College of Engineering, Mathematics and Physical Sciences

Mathematical and Computational Study of Markovian Models of Ion Channels in Cardiac Excitation

Submitted by

Tomáš Starý

to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Mathematics.

June 2016

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

....

Tomáš Starý

Abstract

This thesis studies numerical methods for integrating the master equations describing Markov chain models of cardiac ion channels. Such models describe the time evolution of the probability that ion channels are in a particular state. Numerical simulations of such models are often computationally demanding because many solvers require relatively small time steps to ensure numerical stability. The aim of this project is to analyse selected Markov chains and develop more efficient and accurate solvers.

We separate a Markov chain model into fast and slow time-scales based on the speed of transitions between states. Eliminating the fast transitions, we find an asymptotic reduction of zeroth-order and first-order in a small parameter describing the time-scales separation. We apply the theory to a Markov chain model of the fast sodium channel I_{Na}. We consider several variants for classifying some transitions as fast in order to find reduced systems that yield a good accuracy. However, the time step size is still restricted by numerical instabilities.

We adapt the Rush-Larsen technique originally developed for gate models. Assuming that a transition matrix can be considered constant during each time step, we solve the Markov chain model analytically. The solution provides a recipe for a stable exponential solver, which we call "Matrix Rush-Larsen" (MRL). Using operator splitting we design an even more flexible "hybrid" method that combines the MRL with other solvers. The resulting improvement in stability allows a large increase in the time step size. In some models, we obtain reasonably accurate results 27 times faster using a hybrid method than with the forward Euler method, even with the maximal time step allowed by the stability constraint.

Finally, we extend the cardiac simulation package $\mathrm{BeatBox}$ by the developed exponential solvers. We upgrade a format of "`ionic`" modules which describe a cardiac cell, in order to allow for a specific definition of Markov chain models. We also modify a particular integrator for `ionic` modules to include the MRL and the hybrid method. To test the functionality of the code, we have converted a number of cellular models into the `ionic` format. The documented code is available in the official $\mathrm{BeatBox}$ package distribution.

Acknowledgement

This thesis would not have been possible without the continuous support I received from many people.

Firstly, I would like to express my deep gratitude to my supervisor Prof Vadim N. Biktashev for his excellent scientific supervision of my work. His expertise, understanding, guidance and constructive criticism were truly essential for the development of this inspiring project. Thank you, Vadim, for your assistance, time and patience.

Secondly, I extend my thanks to my colleagues at Exeter University for interesting discussions, tips on computer use, and explanation of various mathematical problems, especially to Paul Ritchie, Burhan Bezekci, Ildar Sadrev, Arjaree Saengsathien, Pierre Aumjaud, Thomas Mendlik, Ummu Atiqah Mohd Roslan, Clare Perryman, Damian Smug, Abdullah Aldurayhim, Saad Almuaddi, and to Stefan Siegert, Courtney Quinn, Adam Peddle, Lewis Watson, and Ian Wooley for proofreading of selected chapters of my thesis.

I would like to acknowledge the College of Engineering, Mathematics and Physical Sciences for the financial support which enabled me to undertake this project.

This thesis was developed based on freely licensed computer programs namely GNU, Linux, Emacs, LaTeX, Python, BeatBox, and others. I thank all the contributors for allowing anyone to use, study, modify and share the product of their work.

Finally, I would like to thank all my friends who made my life during this time even more enjoyable. I owe a great debt of gratitude to my parents, my sister, and the whole family for their lasting support and encouragement.

Publications

- T. Starý and V. N. Biktashev. **Exponential integrators for a Markov chain model of the fast sodium channel of cardiomyocytes.** *IEEE Trans. BME* arXiv:1411.6204.
- Tomáš Starý, Vadim Biktashev. **Evaluating Exponential Integrators for Markov Chain Ion Channel Models.** *IEEE Computing in Cardiology Proceedings.* 2015 42:885-888.
- Mario Antonioletti, Vadim N. Biktashev Adrian Jackson, Sanjay R. Kharche, Tomas Stary, Irina V. Biktasheva. BeatBox**— HPC Environment for Biophysically and Anatomically Realistic Cardiac Simulations.** Submitted to *PLoS ONE* arXiv:1605.06015.
- **The** BeatBox **Cardiac Simulation Software: A User's Guide**, modifications in `rushlarsen` device, `ionic` format and `ionic` modules, BeatBox Home Page

Presentations

- Tomáš Starý*, Vadim Biktashev. **Analysis of Numerical Methods for Markov Chain Models of Ionic Channels.** *British Applied Mathematics Colloquium*, Oxford, April 2016
- Tomáš Starý*, Vadim Biktashev. **Evaluating Exponential Integrators for Markov Chain Ion Channel Models.** Computing in Cardiology. Nice, September 2015
- Tomáš Starý, Vadim Biktashev*. **Evaluating Exponential Integrators for Markov Chain Ion Channel Models.** *Dynamics Days.* Exeter, September 2015
- Tomáš Starý*, Vadim Biktashev. **Practical methods of solving Markov chain type ion current models.** *BioDynamics workshop.* Exeter, June 2014
- Tomáš Starý, Vadim Biktashev*. **Dynamical Properties of Markov Chain Cardiac Ion Channel.** *Computational Cardiac Electrophysiology workshop.* Imperial College, May 2014
- Tomáš Starý*, Vadim Biktashev. **Practical methods of solving Markov chain type ion current models.** *British Applied Mathematics Colloquium.* Cardiff, April 2014

