for both voltage and concentration protocols. A visualization of the protocol is provided in the stimulus window, and is updated immediately after a change in any of the fields.

Experiment definition

The experiment definition window provides settings and visualizations specific to the experiment. In creating a new experiment using IonChannelLab, the user chooses from the following experiment types: current vs time, conductance vs time, gating



Misc	
Charge	0
Color	SteelBlue
Gating Value	0
Name	C00
Type	Closed
ZOrder	0

Name

Name	Function
a1	4.1*Exp(-0.57*VF/RT) s-1
b1	100.3*Exp(0.18*VF/RT) s-1
a2	6.4*Exp(-0.2*VF/RT) s-1
b2	10.6*Exp(0.32*VF/RT) s-1
l	1.7*Exp(-0.3*VF/RT) + 8.8*Exp(0.14
u	4.9*Exp(0.18*VF/RT) s-1

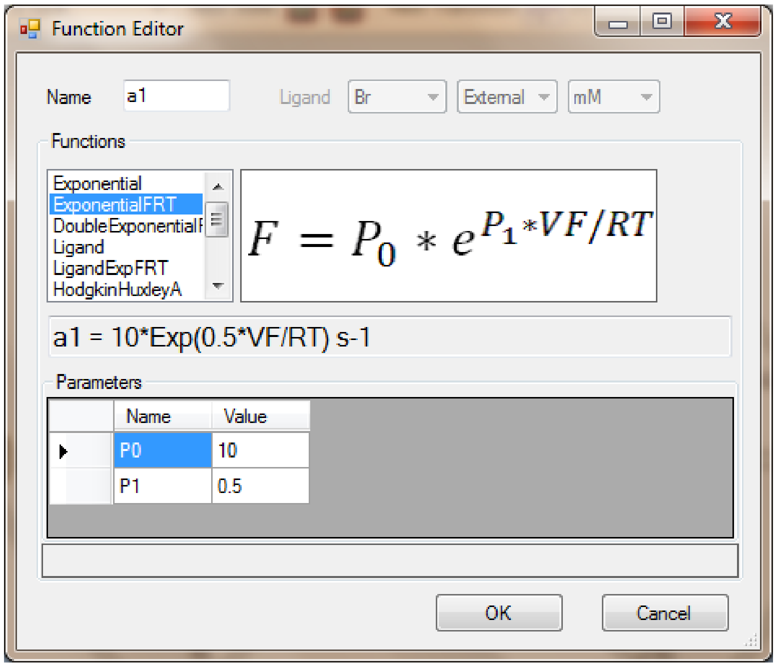
Figure 2.4: IonChannelLab model definition screen. The main area contains the Markov model with 12 states connected by various rate constants. The rate constants and their equations are shown in the sidebar on the left.

current vs time, math expression vs time, steady state current vs X, steady state conductance vs X, steady state expressions vs X, and time constants vs X.

The software gives the user the option of selecting one of several different solvers, including q-matrix, integration via Runge-Kutta, integration via Gear's BDF, and single channel Monte Carlo. Implementation of the solvers is provided by DotNumerics, a numerical library written using Microsoft's .NET framework. The initial conditions of the simulation can be calculated from the holding values, or set manually by the user. Additional experiment settings include the number of channels to simulate, temperature, and an optional noise.

Plotting and analysis

All plots are interactive and allow for the user to pan and zoom. If the user wishes to save a plot, he can either save the data in CSV format, or copy the image to the



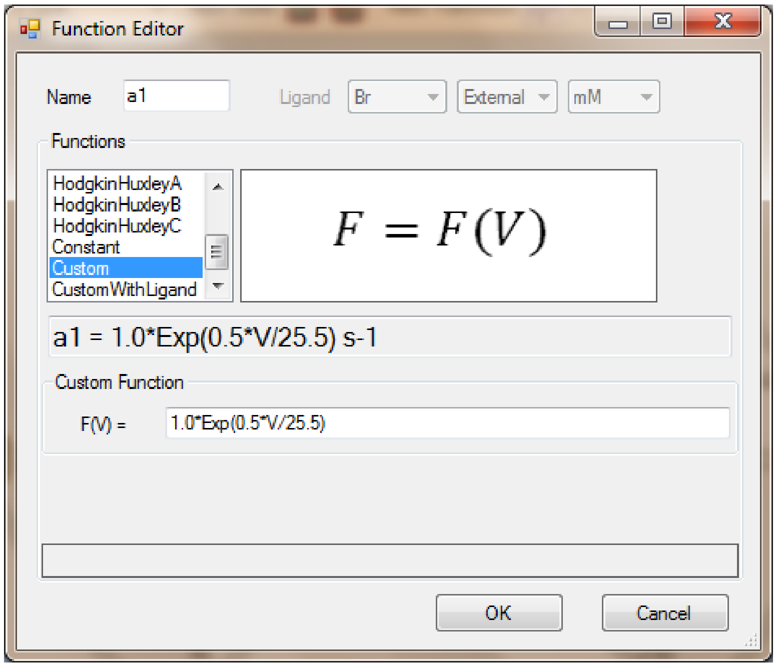
Name a1 Ligand Br External mM

$$F = P_0 * e^{P_1 * VF/RT}$$

a1 = 10*Exp(0.5*VF/RT) s-1

Name	Value
P0	10
P1	0.5

Figure 2.5: IonChannelLab function editor window showing an exponentially voltage dependent rate constant to which the user has assigned the name "a1". The equation used to calculate the rate is shown in easy-to-read text.



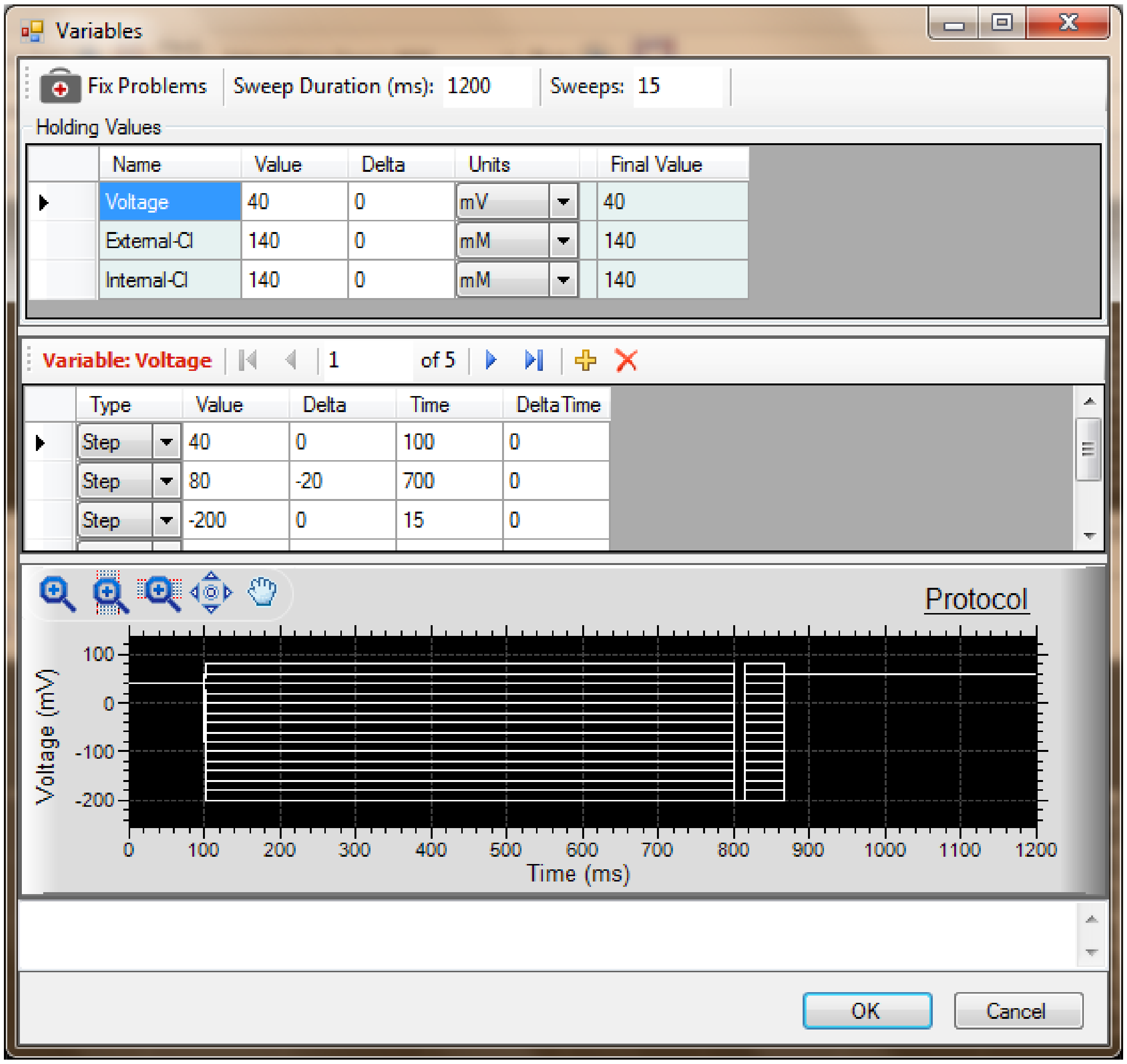
Name a1 Ligand Br External mM

$$F = F(V)$$

a1 = 1.0*Exp(0.5*V/25.5) s-1

F(V) = 1.0*Exp(0.5*V/25.5)

Figure 2.6: IonChannelLab function editor window showing a custom rate constant. The user has defined this particular rate constant as exponentially voltage dependent.



Sweep Duration (ms): 1200

Sweeps: 15

Holding Values

Name	Value	Delta	Units	Final Value
Voltage	40	0	mV	40
External-Cl	140	0	mM	140
Internal-Cl	140	0	mM	140

Variable: Voltage 1 of 5

Type	Value	Delta	Time	Delta Time
Step	40	0	100	0
Step	80	-20	700	0
Step	-200	0	15	0

Figure 2.7: IonChannelLab stimulus editor window showing a visualization of the voltage protocol. As shown in the top of the window, this experiment also defines internal and external chloride concentration protocols.

clipboard. There is no option to export plots in a vector graphics format.

2.4.2 ChannelLab

ChannelLab is a powerful, but dated commercial single ion channel modeling program developed over the last 11 years by Synaptosoft, Inc. [18]. It is available only for Microsoft Windows.

General observations

Unlike the other two simulators reviewed here, ChannelLab is not free; an individual license costs \$250. While the software is quite old (the last version was released in 2003), it contains many useful features, such as curve fitting and analysis tools, and data conversion utilities. Over 100 published channel models are included as examples.

Nevertheless, the interface does appear dated, especially when adding transitions between states. Additionally, ChannelLab lacks any scripting support.

Model definition

ChannelLab supports up to 20 states, with each state having an adjustable gating parameter. To add connections between states, the user can either use the 2D grid layout (Figure 2.8), or the 3D table layout (Figure 2.9). Both layouts are decidedly less intuitive and more constraining than the other simulators mentioned here.

Rate constants support dependence on up to six drugs or voltages, although only the user is limited to exponential equations.