* *presenting author*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The area of application of this thesis is cardiac electrophysiology, specifically mathematical models of ion channels, that use master equations of continuous Markov chains. A Markov chain model of an ion channel consists of a finite number of states that correspond to conformations of the channel, which are either permissive or non-permissive for an ionic flow. These models are systems of linear ordinary differential equations (ODEs). Dynamic variables of the ODEs represent the probability that the channel resides in a particular state. Coefficients of the ODEs are the probabilities of transitions from one state to another, the coefficients are contained in a *transition matrix*.

Time evolution of dynamic variables is found using numerical solvers, however, many popular solvers, such as the simplest forward Euler, suffers from so-called numerical instabilities. The numerical instabilities appear as the time step size increases and cause the simulation to fail. Systems with fast processes, as is often the case for Markov chain models of ion channels, only provide numericaly stable results using very small time steps. The solution then requires a large number of arithmetic operations, hence, the computational cost of the solution increases. In this work we aim to tackle the issue of the numerical instability in certain Markov chain models to design more efficient, accurate solvers.

The thesis is divided into seven chapters. After this Introduction which describes the contents of the thesis, Chapter 2 gives some background information about the field of electrophysiology. We describe the full Hodgkin and Huxley cellular model, with an emphasis on the ion channels. Hodgkin and Huxley used so-called gate model, which is still a popular ion channel model, but sometimes fails to reproduce experimental data. Markov chain models address some of these limitations. We present equivalent Markov chain models to gate models and provide a description of a few modern cardiac Markov chain models, which will be used in subsequent chapters.

Chapter 3 describes the numerical solvers and asymptotic methods relevant to the project. Section 3.2 introduces the numerical solvers based on discretisation in the time domain. The system is then integrated by approximating the evolution of the dynamic variables at the discrete points in time. We describe the simplest forward Euler solver and the Rush-Larsen (1978) solver, which is designed specifically for gate models. The Rush-Larsen method is more accurate and stable, however, it cannot be applied to the Markov chains directly. Section 3.1 describes asymptotic methods for a reduction of the dimensionality of a generic Markov chain model in which some transition rates are much faster than others. The asymptotic reduction gives an approximation in terms of an order of a small parameter that characterises the time-scale separation. We use the approximation in a zeroth-order (also called leading-order) and a first-order in the small parameter.

Chapter 4 applies the dimensionality reduction described in Chapter 3 to a Markov chain of sodium channel I_{Na} by Clancy and Rudy (2002). To find processes for elimination we use a procedure called “transition rates embedding”. We select several variants for classifying some of the I_{Na} transition rates as fast and other as slow, and reduce the dimensionality by a zeroth-order, and in one case we also include the first-order correction. However, the test simulations of reduced I_{Na} model showed numerical instabilities at the same value of the time step as the original model. Hence, the dimensionality reduction failed to provide a significant speed-up of the computation in this case.

Chapter 5 describes the development of exponential integration methods and their application to Markov chain ion channel models. We apply those methods to an example of a stiff Markov chain of I_{Na} channel developed by Clancy and Rudy (2002) [2]. Using operator splitting methods we split I_{Na} into three subsystems. The first contains the fast transition rates at high voltages, the second contains fast transition rates at low voltages and the third contains uniformly slow transition rates. Both fast subsystems are coupled in a way that allows an analytical solution if we “freeze” (consider constant) the transition rates at their initial value, for the duration of the time step. The analytical solution provides a recipe for exponential integration. In a “hybrid” approach we build a numerical solver combining the exponential integration for fast subsystems and the forward Euler method for the slow subsystem.

In Chapter 5 we describe one further method called Matrix Rush-Larsen (MRL). The MRL “freezes” the transition rates matrix for the duration of the time step, analogously to the fast subsystems in the “hybrid” method, and integrate the whole Markov chain model, which gives a solution with a matrix exponential. We exponentiate the diagonal matrix found by an eigenvalue decomposition. Before the simulation starts we precompute the exponential operator for a particular time step and save it into a look-up table. During the computation we refer to the values in the look-up tables.

In Section 5.2, we apply the methods to two examples of Markov chain models in the Faber *et al.* (2007), which are known to require very small time steps in the forward Euler solver used in the authors' implementation of the model. In Section 5.3 we provide the formulas to analyse the accuracy of the numerical integration methods.

Chapter 6 describes the implementation of the exponential integration methods into a cardiac simulation package BeatBox. This task requires a modification of the BeatBox framework for cellular models – the so-called `ionic` format. Before our modifications, the `ionic` format used to distinguish between three types of dynamical variables: gating variables dependent on membrane voltage, gating variables dependent on other variables, and "other" variables. We extended the `ionic` format by a fourth type: Markov chain variables. We implemented the hybrid methods with the MRL for integration of Markov chains. We also include specific functions and macros for definition of Markov chain models into `ionic` models, which describe the cardiac cells, and modify the data structure of `ionic` modules, which contain the variables and parameters of the simulation. We test the functionality on `ionic` models and include Markov chain ion channels. The code and the documentation are made available along with the BeatBox distribution.

Finally, we conclude by discussing the main results, limitations and interesting directions for further work in Chapter 7.

Appendix A defines the cellular model that was used in the study by Clancy and Rudy (2002) [2] together with the I_{Na} Markov chain. Appendix B discusses the computation of eigenvalues and presents the C-code for the computation using GSL and LAPACK libraries. It also describes the extraction of required functions for the purposes of implementating the MRL into BeatBox. Appendix C presents the updated files in BeatBox distribution. Appendix D describes an implementation of some cellular modules into BeatBox.

Chapter 2

Cellular Electrophysiology

This chapter gives a brief introduction to cellular electrophysiology. Section 2.1 reviews the pioneering work of Hodgkin and Huxley (1952). Section 2.2 introduces cardiac models and describes some differences from the Hodgkin and Huxley work. Section 2.3 focuses on the description of the limitations of the Hodgkin-Huxley type gate models of ion channels and introduces the so-called Markov chain models that generalise the gate models and so allow to simulate more complex phenomena. Detailed description of the conversion from a gate model to a Markov chain model is shown using the gate models of Hodgkin and Huxley. Section 2.4 describes a few examples of popular Markov chain models.

2.1 The Hodgkin-Huxley Model

2.1.1 Model of a Cell

Electrophysiology is based on the study of excitable cells. The cells are bounded by a cellular membrane that separates the intra- and extracellular environments. Those environments contain an electrolyte, which is a water solution of electrically charged particles, such as ions of sodium and potassium. The ionic concentrations in intra- and extracellular space of the cell differ, which causes polarisation of the membrane.

Under certain conditions the membrane becomes permeable for certain types of ions, which then diffuse between the environments down the concentration and electric gradient. The permeability of the membrane to specific ions is determined by dynamical processes which enable flow of ions through the membrane. These ion flows constitute electric currents. Due to the changing amount of specific current an electric excitation called action potential is observed.

A breakthrough theory was introduced in 1952 by Hodgkin and Huxley who suggested a mathematical model for the electric processes in cells [3]. Although, there are more advanced models, we describe the Hodgkin-Huxley model here, as is a straightforward and simple model which shows the fundamental principles which are also present in modern cellular models.

Hodgkin and Huxley expressed the cellular membrane by a diagram as shown in Figure 2.1. The cellular membrane is a lipid layer that is not conductive and in an electric field acts as a capacitor. The permeability of the membrane is expressed as an electrical conductance specific for every kind of ion.

The membrane currents have to obey Kirchhoff's law. Treating the cell membrane as a node in an electrical circuit, Kirchhoff's law gives the relation

$$\sum_s I_s = 0, \tag{2.1}$$

where I_s represent the currents corresponding to the capacitor I_C, sodium I_{Na} potassium I_{K} and a small leakage current I_l due to chloride and other ions.

The relation of the capacitive current is obtained from the equation of the capacitor

$$I_C(t) = C\frac{\mathrm{dV_m}}{\mathrm{d}t} \tag{2.2}$$

where $\mathrm{V_m}$ is the membrane voltage corresponding to the potential difference between the intra- and extracellular environment. The expressions for sodium, potassium and leakage currents depend on the conductance of the membrane and the driving force for the type of ions, which is expressed in terms of Ohm's law as

$$I_{\mathrm{K}}(t) = g_{\mathrm{K}}(t)(\mathrm{V_m} - E_{\mathrm{K}}), \tag{2.3a}$$

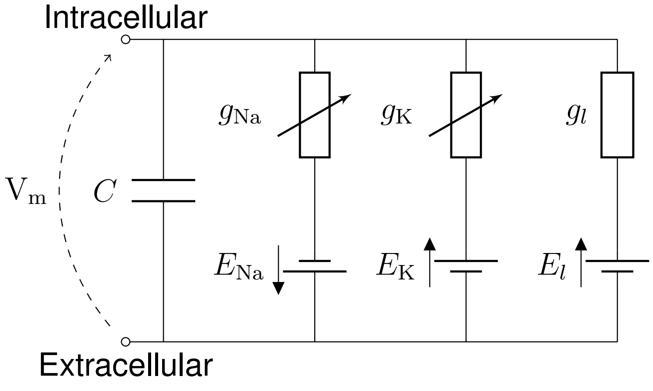


Figure 2.1: Electric diagram of the cellular membrane [3]. Each branch of the diagram represents a specific process in the cell. From the left to right it is the capacitance of the membrane, the sodium, the potassium and the leakage current.

$$I_{\mathrm{Na}}(t) = g_{\mathrm{Na}}(t)(\mathrm{V_m} - E_{\mathrm{Na}}), \tag{2.3b}$$

$$I_l(t) = g_l(\mathrm{V_m} - E_l) \tag{2.3c}$$

where g_{K} and g_{Na} and g_l quantify the permeability of the membrane, and E_{K}, E_{Na} and E_l are equilibrium potentials depending on the concentrations of particular ions. The equilibrium potentials are considered constant in the Hodgkin-Huxley model.