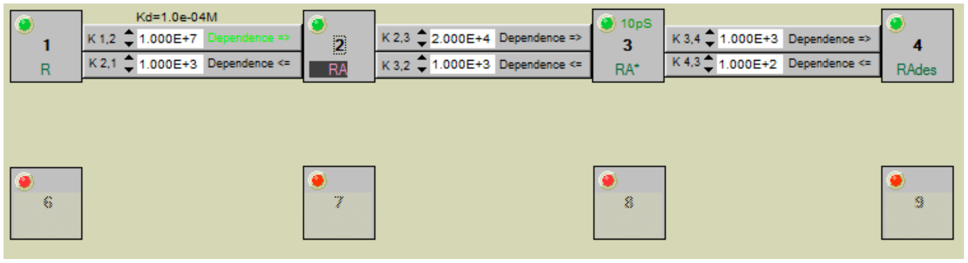
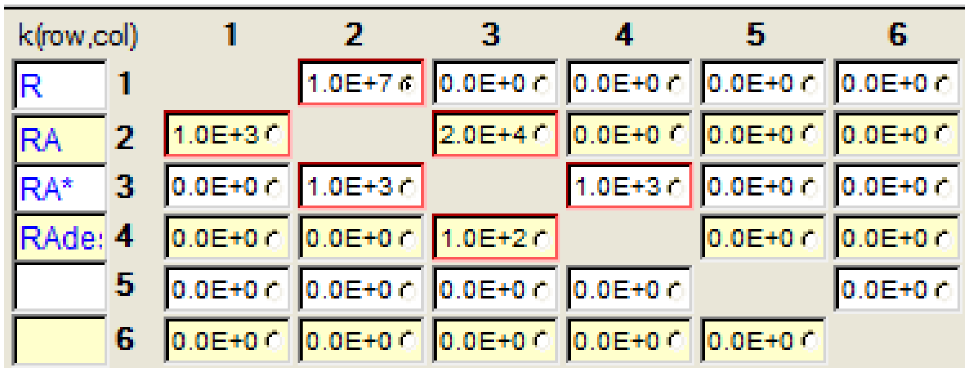

Kd=1.0e-04M
1
R
K 1,2 1.000E+7 Dependence =>
K 2,1 1.000E+3 Dependence <=
2
RA
K 2,3 2.000E+4 Dependence =>
K 3,2 1.000E+3 Dependence <=
10pS
3
RA*
K 3,4 1.000E+3 Dependence =>
K 4,3 1.000E+2 Dependence <=
4
RAdes
6
7
8
9

Figure 2.8: ChannelLab 2D grid layout of a four state model connected with six rate constants.

Stimulus definition

The software provides a powerful stimulus definition window, as shown in Figure 2.10. Concentration and voltage protocols are defined using automatic step functions, man-

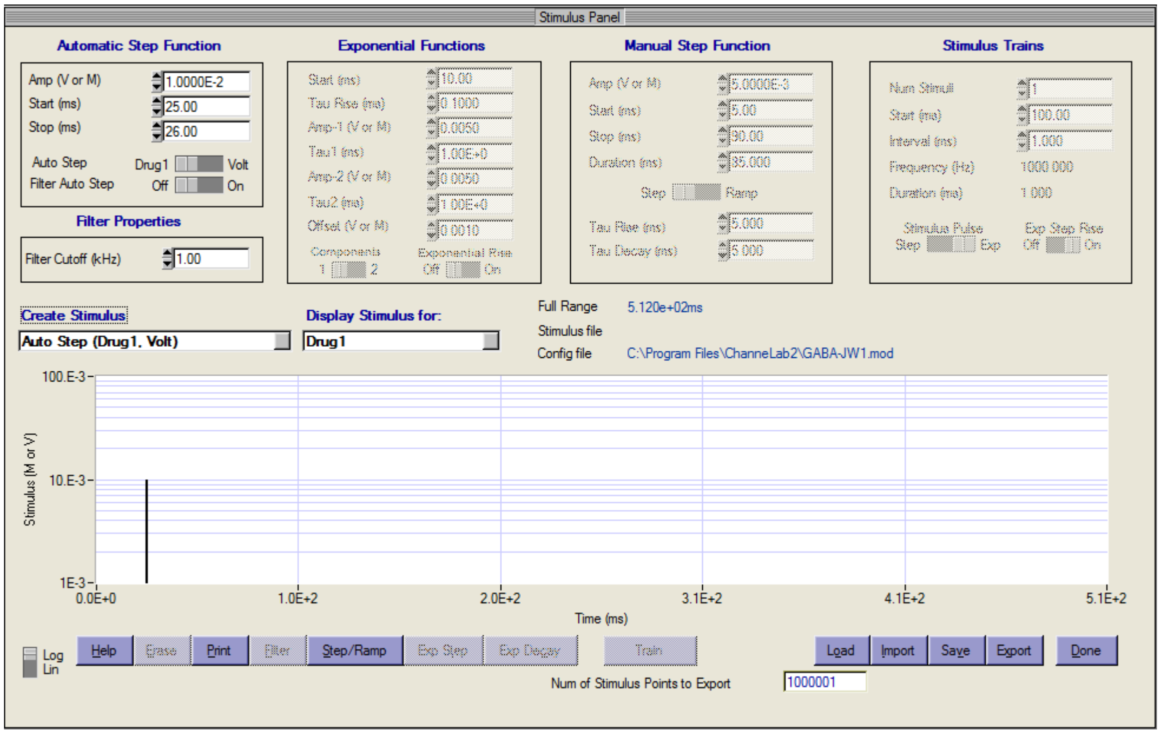


k(row,col)		1	2	3	4	5	6
R	1		1.0E+7	0.0E+0	0.0E+0	0.0E+0	0.0E+0
RA	2	1.0E+3		2.0E+4	0.0E+0	0.0E+0	0.0E+0
RA*	3	0.0E+0	1.0E+3		1.0E+3	0.0E+0	0.0E+0
RAde:	4	0.0E+0	0.0E+0	1.0E+2		0.0E+0	0.0E+0
	5	0.0E+0	0.0E+0	0.0E+0	0.0E+0		0.0E+0
	6	0.0E+0	0.0E+0	0.0E+0	0.0E+0	0.0E+0	

Figure 2.9: ChannelLab 3D table layout of the same model shown in Figure 2.8.

ual step or ramp functions, exponential functions, or stimulus trains. The protocols are plotted in the stimulus definition window using either linear or log scaling in order to provide immediate feedback to the user.

Channel lab also allows for the import stimulus and voltage waveforms from ascii text files.



Stimulus Panel

Automatic Step Function
Amp (V or M) 1.0000E-2
Start (ms) 25.00
Stop (ms) 26.00
Auto Step Drug1 / Volt
Filter Auto Step Off / On

Filter Properties
Filter Cutoff (kHz) 1.00

Exponential Functions
Start (ms) 10.00
Tau Rise (ms) 0.1000
Amp-1 (V or M) 0.0050
Tau1 (ms) 1.00E+0
Amp-2 (V or M) 0.0050
Tau2 (ms) 1.00E+0
Offset (V or M) 0.0010
Components 1 / 2
Exponential Rise Off / On

Manual Step Function
Amp (V or M) 5.0000E-3
Start (ms) 5.00
Stop (ms) 90.00
Duration (ms) 35.000
Step / Ramp
Tau Rise (ms) 5.000
Tau Decay (ms) 5.000

Stimulus Trains
Num Stimuli 1
Start (ms) 100.00
Interval (ms) 1.000
Frequency (Hz) 1000.000
Duration (ms) 1.000
Stimulus Pulse Step / Exp
Exp Step Rise Off / On

Create Stimulus: Auto Step (Drug1, Volt)
Display Stimulus for: Drug1
Full Range 5.120e+02ms
Stimulus file
Config file C:\Program Files\ChanneLab2\GABA-JW1.mod

Num of Stimulus Points to Export 1000001

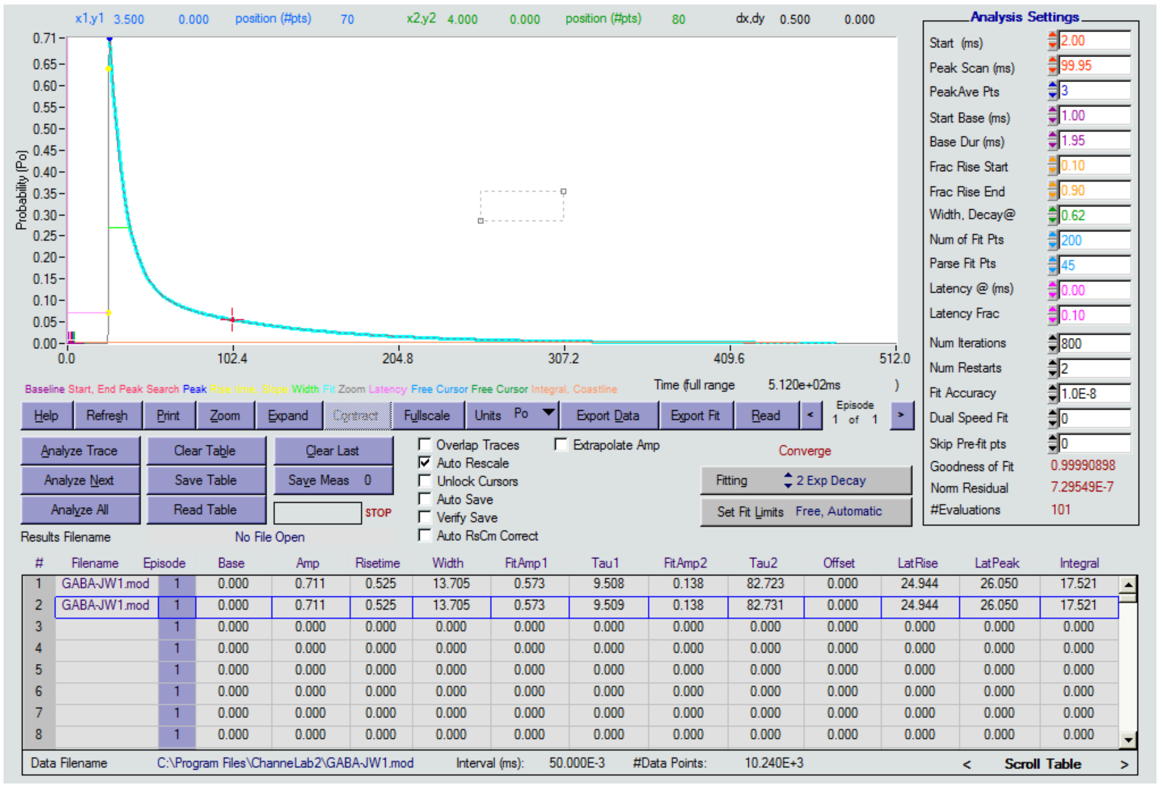
Figure 2.10: The ChannelLab stimulus window allows the user to create complex voltage and concentration protocols while providing immediate visual feedback.

Experiment definition

Three solvers are supported by ChannelLab: Runge-Kutta 4, Runge-Kutta 5, and Monte Carlo. As expected, the software allows the user to select the initial conditions as either the steady state probabilities, or user-provided values.