The equation for the membrane then gets the form

$$\frac{\mathrm{dV_m}}{\mathrm{d}t} = -\frac{1}{C}\left(I_{\mathrm{Na}} + I_{\mathrm{K}} + I_l\right). \tag{2.4}$$

2.1.2 Ionic Currents

The conductance of the membrane to specific ions cannot be derived from first principles. Instead, a theoretical model aims to describe the empirically observed phenomena in a neural cell. Hodgkin and Huxley performed experiments to measure ionic currents due to sodium and potassium ions through the membrane of a giant squid axons.

The conductance of potassium and sodium ions in equations (2.3a) and (2.3b) change in time. The conductance of potassium is described by

$$g_{\mathrm{K}}(t) = \bar{g_{\mathrm{K}}} n(t)^4, \tag{2.5}$$

where $\bar{g_{\mathrm{K}}}$ is the maximum conductance and n is a dynamical variable, that takes values between 0 and 1.

The permeability of the ionic membrane for currents is due to large protein molecules residing in the cellular membrane, which has been discovered some time after Hodgkin and Huxley developed that model. However, as it simplifies the description of a cellular model and it is more realistic, we will use a notion of ion channel model, which is known as a gate model.

The gate model assumes that the conformation of the channel molecule can be expressed by a finite number of gates, that can be either open or closed. When all gates are open, the channel allows ions to pass through. The number and type of gates differs in each type of channel. The potassium channel contains one type of gate. The probability that a gate is open is given by the dynamic variable n. In a single potassium channel there are four gates, hence the probability of all gates being open is n^4 (assuming independence between gates). The time evolution of n is described by the differential equation

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \frac{n_\infty - n}{\tau_n} \tag{2.6}$$

where n_∞ represents steady-state value of the variable n and τ_n is a characteristic time of potassium channel n, which in some literature called “time constant”, in the context of voltage-clamp experiments. However, in the context of dynamical system, this term is not appropriate, as it depends on the membrane voltage $\mathrm{V_m}$, which itself is a dynamical variable as given by equation (2.4).

In the Hodgkin-Huxley model the characteristic times and steady-states were found experimentally using so-called voltage-clamp procedure. In this procedure, two electrodes are attached to the intracellular and extracellular environment of the axon. Then a current is applied to the cell such that the membrane voltage remains at a constant testing value. The time evolution of the current is recorded. The measurements are repeated to determine steady-state values and characteristic times for a range of membrane voltages.

The solution of (2.6) has to satisfy the initial condition $n(0) = n_0$ which gives

$$n = n_\infty - (n_\infty - n_0) \exp\left(-\frac{t}{\tau_n}\right). \tag{2.7}$$

The relation with the experimental results is given by combining equation (2.7) with equation (2.5) as

$$g_\mathrm{K} = \left[(g_{\mathrm{K}\infty})^{\frac{1}{4}} - \left((g_{\mathrm{K}\infty})^{\frac{1}{4}} - (g_{\mathrm{K}0})^{\frac{1}{4}}\right) \exp\left(-\frac{t}{\tau_n}\right)\right]^4, \tag{2.8}$$

where $g_{\mathrm{K}0}$ corresponds to the initial value of conductance and $g_{\mathrm{K}\infty}$ and steady-state conductance. Then the characteristic time and steady-state can be defined as

$$\tau_n = \frac{1}{\alpha_n + \beta_n}, \tag{2.9a}$$

$$n_\infty = \frac{\alpha_n}{\alpha_n + \beta_n}, \tag{2.9b}$$

and using these expressions the equation (2.6) gets the form

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(1-n) - \beta_n n. \tag{2.10}$$

The α_n and β_n can now be interpreted as transition rates of the gate model, which determine the probability per a unit of time that the gate n opens (α_n) or closes (β_n). Finally, the transition rates are found as

$$\alpha_n = \frac{n_\infty}{\tau_n}, \tag{2.11a}$$

$$\beta_n = \frac{1 - n_\infty}{\tau_n}. \tag{2.11b}$$

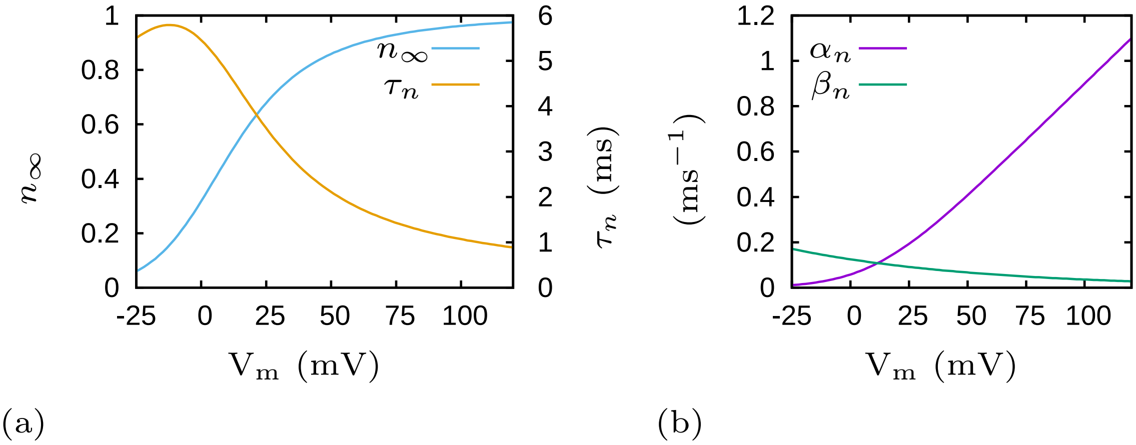


Figure 2.2: Characteristic time, steady-state and transition rates of I_K gate model. Panel (a) shows the steady-state n_∞ (blue line, left ticks) and characteristic time τ_n (yellow line, right ticks). Panel (b) shows the opening transition rate α_n (green line) and closing transition rate β_n (purple line).

Using the determined transition rates for the whole range of experimental voltages, Hodgkin and Huxley found that the curves which best fit the transition rates of the potassium channel are given by

$$\alpha_n = \frac{0.01(-\mathrm{V_m}+10)}{\exp\left(\frac{-\mathrm{V_m}+10}{10}\right)-1}, \tag{2.12a}$$

$$\beta_n = 0.125\exp\left(\frac{-\mathrm{V_m}}{80}\right). \tag{2.12b}$$

Figure 2.2 shows the characteristic times and steady-states in panel (a), and transition rates in panel (b) from the Hodgkin-Huxley model. Those traces were fitted to the data obtained experimentally.

The situation is a bit more complicated in the case of the sodium current (2.3b). The conductance for sodium is described by the equation

$$g_\mathrm{Na} = \bar{g_\mathrm{Na}} m^3 h, \tag{2.13}$$

where $\bar{g_\mathrm{Na}}$ is the maximum conductance. This equation can be interpreted using a similar notion of a gate model, as we described in the case of potassium current. The model for sodium has three independent activation gates m and one inactivation gate h. The channel is only open when all activation and inactivation gates are open. The time evolution of the gates are described by the first-order differential equations

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(1-m) - \beta_m m, \tag{2.14a}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(1-h) - \beta_h h, \tag{2.14b}$$

where α_m and β_m are transition rates of the activation gate m, and α_h and β_h are transition rates of the inactivation gate h.

The solution of these equations have to satisfy the initial conditions $m(0) = m_0$ and $h(0) = h_0$, which gives

$$m = m_\infty - (m_\infty - m_0)\exp\left(-\frac{t}{\tau_m}\right), \tag{2.15a}$$

$$h = h_\infty - (h_\infty - h_0)\exp\left(-\frac{t}{\tau_h}\right), \tag{2.15b}$$

where the initial state and characteristic times are given by

$$m_0 = \frac{\alpha_{m0}}{\alpha_{m0} + \beta_{m0}}, \tag{2.16a}$$

$$\tau_m = \frac{1}{\alpha_m + \beta_m}, \tag{2.16b}$$

$$h_0 = \frac{\alpha_{h0}}{\alpha_{h0} + \beta_{h0}}, \tag{2.16c}$$

$$\tau_h = \frac{1}{\alpha_h + \beta_h}. \tag{2.16d}$$

Unlike the potassium model there are two types of gates with different behaviour, which need to be found, to describe the behaviour of the channel as observed in experimental data. This is done using the assumption that the sodium conductance of the activation gate is very small for values below $\mathrm{V_m} = 30$ mV and therefore m_0 can be set to zero for small values of the membrane voltage. For values above $\mathrm{V_m} = -30$ mV the inactivation gate is fully inactivated and so h_∞ is ignored for high values of the membrane voltage. Then equation (2.15) gets the form

$$m = m_\infty\left[1 - \exp\left(-\frac{t}{\tau_m}\right)\right], \tag{2.17a}$$

$$h = h_0\exp\left(-\frac{t}{\tau_h}\right). \tag{2.17b}$$

This approximation is substituted into equation (2.13), which yields

$$g_{\mathrm{Na}} = \bar{g}_{\mathrm{Na}} m_\infty^3 h_0\left[1 - \exp\left(-\frac{t}{\tau_m}\right)\right]^3 \exp\left(-\frac{t}{\tau_h}\right). \tag{2.18}$$

Then the values of the characteristic time are estimated from the experimental data.

Hodgkin and Huxley found the values of the transition rates from the voltage-clamp experiments to be

$$\alpha_m = \frac{0.1(-\mathrm{V_m} + 25)}{\exp\left(\frac{-\mathrm{V_m}+25}{10}\right) - 1}, \tag{2.19}$$