Plotting and analysis

The most attractive features of ChannelLab are the interactive analysis tools, which include maximum likelihood and least squares waveform fitting. A screenshot of the analysis window is provided in Figure 2.11. The plots in the analysis window are interactive and can be exported to a variety of file formats, but cannot be saved as vector graphics.



x1,y1 3.500 0.000 position (#pts) 70 x2,y2 4.000 0.000 position (#pts) 80 dx,dy 0.500 0.000

Time (full range 5.120e+02ms)

☐ Overlap Traces
☑ Auto Rescale
☐ Unlock Cursors
☐ Auto Save
☐ Verify Save
☐ Auto RsCm Correct
☐ Extrapolate Amp

Fitting 2 Exp Decay
Set Fit Limits Free, Automatic

Results Filename No File Open

Analysis Settings

Start (ms)	2.00
Peak Scan (ms)	99.95
PeakAve Pts	3
Start Base (ms)	1.00
Base Dur (ms)	1.95
Frac Rise Start	0.10
Frac Rise End	0.90
Width, Decay@	0.62
Num of Fit Pts	200
Parse Fit Pts	45
Latency @ (ms)	0.00
Latency Frac	0.10
Num Iterations	800
Num Restarts	2
Fit Accuracy	1.0E-8
Dual Speed Fit	0
Skip Pre-fit pts	0
Goodness of Fit	0.99990898
Norm Residual	7.29549E-7
#Evaluations	101

#	Filename	Episode	Base	Amp	Risetime	Width	FitAmp1	Tau1	FitAmp2	Tau2	Offset	LatRise	LatPeak	Integral
1	GABA-JW1.mod	1	0.000	0.711	0.525	13.705	0.573	9.508	0.138	82.723	0.000	24.944	26.050	17.521
2	GABA-JW1.mod	1	0.000	0.711	0.525	13.705	0.573	9.509	0.138	82.731	0.000	24.944	26.050	17.521
3		1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4		1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5		1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6		1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7		1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8		1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Data Filename C:\Program Files\ChanneLab2\GABA-JW1.mod Interval (ms): 50.000E-3 #Data Points: 10.240E+3

Figure 2.11: ChannelLab analysis window. Note the large number of adjustable analysis settings to the right.

Chapter 3

ModFossa Software

3.1 Overview

We present ModFossa, a fast and easy-to-use Python library for creating and simulating ion channel kinetics using continuous time Markov processes. A discussion of ModFossa's features, components, design methodology, and validation are provided in this chapter. Detailed class- and function-level documentation for the both the C++ core, and the Python library can be found in the Appendix.

For those interested, the name ModFossa was formed from two Latin words — mod, and fossa, which can be interpreted as *open channel* when written together.

3.2 Features

3.2.1 Python interface

ModFossa provides easy, yet versatile and powerful ion channel modeling through its novel Python interface. Python is well-used in the scientific and academic community; however, we are unaware of any Python libraries which facilitate the creation and simulation of ion channel kinetics. The popularity of Python as a research tool stems from its simple syntax and extensive standard library, and the availability of many high-quality open source numeric and scientific libraries. Additional qualities of Python include easy third party library installation; strong, dynamic typing; and an interactive interpreter.

ModFossa can also be used directly as a C++ library, allowing for more flexibility.

3.2.2 Easy model creation

ModFossa allows the user to define an ion channel's Markov model using states, rates, and connections. States and rates are referred to by their user-supplied names. This allows the user to create the model in any order. For example, connections can be defined before the definition of the states and rate constants.

States

States represent the various channel states as defined by a continuous time Markov chain. All states are referred to by unique name which is assigned by the user during creation. States are created as either non-conducting or conducting. Additionally, the gating of conducting states can by specified during construction. The gating parameter is used to simulate states that are partially conducting. For example, assigning a gating value of 0.5 to a state in a particular model may be used to simulate channel blocking by 50 percent.

Rates

Rates represent the kinetic rate constants which define the rate of transition between two states. Rates are referred to by unique name. ModFossa includes the following rate constant types: constant, Boltzmann voltage dependence, exponential voltage dependence, and ligand-gated. The equations for each rate constant are presented below.

Constant rate constant are defined by a single parameter, k, like so:

$$rate = k. \tag{3.1}$$

Exponential voltage dependent rate constants depend exponentially on the membrane voltage:

$$rate(V) = a * exp(k * V), \tag{3.2}$$

where V is the membrane voltage, exp is the exponential function, and a and k are parameters.

Boltzmann voltage dependence is defined using the sigmoid-like Boltzmann equation:

$$rate(V) = \frac{a}{1 + exp[(V - V_{0.5})/k]}, \tag{3.3}$$

where V is the membrane voltage, $V_{0.5}$ is the half-maximal activation voltage (given as a parameter), exp is the exponential function, and a and k are parameters. Taken from Angermann et al. [3].

Ligand-gated rate constants depend on the concentration of a particular ionic species, S, like so:

$$rate([S]) = k * [S]^n, \tag{3.4}$$

where $[S]$ is the concentration of ligand S, n is the ligand power, and k is a parameter.

Connections

Connections define a transition from one unique state to another using a specified rate constant. States can have multiple ingoing and outgoing connections. Rate constants may also be used in multiple connections.

3.2.3 Experiment definition

An experiment consists of a channel Markov model, a voltage protocol, and an optional concentration protocol.

Voltage protocol

Voltage protocols are a fundamental technique in electrophysiology used to measure the current response of ion channels. In the laboratory, a small electrode is inserted into the cell, through the cell membrane. The electrode, and therefore the cell membrane, is held constant at a desired voltage for a length of time, as defined by the voltage protocol. A typical voltage protocol is shown in Figure 3.1. As shown in the figure, the voltage protocol begins with a *hold stage* at -50 mV for 100 ms. Next, a 1 second long *stepped stage* is defined, which steps the voltage from -100 mV to 140 mV in 20 mV increments. The voltage protocol ends with another hold stage at -80 mV.

The voltage protocol shown in Figure 3.1 is easily defined in ModFossa by adding two hold stages, and one stepped stage. A ModFossa voltage protocol can have any number of hold stages, but only one stepped stage.

Concentration protocol

A concentration protocol defines the concentration values (extracellular or intracellular) of a certain ligand. Like a voltage protocol, ModFossa's concentration protocols are defined using *hold stages*. However, there is no support for stepped concentration protocol stages. During simulation, the simulator will run the model through the entire voltage protocol using a single stage from the concentration protocol. Next, the

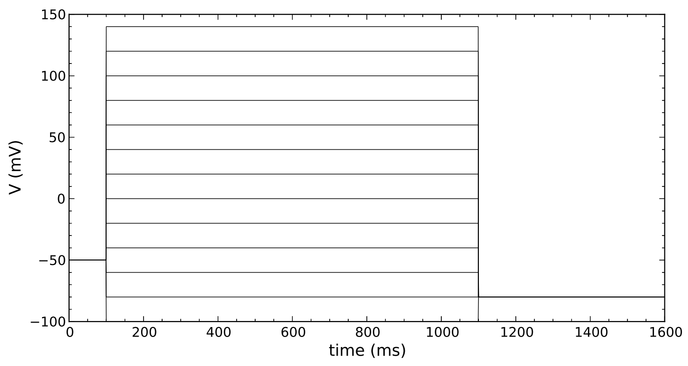


Figure 3.1: Example of a voltage protocol plot generated using ModFossa. This protocol starts with a hold stage at -50 mV for 100 ms. Next, a 1 second long *stepped stage* is defined, which steps the voltage from -100 mV to 140 mV in 20 mV increments. The voltage protocol ends with another hold stage at -80 mV.

concentration protocol will advance to the next stage, and the voltage protocol will be run using the new concentration value. This continues until there are no stages left in the concentration protocol.

3.2.4 Data analysis and plotting

ModFossa supports the creation of a number of useful plots using the Python plotting library, PyPlot. Unlike other simulators reviewed in Chapter 2, the plots can be saved easily in a vector graphics format. Visual examples of each plot type are found in Chapter 4.

Model and experiment validation

To assist the user in creating a valid simulation, ModFossa provides validation methods which return detailed messages regarding any problems with the model and experiment definitions. Specific error conditions include: *no connections defined*, *max conductance not defined*, *rate constant not defined*, *state not defined*, *ligand not defined*, *no voltage protocols defined*, and *no experiment sweeps defined.*

3.3 Design

A class diagram showing ModFossa's classes and their associations is provided in Figure 3.2. The organization of the classes in the diagram is as follows: the left side of diagram contains classes related to the model definition, including rates, states, connections, and the transition matrix.

The bottom right section of the diagram is centered around the SimulationRunner class, which is top level class of ModFossa, and is the point of entry for the Python interface (developed using Boost.Python). Also shown are the Simulator and ODESolver classes, which use the numeric solver, Sundials, to solve the differential equations that describe the evolution of state probabilities. The Results class stores the results for each simulation in a map structure, for later access by the user.

The remaining classes in the diagram are related to the Experiment class and the stimulus definition.

3.4 Implementation

3.4.1 Development tools

C++ code was developed under Ubuntu Linux using Eclipse CDT. CMake was chosen as the build system. Unit tests for the C++ code were written using the Google Test framework. Having unit tests available aided the development process by instilling confidence in the correctness of the code, especially during refactoring. Google Test is included in the source tree, and is built by CMake during the build process. This removes the need for the user to have Google Test installed on the target computer.

Doxygen was used to generate documentation for the C++ code, while the Python documentation was generated using Sphinx. Generated documentation can be found in the Appendix.

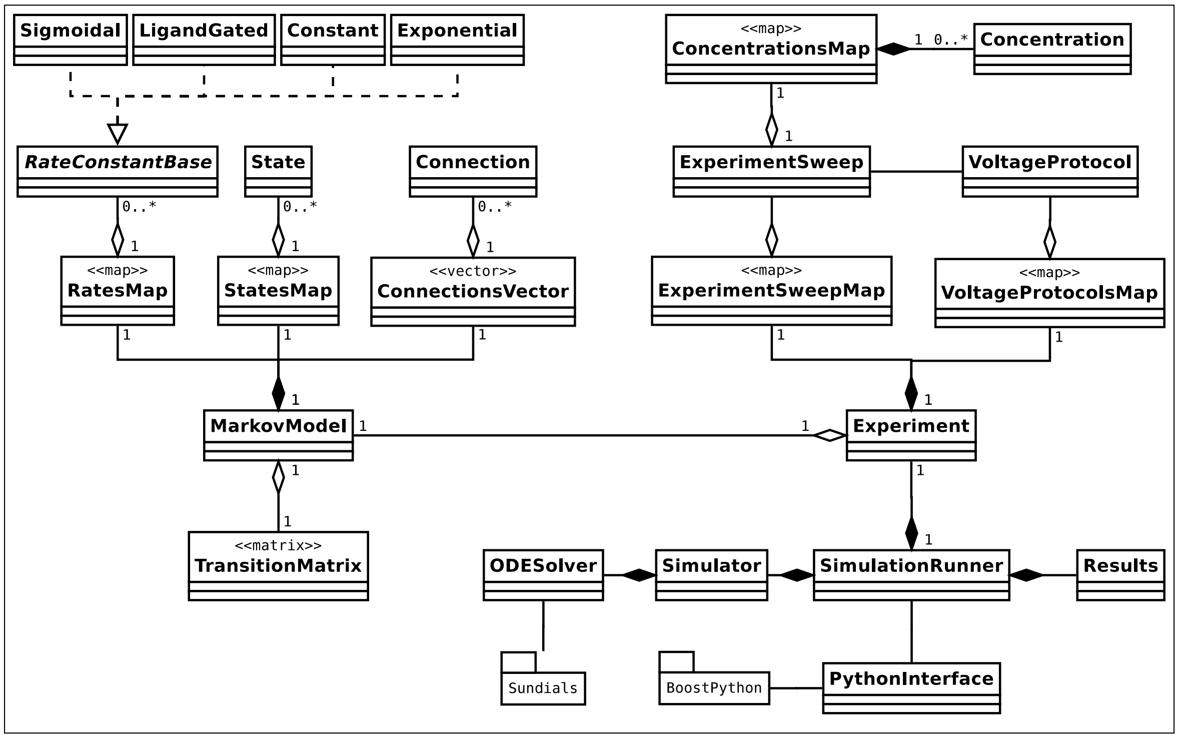


Figure 3.2: ModFossa C++ library class diagram. The classes on the left side of the diagram are related to the model definition. Classes relating to the experiment and stimulus definition are grouped to the right. Boost.Python is used to provide a Python interface to the library. Additionally, the numeric solver library Sundials provides the ordinary differential equations solvers required to simulate the channel model.