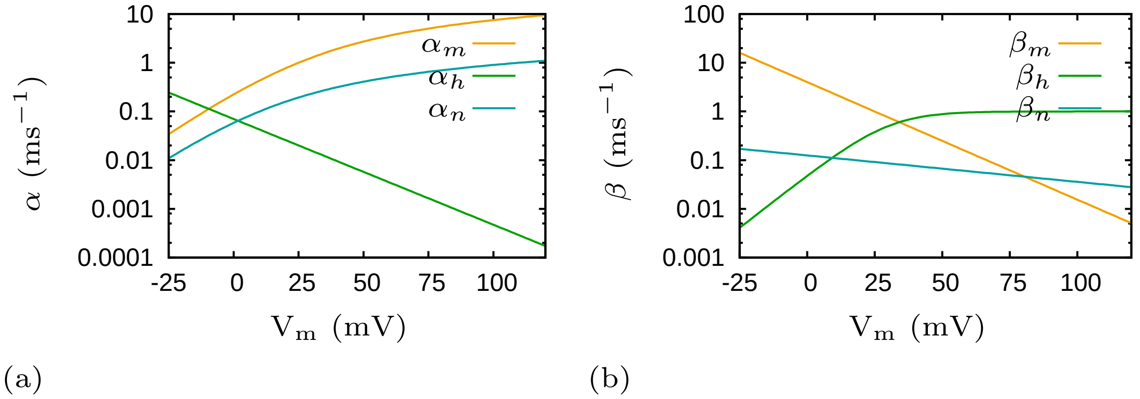


Figure 2.3: Transition rates of Hodgkin-Huxley gate models as function of membrane voltage $\mathrm{V_m}$: (a) opening transition rate α; (b) closing transition rate β. Yellow lines represent gate m, green represent gate h, cyan represent gate n. The rates are shown in logarithmic scale.

$$\beta_m = 4 \exp\left(\frac{-\mathrm{V_m}}{18}\right), \tag{2.20}$$

$$\alpha_h = 0.07 \exp\left(\frac{-\mathrm{V_m}}{20}\right), \tag{2.21}$$

$$\beta_h = \frac{1}{\exp\left(\frac{-\mathrm{V_m}+30}{10}\right)+1}. \tag{2.22}$$

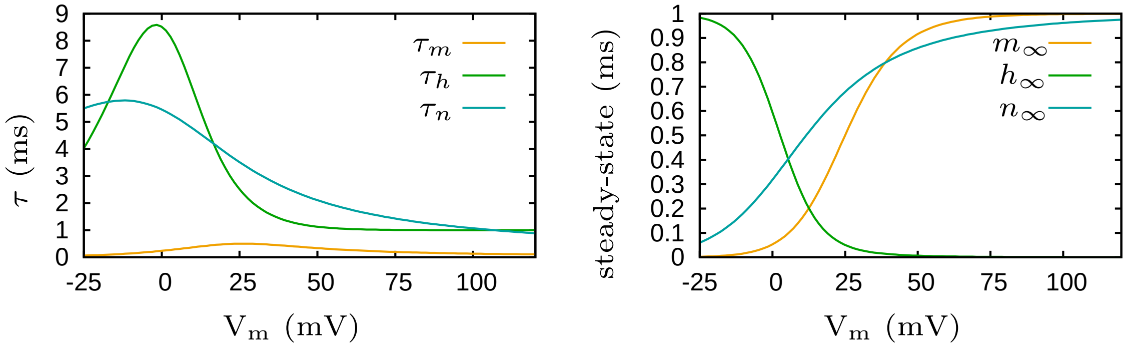
The transition rates of both I_{K} and I_{Na} model suggested by Hodkin and Huxley are shown in Figure 2.3.

2.1.3 Summary of the Hodgkin-Huxley Model

This section summarises the equations describing the giant squid axon in the Hodgkin-Huxley model. We implemented this model in the C programming language as listed in Appendix D. The complete definition of the model is sumarised in Tables 2.1–2.4.

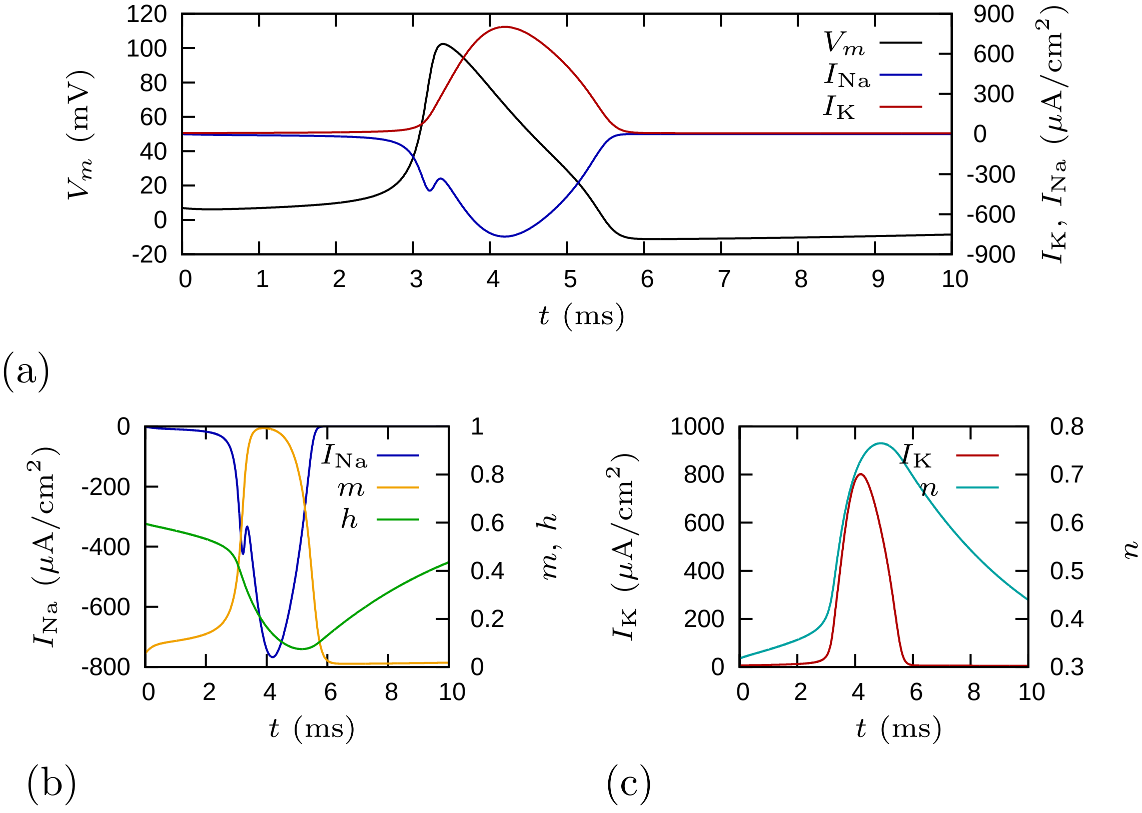
Figure 2.4 shows the characteristic times in panel (a), and steady-state values panel (b), for both potassium and sodium channel's gates n, m, h. The inverse of the characteristic time can be understood as the speed, in which the corresponding variable approaches the steady-state, i.e. lower value of the characteristic time, the faster it will approach to the steady-state.

The reproduction of the action potential and ionic currents using the Hodgkin-Huxley squid axon model is shown in Figure 2.5. Panel (a) shows the membrane voltage and sodium and potassium currents (the leakage current is not shown). Panel (b) shows the activation gate m and inactivation gate h during the same action potential and the corresponding current I_{Na}. Panel (c) shows the activation gate n and the corresponding current I_{K}.



(a) (b)

Figure 2.4: (a) Characteristic times and (b) steady-states of Hodgkin-Huxley gate models as a function of membrane voltage $\mathrm{V_m}$. Yellow lines represent gate m, green represent gate h, cyan represent gate n.



(a)

(b) (c)

Figure 2.5: Hodgkin and Huxley action potential and currents (a), I_Na current and gates m, h (b), I_K currents and n-gate (c).

The initial conditions of the gates were set to the steady-state values. The action potential was initiated by setting the initial conditions of the membrane potential $\mathrm{V_m}$ above a threshold value which initiates the excitation. At the resting state, which is at the voltages around the value of $\mathrm{V_m} = 0$ mV, the activation gates n and m gates are closed, while the inactivation gate h is open as can be seen from Figure 2.4. Applying an external stimulation, the membrane voltage is increased, i.e. the membrane depolarises. As a result, the activation gate m, which has a small characteristic time, i.e. is fast, opens and allows the influx of sodium ions. This causes further increase of membrane voltage up to values about $\mathrm{V_m} = 100$ mV. At those values, the inactivation gate h which restricts the sodium influx. While the membrane potential was increasing, the potassium gate n, started gradually opening. At the peak of the membrane voltage, the characteristic time of depolarisation τ_n is small, so the potassium channels are opening faster. This allows the flow of potassium outside of the cell, that causes a decrease of membrane voltage back to the resting values.

2.2 Cardiac Excitation Models

2.2.1 Development of Cardiac Models

The Hodgkin and Huxley model was introduced as the first electrophysiological model. The same ideas are to a certain degree repeated and improved in the modern biophysically detailed models of cardiac cells. Those models contain many differential equations, e.g. the Bondarenko (2014) model contains over one hundred dynamical variables [4]. In this subsection we describe the main differences between modern cardiac models and the Hodgkin-Huxley model.

Structurally, the Hodgkin-Huxley model has only one intracellular compartment and assumes that intra- and extracellular concentrations of ions are fixed. The modern models introduce further dynamical equations for the concentrations of $\mathrm{Na^+}$, $\mathrm{K^+}$, $\mathrm{Ca^{2+}}$ which are based on the laws for the conservation of mass within the compartments. Using the values of intra- and extracellular concentrations, the electrochemical equilibrium is determined from the Nernst equation. For instance, the Nernst equation for the electrochemical equilibrium potential of potassium is given by

$$E_{\mathrm{K}} = \frac{RT}{F} \ln \left(\frac{[\mathrm{K^+}]_o}{[\mathrm{K^+}]_i} \right) \tag{2.23}$$

where R is the gas constant, T is the temperature, F is Faraday's constant, and $[\mathrm{K^+}]_i$ and $[\mathrm{K^+}]_o$ are potassium intracellular and extracellular concentrations, respectively.

Modern models use a more detailed structure of the cell. The interior of the cell is further divided into compartments constituting different functional elements. From the electrophysiological point of view, the most interesting compartment is the sarcoplasmic reticulum which contributes to dynamics of calcium in the cell. Figure 2.6 shows a simplified diagram of calcium dynamics. The sarcoplasmic reticulum acts as a store of calcium ions.

The release of the calcium ions from the sarcoplasmic reticulum is controlled by so-called ryanodine receptor (RyR). The release of calcium into the intracellular environment causes biochemical processes that trigger the mechanical contraction of the cell. More detailed description of the functionality of RyR is provided in Subsection 2.4.3.