3.4.2 Dependencies

Sundials (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) is a C library developed by Lawrence Livermore National Laboratory. ModFossa uses Sundials to solve the system of ordinary differential equations governing the ion channel state probabilities. Armadillo, a C++ linear algebra library provides easy-to-use matrix structures and linear solvers, and is also used by ModFossa. However, it would be ideal to remove the dependency on Armadillo in order to remove complications during ModFossas installation.

In order to run ModFossa's Python interface, two Python libraries are required: MatplotLib and Numpy.

Chapter 4

Results

Several example models of varying complexity have been created and simulated using ModFossa. A description of the each model is provided, along with a listing of the Python source code used define the model. Visual results are provided in the form of plots generated by the code. Finally, an examination of ModFossa's computational performance is provided.

4.1 Simple models

Simple two and three state models are presented in order to give the reader an introduction to creating Markov models using ModFossa's Python interface. The simple models in this section have little biological relevance; for results from a published calcium gated chloride ion channel model, see Section 4.2. This section follows a tutorial format, with explanations provided alongside a sequence of short source code snippets.

4.1.1 Two State Model

The first example provided is a simple ion channel model with two states, Open, and Closed (Figure 4.1). The model defines a transition from the Open state to the Closed state with a rate k_{off} of 1 s^{-1}, and a transition from the Closed state to the Open state with a rate k_{on} of 10 s^{-1}. We define the initial state of the model as the Closed state.

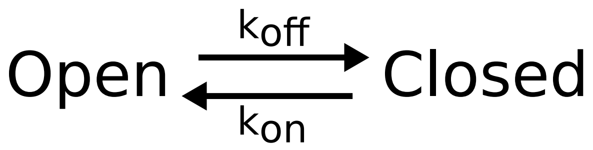


Figure 4.1: A two state Markov model containing two states, 'Open', and 'Closed.' The arrows between the states represent transition rates.

The two state model is implemented using ModFossa's Python interface. Let's examine the source code in detail. The first step is to import the library:

```python
from modFossa import *
```

Then we define our two states and name them 'Open' and 'Closed', and make 'Open' a conducting state. Note that although we are defining 'Open' as conducting, it will have no effect on the results presented for the two state model, because we are not calculating the channel current or conductance, only state probabilities.

```python
state('Open', conducting=True)
state('Closed')
```

Next, the connections between the two states are created. ModFossa allows the user to define connections before the states or rate constants have been defined.

```python
connect(from_state='Closed', to_state='Open', rate='kon')
connect(from_state='Open', to_state='Closed', rate='koff')
```

Now we define the rate constants, k_{on} and k_{off}, both of type 'constant'.

```
rate('kon', type='constant', k=10)
rate('koff', type='constant', k=1)
```

The last piece of information needed to describe the behavior of the model is the initial state, so we set it to closed. We also instruct ModFossa not to set the initial conditions of the simulation to the steady state probabilities, and to instead use the initial state, ‘Closed’ as the initial conditions. This allows us to observe the state probabilities of ‘Open’ and ‘Closed’ approach steady state, as shown in Figure 4.2.

```
initialState('Closed')
startAtSteadyState(False)
```

Although the simple two state model has no use for them, we must set the maximum channel conductance, and the reversal potential. These are required for the simulation to run.

```
maxChannelConductance(0) #mS/cm^2
reversalPotential(0)     #mV
```

Now we define our voltage protocol. We give it a name, ‘vp’, and add a single stage to it with the name ‘hold’. The ‘hold’ stage will hold the membrane potential at -50 mV for 500 ms. Next, an experiment sweep with the name ‘twoStateModelSweep’ is created. It used the voltage protocol we just defined, ‘vp’. In later examples we will use the experiment sweep to define the ionic concentrations in addition to the voltage protocol.

```
voltageProtocol('vp')
voltageProtocolAddStage('vp', 'hold',
  voltage=-50, duration=500)
experimentSweep('twoStateModelSweep', 'vp')
```

Finally, we can validate our model, run our experiment sweeps, and plot the results. We have only defined one experiment sweep, but if we had more they would run as well.

```
validate()
run()
plotStates('twoStateModelSweep')
```

Figure 4.2 shows the evolution of the state probabilities as a plot generated by ModFossa. At time $t = 0ms$, the model is the Closed state. As time advances, the probabilities move towards steady state. At $t = 500ms$, the probability of the Open state (rounded to three decimal places) is 0.905. Let's compare that to the analytical solution of the steady-state probability.

The differential equation describing the Open state probability is

$$\frac{dP_{Open}}{dt} = k_{on}(1 - P_{Open}) - k_{off}P_{Open}. \tag{4.1}$$

Steady-state occurs when the state probabilities are not changing, i.e., $\frac{dP_{Open}}{dt} = 0$. The steady-state Open probability is therefore

$$P_{OpenSS} = \frac{k_{on}}{k_{on} + k_{off}}. \tag{4.2}$$

Substituting in our rate constants from the two state model gives us

$$P_{OpenSS} = \frac{10}{10 + 1} = 0.909. \tag{4.3}$$

So, after 500 ms of simulation time, we are within 99.6% of the true steady-state probabilities.

4.1.2 Three State Model

Now let's create a more complex model that will show some features we can actually use to define real ion channel models, such as voltage steps, gating variables, and voltage and ligand gated rate constants. Figure 4.3 shows a Markov model with three

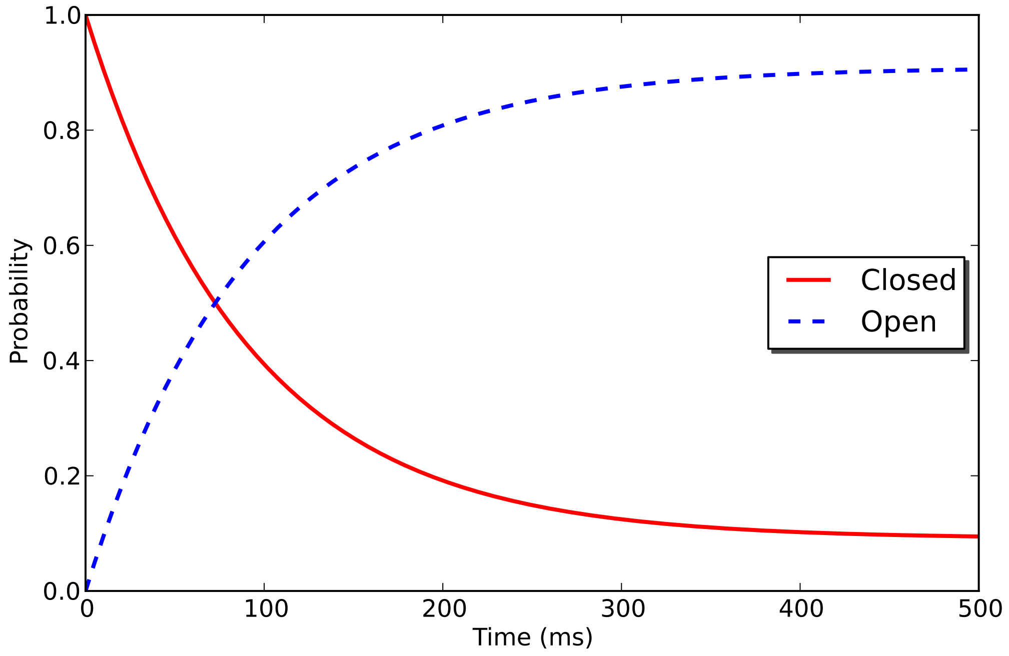


Figure 4.2: Evolution of the 'Open' and 'Closed' state probabilities of the simple two state model. At the initial conditions at time t=0, the system is entirely in the Closed state. As the simulation runs, the probabilities approach steady state.

states that we will use for this example. C_1 is a closed state, while O_1 is a partially conducting state with a gating variable of 0.5, and O_2 is a fully conducting, open state. The rate constants for the model are provided in Table 4.1.

$$\begin{tikzcd} C \arrow[r, shift left, "k_1"] & O_1 \arrow[l, shift left, "k_3"] \arrow[r, shift left, "k_2"] & O_2 \arrow[l, shift left, "k_4"] \end{tikzcd}$$

Figure 4.3: Three state Markov model. The states 'C', 'O1', and 'O2' are connected with the transition rates k1-k4.

Table 4.1: Rate constants used in the three state model example.

Name	Type	Parameters	Units
k_1	ligand gated	$k = 10e^8$, ligand $= Ca^{2+}$, power=1	$M^{-1}s^{-1}$
k_2	constant	k=1	s^{-1}
k_3	sigmoidal voltage gated	k=1, a=100, $V_{0.5} = 50$	s^{-1}
k_4	constant	k=4	s^{-1}

Let's examine the code for the three state model. First, we define our states and their gating variables. The closed state does not conduct, so we do not have to specify its gating variable (it is 0 by default). State O_1 is a partially conducting state, and O_2 is fully conducting.

```
from modFossa import *
state('C')
state('O1', gating=0.5)
state('O2', conducting=True)
```

Next, connect the states.

```
connect(from_state='C', to_state='O1', rate='k1')
connect(from_state='O1', to_state='O2', rate='k2')
connect(from_state='O1', to_state='C', rate='k3')
connect(from_state='O2', to_state='O1', rate='k4')
```

Now we define our rate constants. We have a ligand gated rate constant which depends on the concentration of calcium. This could be any ligand that we chose, but we must define its concentration in the experiment sweep, or ModFossa will give us an error.

```
rate('k1', type='ligandGated', k=10e8, ligand='Ca', power=1)
rate('k2', type='constant', k=1)
rate('k3', type='sigmoidal', k=1, a=100, v_half=50)
rate('k4', type='constant', k=4)
```

We must supply the remainder of the model definition. This is similar to the previous two state model example.

```
initialState('O2')
startAtSteadyState(False)
maxChannelConductance(1.16)
reversalPotential(0)
membraneCapacitance(100)
```

Next, define the voltage protocol and experiment sweep. The voltage protocol holds the voltage at -50 mV for 500 ms, then depolarizes the membrane by jumping to 50 mV and holding for another half of a second. The concentration of calcium is set to 500 nM in the experiment sweep. This is necessary because k_1 is a ligand gated rate dependent on $[Ca]$.

```
voltageProtocol('vp')
voltageProtocolAddStage('vp', 'hold1', voltage=-50,
                                duration=500)
voltageProtocolAddStage('vp', 'hold2', voltage= 50,
                                duration=500)
experimentSweep('three_state', 'vp', Ca=500e-9)
```

Finally, validate and run the experiment and plot the results.

```
validate()
run()
states = plotStates('three_state')
```

Figure 4.4 shows the results from the three state model. One again, we did not start in steady state, so we see the state probabilities moving towards steady state under the experiment sweep conditions of a -50 mV membrane voltage, and 500 nM $[Ca]$. At 500 mS, the voltage protocol stage ‘hold2’ is activated, and the voltage is increased immediately to positive 100 mV. The effect of this jump in voltage is evident in Figure 4.4. The jump in the probability of state O_1 is due to a decrease in the value of the sigmoidal rate constant, k_3.

4.2 Angermann CaCl currents model

Our final example presents an implementation of the calcium-activated chloride channel model developed by Angermann et al. [3]. A brief description of the study is provided, along with the model’s description and implementation details. Later, the Python source code used to implement the model in ModFossa is listed, along with several figures and accompanying discussion.

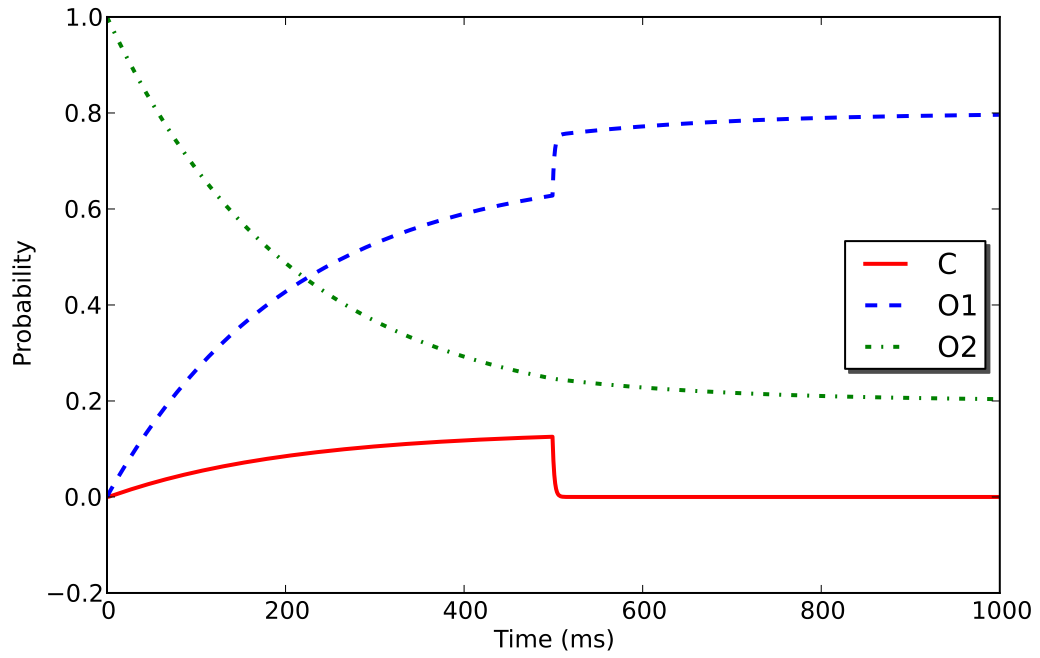


Figure 4.4: A plot showing the evolution of the three state model state probabilities as generated by ModFossa.

4.2.1 Study details

In [3], Angermann et al. attempt to determine how phosphatase activity influences calcium-activated chloride channels in rabbit pulmonary artery myocytes (smooth muscle cells). Whole cell patch clamp experiments were conducted to measure the calcium-activated chloride current (I_{ClCa}) under intracellular [Ca^{2+}] ranging from 20 mM to 1000 mM. The control group contained 3 mM ATP in the pipette solution, while the test group used 3 mM adenosine $5' - (\beta, \gamma - imido)$-triphosphate (AMP-PNP).

Under both groups, the maximum I_{ClCa} is greater at higher intracellular Ca^{2+} concentrations, as is expected for such channels. However, the currents in test group (containing AMP-PNP) exhibited behavior indicative of a negative shift in voltage dependent activation. The maximal conductance of I_{ClCa} was more than three-fold larger in the test group. Additionally, the activation times were less and the deactivation time greater in the test group compared to the control. These observations led the authors to the conclusion that the increased I_{ClCa} activity in the test group (pipette containing AMP-PNP) is not explained by Ca^{2+} channel sensitivity, nor the number of Ca^{2+} activating the channel, but instead by a negative shift in the voltage dependent activation.

4.2.2 Model description

To simulate the behavior of the test and control groups, Angermann et al. created a Markov chain kinetic model based on the work by Kuruma and Hartzell [12]. Slightly different parameters were chosen for the test and control groups in an attempt to make clear the difference in behavior between the two. The model, shown in Figure 4.5, consists of four closed states and three open states. Calcium binding occurs in the closed states with rates directly proportional to $[Ca]$. State C_1 is representative of the channel's state when no Ca^{2+} ions are bound to the receptor sites, while state C_4 represents the channel with all three Ca^{2+} binding sites occupied. The closed states with at least one binding site occupied can transition into open states with constant channel opening rates. Closed states with more occupied binding sites have higher values for the channel opening rates, giving the model its concentration-dependent behavior. Voltage dependent channel closing rates take the model from open to closed states.

In [3], the behavior of the channel in the presence of AMP-PNP was simulated by setting the gating variables of all three of the channel's open states to 1. Conversely, to simulate the channel in the presence of ATP, only channel O_1 was assigned a gating variable of 1, while the rest of the open states were given a value of zero. This reproduces the blocking effect by phosphorylation that the authors hypothesize as a possible explanation for the inhibition of I_{ClCa} while the channel is subjected to ATP. In addition to reducing the channel gating variables to model the effect of ATP, the authors also increased the magnitude of the channel closing rates and shifted the voltage dependence towards more positive potentials.

By altering the Markov model's parameters in an insightful manner, Angermann et al. were able to simulate the behavior of Ca^{2+}-activated Cl^- currents in the presence of both AMP-PNP and ATP.

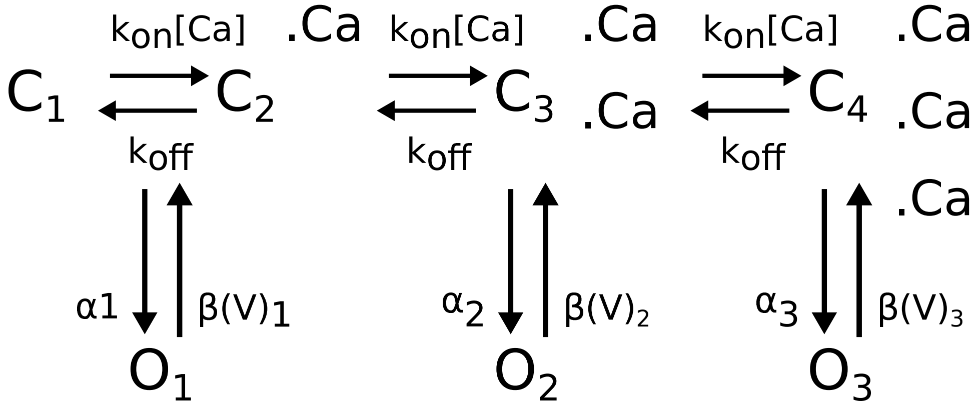


Figure 4.5: Angermann CaCl Markov model. The model has four closed states and three open states. State C2 represents the channel with a single Ca^{2+} binding site occupied, while state C4 represents the channel with all three binding sites occupied. Note that the model can only transition to an open state in the presence of at least one bound Ca^{2+}.

4.2.3 Model implementation in ModFossa

The calcium activated chloride channel Markov model shown in Figure 4.5 was implemented in ModFossa using the parameters in Table 4.2. The parameter values were chosen to match the behavior of I_{ClCa} in the presence of ATP, and were modified slightly from Thibeault et al. [3].

4.2.4 Code listing

The Python source code for the model is provided on the following page.

```
from modFossa import *

state('O1', conducting=True, gating=1.0)
state('O2', conducting=True, gating=0.5)
state('O3')
state('C1')
state('C2')
state('C3')
state('C4')

connect(fromState='C1', toState='C2', rate='kon')
connect(fromState='C2', toState='C3', rate='kon')
connect(fromState='C3', toState='C4', rate='kon')

connect(fromState='C4', toState='C3', rate='koff')
connect(fromState='C3', toState='C2', rate='koff')
connect(fromState='C2', toState='C1', rate='koff')

connect(fromState='C2', toState='O1', rate='a1')
connect(fromState='C3', toState='O2', rate='a2')
connect(fromState='C4', toState='O3', rate='a3')

connect(fromState='O1', toState='C2', rate='b1')
connect(fromState='O2', toState='C3', rate='b2')
connect(fromState='O3', toState='C4', rate='b3')

rate('koff', type='constant', k=50)
rate('kon', type='ligandGated', k=25e6, ligand='Ca', power=1)

rate('a1', type='constant', k=1)
rate('a2', type='constant', k=25)
rate('a3', type='constant', k=200)

rate('b1', type='sigmoidal', a=60, v_half=-40, k=40)
rate('b2', type='sigmoidal', a=35, v_half=0, k=50)
rate('b3', type='sigmoidal', a=25, v_half=140, k=40)

initialState('C1')
membraneCapacitance(100)

maxChannelConductance(1.16)
reversalPotential(0)

voltageProtocol('vp')
voltageProtocolAddStage('vp', 'hold', voltage=-50, duration=100)
voltageProtocolAddStage('vp', 'step', start=-100, stop=140, step=20, duration=1000)
voltageProtocolAddStage('vp', 'hold2', voltage=-80, duration=500)

concentrationProtocol('concentrations')
addConcentration('concentrations', Ca=20E-9)
addConcentration('concentrations', Ca=100E-9)
addConcentration('concentrations', Ca=250E-9)
addConcentration('concentrations', Ca=500E-9)
addConcentration('concentrations', Ca=750E-9)
addConcentration('concentrations', Ca=1000E-9)

experiment('angermann', 'vp', 'concentrations')
validate()
run()

# Plotting
currents = plotMultipleCurrents('angermann')
gVsV = plotGvsV('angermann', time_ms = 1099)
gVsCa = plotGvsConcentration('angermann', time_ms = 1099)

iv_late = plotMultipleIV('angermann',
                         time_ms = 1099, ymin = -10,
                         ymax = 50, labelHeight = 40)

iv_tail = plotMultipleIV('angermann',
                         time_ms = 1100, ymin = -30,
                         ymax = 30, labelHeight = 20)
```

Table 4.2: Parameters for Angermann CaCl model in presence of ATP. Reproduced with new values from [19].

	Value	Units
Conductance and equilibrium potential		
Maximal conductance	1.16	mS/cm^2
E_{Cl}	0	mV
Values of gating variables		
C_1	0	
$C_2.Ca$	0	
$C_3.2Ca$	0	
$C_4.3Ca$	0	
O_1	1	
O_2	0.5	
O_3	0	
Ca^{2+} binding rates		
$k_{on}(C_1 \rightarrow C_2.Ca)$	20×10^6	$M^{-1}s^{-1}$
$k_{on}(C_2.Ca \rightarrow C_3.2Ca)$	20×10^6	$M^{-1}s^{-1}$
$k_{on}(C_3.2Ca \rightarrow C_4.3Ca)$	20×10^6	$M^{-1}s^{-1}$
Unbinding rates		
$k_{off}(C_2.Ca \rightarrow C_1 + Ca)$	50	s^{-1}
$k_{off}(C_3.2Ca \rightarrow C_2.Ca + Ca)$	50	s^{-1}
$k_{off}(C_4.3Ca \rightarrow C_3.2Ca + Ca)$	50	s^{-1}
Channel opening rates		
$\alpha_1(C_2.Ca \rightarrow O_1)$	1	s^{-1}
$\alpha_2(C_3.2Ca \rightarrow O_2)$	25	s^{-1}
$\alpha_3(C_4.3Ca \rightarrow O_3)$	200	s^{-1}
Channel closing rates		
$\beta_1(V)(O_1 \rightarrow C_2.Ca)$		
where $a, V_{0.5}$, and $k =$	$60, -40$, and 40	s^{-1}, mV, mV
$\beta_2(V)(O_2 \rightarrow C_3.2Ca)$		
where $a, V_{0.5}$, and $k =$	$35, 0$, and 50	s^{-1}, mV, mV
$\beta_3(V)(O_3 \rightarrow C_4.3Ca)$		
where $a, V_{0.5}$, and $k =$	$25, 140$, and 40	s^{-1}, mV, mV

All $\beta_x(V)$ use the following Boltzmann equation form:
$\beta_x(V) = a/\{1 + exp[(V - V_{0.5})/k]\}$.

4.2.5 ModFossa output

Figure 4.6 shows the figure generated by ModFossa using the *plotCurrents* method. Each current trace shows the current passing through the channel at various Ca^{+2} levels. As expected, higher $[Ca^{+2}]$ invoke larger channel currents. The bottom item in the figure is the voltage protocol used in the experiment. Note that labels and calibration bars were added manually using Inkscape.

ModFossa can generate a number of IV curves at a given time, and place them on a single plot, as shown in Figure 4.7. This figure shows the *late IV* curves, measured at 1099 milliseconds into the simulation, which is the moment of maximum channel current. Similarly, Figure 4.8 shows the *tail IV* curves, which were measured at 1100 milliseconds.

To show the effects of both ligand concentration and voltage on the channel conductance, ModFossa's *plotGvsConcentration* method can be used to generate a chord conductance plot, as shown in Figure 4.9. This plot contains 13 traces of conductance as a function of the internal calcium concentration. Each trace corresponds to a different step value in the voltage protocol. As shown in Figure 4.6, the voltage protocol contains 13 stepped stages, hence the 13 traces in Figure 4.9.

A second chord conductance plot was generated using ModFossa (Figure 4.10). This plot shows channel conductance as a function of voltage, for 6 different concentration values. A chord conductance plot of Angermann et. al's measured experimental data is shown in Figure 4.11. As shown, the simulated data (Figure 4.10) match the experimental data (Figure 4.11) well.

4.2.6 Runtime Analysis

The runtimes of ModFossa versus a similar Matlab implementation are provided in Figure 4.12. ModFossa is approximately 17 times faster.

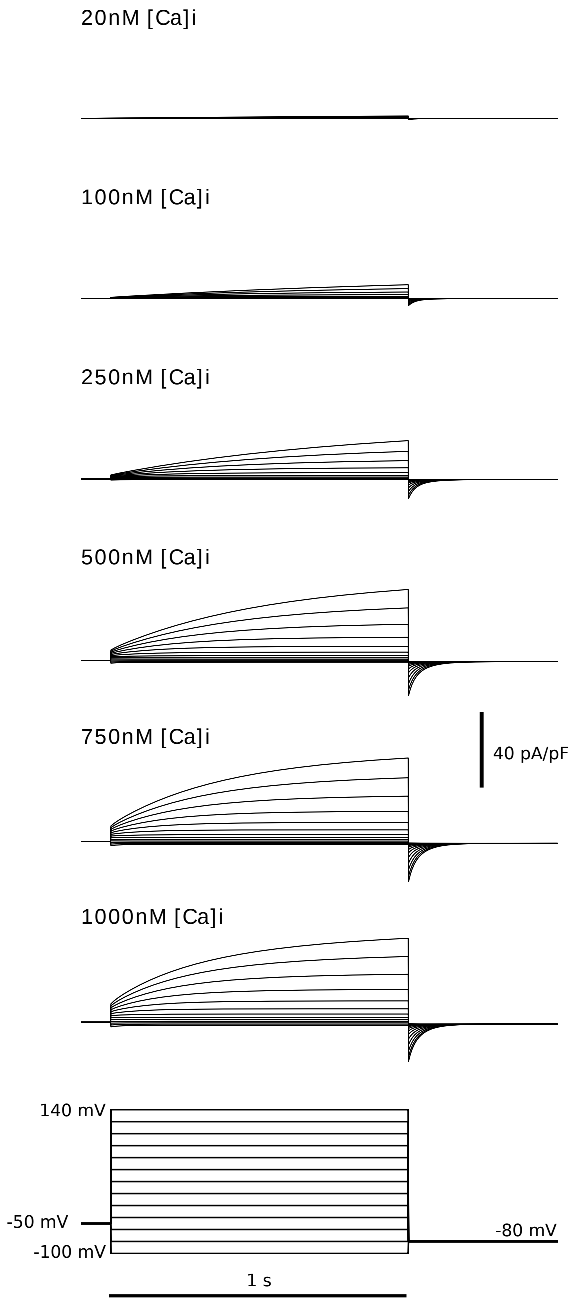


Figure 4.6: Figure generated by ModFossa showing Angermann CaCl currents given various internal Ca^{+2} concentrations. The voltage protocol is shown at the bottom of the figure.

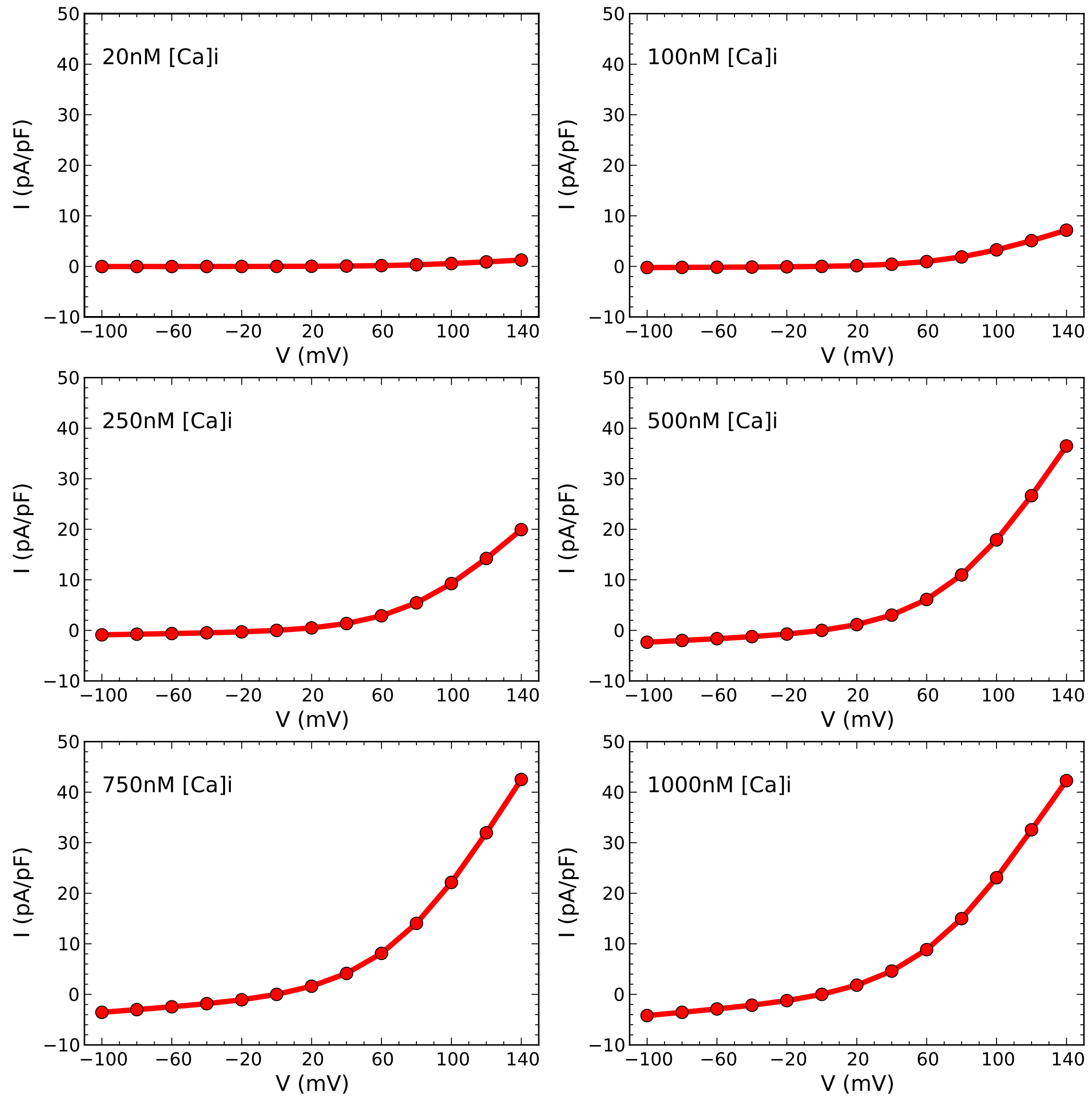


Figure 4.7: Angermann CaCl late IV curves given various internal Ca^{+2} concentrations. This figure was generated using ModFossa's *plotIV* method at time 1099 milliseconds.

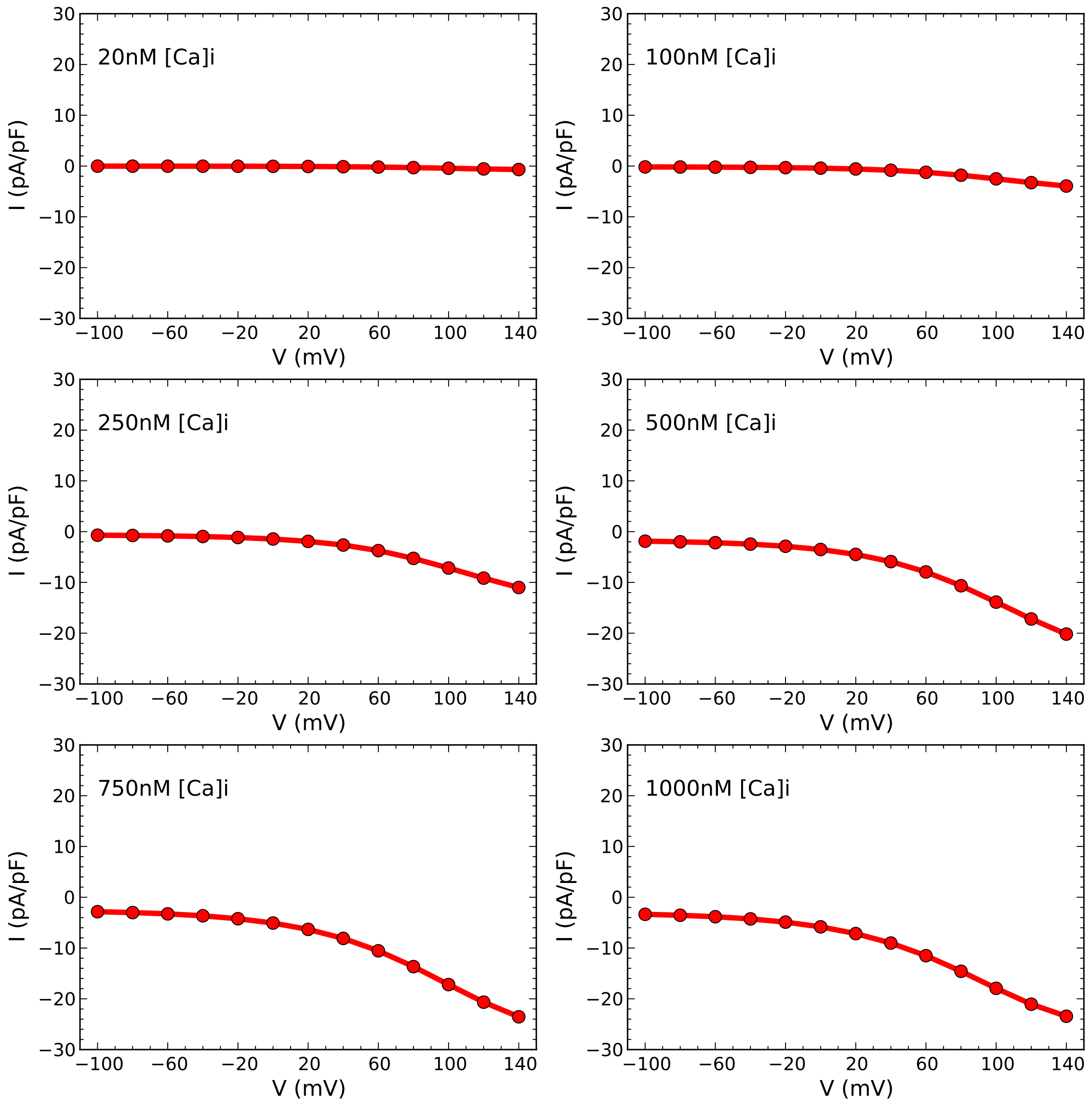


Figure 4.8: Angermann CaCl tail IV curves given various internal Ca^{+2} concentrations. This figure was generated using ModFossa's *plotIV* method at time 1100 milliseconds.

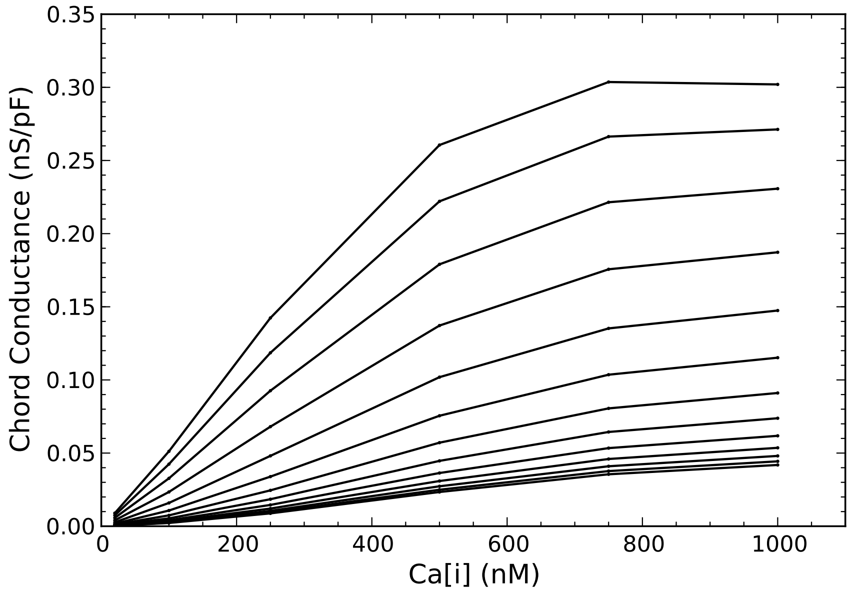


Figure 4.9: Angermann CaCl conductance at varying voltages as a function of intracellular Ca^{+2} concentration. There are 13 traces on the plot, which correspond to the 13 different steps in the voltage protocol. This figure was generated using ModFossa's *plotGvsConcentration* method.

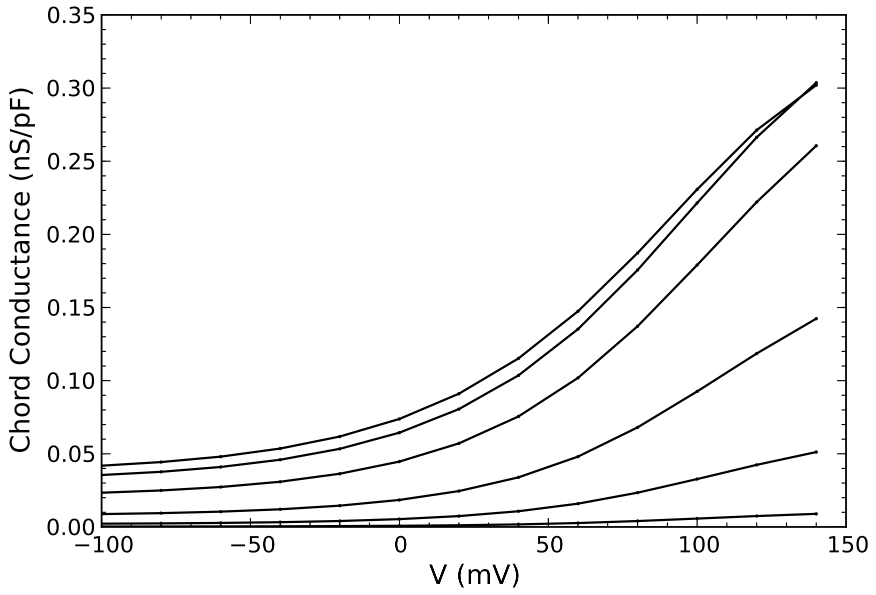


Figure 4.10: Simulated Angermann CaCl conductance at varying concentrations, as a function of membrane voltage. This figure was generated using ModFossa's *plotGvsV* method.

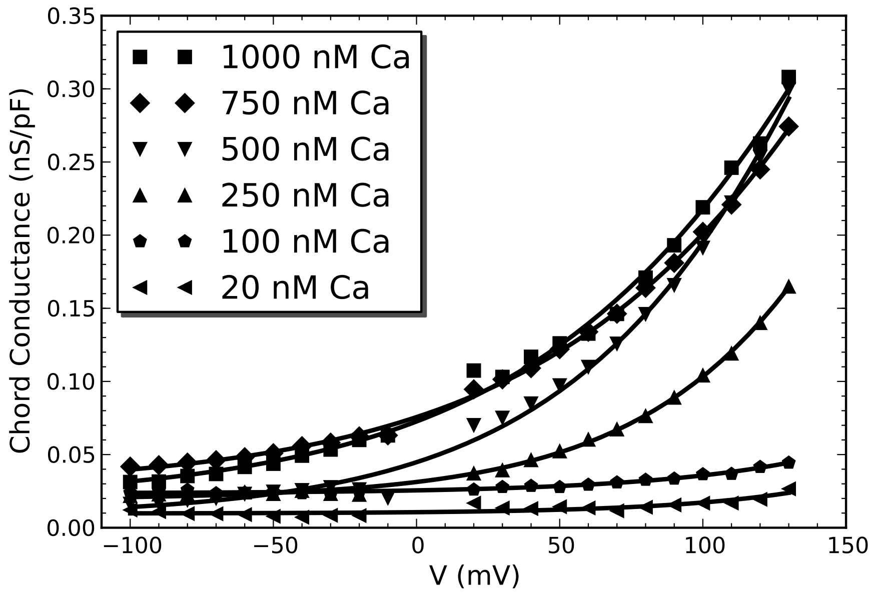


Figure 4.11: Measured data from [3] showing conductance as a function of voltage and concentrations. The data were fit to a Boltzmann function using the least squares method.

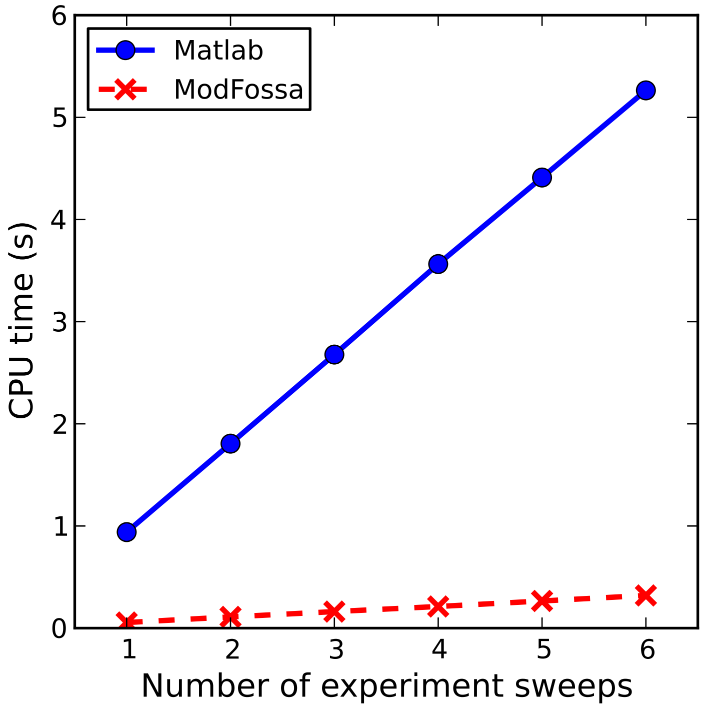


Figure 4.12: ModFossa runtime versus a Matlab implementation. The model and voltage protocol used are described in Section 4.2. The x-axis corresponds to the number of experiment sweeps performed at each data point.

Chapter 5

Discussion

5.1 Summary

We have identified and addressed a gap in ion channel simulation software by developing ModFossa, a Python library dedicated to the construction and simulation of ion channels based on continuous time Markov processes. While several recent software tools have been released for this purpose, none are Python libraries. Python is a popular tool among research scientists, therefore we hypothesize that a versatile and easy-to-use ion channel simulation library has the potential for providing a valuable service to those developing and testing Markov models of ion channels. In summary, ModFossa allows for the creation of channel models using simple Python syntax. Several rate constant types are supported, including exponential and Boltzmann voltage dependent rates, as well as ligand-gated rates. Voltage and current protocols are defined easily. Some common plots including IV curves, state probabilities, conductance versus voltage, and conductance versus ligand concentration are generated using a single function call. Finally, ModFossas core is implemented in C++ using the Sundials differential equation solver, providing the benefit of fast execution times.

5.2 Applications

ModFossa can provide rapid model development and testing for researchers with varying programming skill. A common task in ion channel model development is parameter searching, which requires the modeler to find the rate constant parameters which best fit the experimental data. Because ModFossa is usable from Python, existing machine learning libraries such as scikit-learn can be leveraged to perform parameter searches automatically. This situation is where ModFossas fast execution speed provides a great benefit, as it is not uncommon for modelers to leave parameter searches running for several days, or weeks.

5.3 Future work

Several ideas for additional development of ModFossa are presented here. First, some minor enhancements such as the addition of more sample models, improved customization of plotting functions (plot size, legend, colors, data range, etc.), and standardization of the data structures used in the results module would add value to the software by ensuring a smooth experience for new users. Additionally, creating and distributing a Debian package for ModFossa would ease installation significantly.

Several preliminary users have expressed interest in user-defined rate contant equations. A custom rate equation could be written by the user in Python using either a decorator function, or by inheriting from a base class. During simulation, the simulator would call the user's custom rate equation, passing in the current state of the simulation (membrane voltagte, concentrations, time, etc.), and use the returned value as the rate for whichever transitions the user specified. The performance hit due to calling a Python function from C++ during the simulation would have to be quantified, but it is not expected to be significant.

Lastly, the idea of offering ModFossa as a web service with a rich graphical user interface has been explored. Doing so would encourage use from a wider audience, particularly from those with limited or no programming experience. Such a web service could be implemented using a Python web framework such as Flask. A JavaScript visualization package, such as d3.js could provide rich graph visualizations of the model, as well as the stimuli definitions and simulation results. Finally, a web site would allow like-minded scientists to collaborate and easily share results with the community.

Bibliography

[1] Acton, F. S. (1970). *Numerical Methods That (Usually) Work.* Harper and Row, NY.

[2] Anderson, D. F. (2011). Introduction to stochastic processes with applications in the biosciences. `http://www.math.wisc.edu/~anderson/605F13/Notes/StochBio.pdf`.

[3] Angermann, J. E., Sanguinetti, A. R., Kenyon, J. L., Leblanc, N., and Greenwood, I. A. (2006). Mechanism of the inhibition of Ca2+-activated Cl- currents by phosphorylation in pulmonary arterial smooth muscle cells. *The Journal of general physiology*, 128(1):73–87.

[4] Capener, C. E., Kim, H. J., Arinaminpathy, Y., and Sansom, M. S. (2002). Ion channels: structural bioinformatics and modelling. *Human molecular genetics*, 11(20):2425–2433.

[5] De Santiago-Castillo, J. A., Covarrubias, M., Sánchez-Rodríguez, J. E., Perez-Cornejo, P., and Arreola, J. (2010). Simulating complex ion channel kinetics with ionchannellab. *Channels*, 4(5):422–428.

[6] Fink, M. and Noble, D. (2009). Markov models for ion channels: versatility versus identifiability and speed. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1896):2161–2179.

[7] Gurkiewicz, M. and Korngreen, A. (2007). A numerical approach to ion channel modelling using whole-cell voltage-clamp recordings and a genetic algorithm. *PLoS computational biology*, 3(8):e169.

[8] Hille, B. (2001). *Ionic channels of excitable membranes.* Sinauer associates Sunderland, MA.

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

[10] Jones, S. W. (2006). Are rate constants constant? *The Journal of physiology*, 571(3):502–502.

[11] Keener, J. P. and Sneyd, J. (2009). *Mathematical physiology.* Springer.

[12] Kuruma, A. and Hartzell, H. C. (2000). Bimodal control of a Ca2+-activated Cl-channel by different Ca2+ signals. *The Journal of general physiology*, 115(1):59–80.

[13] Levitt, D. G. (1999). Modeling of ion channels. *The Journal of general physiology*, 113(6):789–794.

[14] Liebovitch, L. (1989). Testing fractal and markov models of ion channel kinetics. *Biophysical journal*, 55(2):373.

[15] Liebovitch, L. S., Scheurle, D., Rusek, M., and Zochowski, M. (2001). Fractal methods to analyze ion channel kinetics. *Methods*, 24(4):359 – 375.

[16] Lowen, S. B., Liebovitch, L. S., and White, J. A. (1999). Fractal ion-channel behavior generates fractal firing patterns in neuronal models. *Physical Review E*, 59(5):5970.

[17] McManus, O., Weiss, D., Spivak, C., Blatz, A., and Magleby, K. (1988). Fractal models are inadequate for the kinetics of four different ion channels. *Biophysical journal*, 54(5):859–870.

[18] Synaptosoft (Accessed May 9th, 2013). Channellab. `http://www.synaptosoft.com/Channelab/`.

[19] Thibeault, C., Wiwchar, M., Huebner, M., Ayon, R. J., Greenwood, I. A., and Leblanc, N. (2011). Allosteric and state-dependent interactions can explain the paradoxical block and stimulation of native and TMEM16a-induced calcium-activated chloride currents by anthracene-9-carboxylic acid. In *2011 Experimental Biology meeting, Washington D.C.*

[20] Uebachs, M., Schaub, C., Perez-Reyes, E., and Beck, H. (2006). T-type ca2+ channels encode prior neuronal activity as modulated recovery rates. *The journal of physiology*, 571(3):519–536.

[21] Wikipedia (Accessed May 9th, 2013). Cell membrane diagram. `http://en.wikipedia.org/wiki/File:Cell_membrane_detailed_diagram_edit2.svg`.

modFossa Documentation

Release 0.1

Gareth Ferneyhough

August 01, 2013

CONTENTS

Contents:

CHAPTER

ONE

MODFOSSA PACKAGE

1.1 `modFossa` Package

1.2 `experiment` Module

`addConcentration`(*cpName*, ***args*)

Add a concentration value to the concentration protocol *cpName*.

Parameters

- **cpName** – name of concentration protocol to append the value to. It must be declared first using `concentrationProtocol()`.
- **args** – required arguments specifying the ligand and its concentration. See example below for more information.

  The following example adds a calcium concentration of 200 nM to *myConcentrations*

  ```
  addConcentration('myConcentrations', Ca=200E-9)
  ```

`concentrationProtocol`(*name*)

Declare a new concentration protocol with the given name.

Concentration values must be added using `addConcentration()`

`experiment`(*name*, *voltageProtocolName*, *concentrationProtocolName*)

Create an experiment using the given voltage and concentration protocols.

For each concentration in the concentration protocol, the voltage protocol will be run through. This is equivalent to using the method `experimentSweep()` to add each concentration value manually.

Individual experiment sweep results can be obtained and plotted using the generated name *name_ligand_concentration*. Additionally, the experiment module provides a dictionary, *_experimentSweepNames*, keyed on the experiment name. The values in the dictionary are lists of the experiment sweep names associated with a given experiment.

For example,:

```
concentrationProtocol('concentrations')
concentrationProtocolAddStage('concentrations', Ca=20E-9)
concentrationProtocolAddStage('concentrations', Ca=100E-9)
experiment('myExperiment', 'myVoltageProtocol', 'concentrations')
validate()
run()
```

will run *myVoltageProtocol* using a Ca concentration of 20 nM, followed by a Ca concentration of 100 nM. The two experiment sweeps will be named *myExperiment_Ca_20E-9*, and *myExperiment_Ca_100E-9*. These two names can be used to plot the experiment sweeps using the *single* plotting methods, such as `plotSingleIV()` and `plotSingleCurrents()`.

The *multiple* plotting methods will plot all the experiment sweeps from the given experiment on a single plot. For example, the following snippet will plot the two individual IV curves from the experiment defined above on a single plot:

```
plotMultipleIV(myExperiment, time=1100)
```

`isValid``()`
Return the validity of the model.

This will return *true* after a successful call to `validate()`.

`run``()`
Run all experiments defined with `experiment()`.

When this method completes, the results will be available using the results module.

`startAtSteadyState``(`*value*`)`
Tell the simulator whether or not to start the simulation at steady state.

This is *true* by default. If set to *false*, the method `initialState()` in the markovModel module can be used to set the initial state at which to begin the simulation.

`validate``()`
Check the validity of the model and notify the user of any problems.

`voltageProtocol``(`*name*`)`
Declare a new voltage protocol with the given name.

Voltage protocol stages must be added using `voltageProtocolAddStage()`

`voltageProtocolAddStage``(`*vpName*, *stageName*, ***args*`)`
Add a voltage protocol stage voltage protocol protocol *vpName*. The stage can either be stepped, or constant.

Parameters

- **vpName** – name of voltage protocol to append the value to. It must be declared first using `voltageProtocol()`.
- **args** – required arguments specifying the voltage stage. Units are in milliVolts and milliSeconds. See example below for more information.

The following example adds a stepped voltage protocol stage with the name *step* to *vp* ::
voltageProtocolAddStage('vp', 'step', start=-100, stop=140, step=20, duration=1000)

The following example adds a constant voltage protocol stage with the name *hold* to *vp* ::
voltageProtocolAddStage('vp', 'hold', voltage=-50, duration=100)

1.3 `markovModel` Module

`connect``(`*fromState*, *toState*, *rate*`)`
Connect two states with the given rate constant.

`initialState``(`*name*`)`
Set the initial state.

For this to have any effect, `startAtSteadyState()` must be *false*.

maxChannelConductance(*conductance*)
Set the max channel conductance in nS.

membraneCapacitance(*capacitance_pf*)
Set the membrane capacitance in pF.

This command currently has no effect on the simulation and will be removed.

rate(*name*, ***args*)
Create a rate constant.

Parameters

- **name** – name of the rate constant. Must be unique.
- **args** – required arguments specifying the type and parameters of the rate constant.

The type of rate constant to create is specified by the *type* argument. This is required. The various rate constant types and their required arguments are listed below.

constant

A rate constant with with a single rate, not dependent on voltage or ligand concentration.

Required arguments:

- k

Implementation: $rate = k$.

Example:

```
rate('myConstantRate', type='constant', k=10)
```

boltzman

A voltage dependent rate constant using the Boltzman equation.

Required arguments:

- a
- v_half
- k

Implementation: $rate(V) = \frac{a}{1+exp[(V-V\_0.5)/k]}$.

Example:

```
rate('myBoltzmanRate', type='boltzman', a=1, v_half=-30, k=10)
```

exponential

An exponential voltage dependent rate constant

Required arguments:

- a
- k

Implementation: $rate(V) = a * exp(k * V)$.

Example:

```
rate('myExponentialRate', type='exponential', a=1, k=10)
```

ligandGated

A ligand gated rate constant

Required arguments:

- ligand
- ligand_power
- k

Implementation: $rate([ligand]) = k * [ligand]^{ligand\_power}$ where $[ligand]$ is the concentration of *ligand*.

Example:

```
rate('myLigandGatedRate', type='ligandGated', ligand='Ca', ligand_power=3, k=50)
```

`reversalPotential`(*reversalPotential*)
Set the reversal potential in mV.

`state`(*name*, *conducting=False*, *gating=1.0*)
Add a new state to the markov model.

Parameters

- **name** – name of the state. Must be unique.
- **conducting** – whether or not the state is a conducting state. False by default.
- **gating** – scales the conductance of the state. Only used *conducting* is True. Default value is 1.0.

1.4 `plotting` Module

`plotCurrents`(*experimentSweepName*)
Plot the current traces for a given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`plotGvsConcentration`(*experimentName*, *time_ms*)
Plot the channel conductance versus concentration for a given experiment sweep at the specified time in milliSeconds.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`plotGvsV`(*experimentName*, *time_ms*)
Plot the channel conductance versus membrane voltage for a given experiment sweep at the specified time in milliSeconds.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`plotIV`(*experimentSweepName*, *time_ms*)
Plot the IV curve for a given experiment sweep at the specifed time in milliSeconds.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`plotMultipleCurrents`(*experimentName*)
Plot all of the current traces for a given experiment.

A figure will be generated with several subplots, one for each experiment sweep in the experiment.

`plotMultipleIV`(*experimentName*, *time_ms*, *ymin*, *ymax*, *labelHeight*)
Plot all of the IV curves for a given experiment at the given time in millieSeconds.

A figure will be generated with several subplots, one for each experiment sweep in the experiment.

`plotStates`(*experimentSweepName*)
Plot the state probablities for a given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`plotVoltageProtocol`(*experimentSweepName*)
Plot the voltage protocol for a given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

1.5 `results` Module

`getConductances`(*experimentSweepName*)
Get the conductances for the given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`getCurrents`(*experimentSweepName*)
Get the currents for the given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`getIV`(*experimentSweepName*, *time_ms*)
Get the IV curve for the given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`getStateNames`()
Get the state names for the given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`getStateProbabilities`(*experimentSweepName*)
Get the state probabilities for the given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

`getVoltageProtocol`(*experimentSweepName*)
Get the voltage protocol for the given experiment sweep.

See `experiment()` in the experiment module on how experiment sweep names are generated for a given experiment.

INDEX