Table 2.1: Dynamical states of the Hodgkin and Huxley model.

var.	description	init. val.	units	definition
$\mathrm{V_m}$	membrane voltage	7.0	mV	$\mathrm{dV_m}/\mathrm{d}t = -(I_{\mathrm{Na}} + I_{\mathrm{K}} + I_l)/C$
n	K^+ activation gate	0.3177	prob.	$\mathrm{d}n/\mathrm{d}t = \alpha_n(1-n) - \beta_n n$
m	Na^+ act. gate	0.0530	prob.	$\mathrm{d}m/\mathrm{d}t = \alpha_m(1-m) - \beta_m m$
h	Na^+ inact. gate	0.5960	prob.	$\mathrm{d}h/\mathrm{d}t = \alpha_h(1-h) - \beta_h h$

Table 2.2: Definitions of currents.

variable	description	units	definition
I_{Na}	sodium current	μA/cm^2	$I_{\mathrm{Na}} = \bar{g}_{\mathrm{Na}} m^3 h(\mathrm{V_m} - E_{\mathrm{Na}})$
I_{K}	potassium current	μA/cm^2	$I_{\mathrm{K}} = \bar{g}_{\mathrm{K}} n^4(\mathrm{V_m} - E_{\mathrm{K}})$
I_l	leakage current	μA/cm^2	$I_l = g_l(\mathrm{V_m} - E_l)$

Table 2.3: Opening (α) and closing (β) transition rates.

n-gate	m-gate	h-gate
$\alpha_n = \dfrac{0.01(-\mathrm{V_m}+10)}{\exp\left(\frac{-\mathrm{V_m}+10}{10}\right)-1}$	$\alpha_m = \dfrac{0.1(-\mathrm{V_m}+25)}{\exp\left(\frac{-\mathrm{V_m}+25}{10}\right)-1}$	$\alpha_h = 0.07\exp\left(\dfrac{-\mathrm{V_m}}{20}\right)$
$\beta_n = 0.125\exp\left(\dfrac{-\mathrm{V_m}}{80}\right)$	$\beta_m = 4\exp\left(\dfrac{-\mathrm{V_m}}{18}\right)$	$\beta_h = \dfrac{1}{\exp\left(\frac{-\mathrm{V_m}+30}{10}\right)+1}$

Table 2.4: Constant parameters.

variable	description	value	units
C	membrane capacitance	1	μF/cm^2
$\bar{g}_{\mathrm{Na}}$	maximum conductance of I_{Na} current	120	mS/cm^2
E_{Na}	reversal potential of sodium ions	115	mV
$\bar{g}_{\mathrm{K}}$	maximum conductance of I_{K} current	36	mS/cm^2
E_{K}	reversal potential of potassium ions	-12	mV
g_l	conductance of current I_l (constant)	0.3	mS/cm^2
E_l	reversal potential of leak ions	10.613	mV

2.2.2 Calcium Buffers Kinetics

The cell can be understood in terms of a number of separated compartments, with specific ionic concentration in each. The calcium dynamics are defined in terms of events involving a single compartment (diffusion of calcium ions, reactions which produce or bind the free calcium) and the flow of calcium between two different compartments.

In certain conditions the membrane separating the compartments becomes permeable and allows the calcium to flow from one compartment of the cell to another, which can be described by an equation

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]}{\mathrm{d}t} = f([\mathrm{Ca}^{+2}], \ldots) \tag{2.24}$$

where $f([\mathrm{Ca}^{+2}], \ldots)$ is a function defined within a particular cell model. The definition of the function $f([\mathrm{Ca}^{+2}], \ldots)$ takes into account the fluxes from and into the particular compartment, and the volumes of the compartments.

The calcium molecules then diffuse within the compartment to achieve homogeneous calcium concentration. Additionally, calcium is a subject to a reaction with molecules called buffers, i.e. proteins which bind the calcium such that it becomes inaccessible to other reactions or flow between the compartments.

The Ca^{+2} binding to the Ca^{+2} specific buffer is described by the chemical reaction

$$[\mathrm{Ca}^{+2}]_f + B_f \rightleftharpoons [\mathrm{Ca}^{+2}]_b$$

where $[\mathrm{Ca}^{+2}]_f$ is the concentration of free Ca^{2+}; B_f is the concentration of free buffer and $[\mathrm{Ca}^{+2}]_b$ is the concentration of complexes of Ca^{2+} bound to the buffer B_f [5]. The total concentration of calcium $[\mathrm{Ca}^{+2}]_t$ and buffer molecules B_t is given as

$$[\mathrm{Ca}^{+2}]_t = [\mathrm{Ca}^{+2}]_f + [\mathrm{Ca}^{+2}]_b, \tag{2.25}$$

$$B_t = B_f + [\mathrm{Ca}^{+2}]_b. \tag{2.26}$$

From equation (2.25) we derive the relation for the buffered calcium concentration as $B_f = B_t - [\mathrm{Ca}^{+2}]_b$. Knowing the rate of binding k_{on} and unbinding k_{off} we can write down the differential equation [5]

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]_b}{\mathrm{d}t} = k_{\mathrm{on}}[\mathrm{Ca}^{+2}]_f \left(B_t - [\mathrm{Ca}^{+2}]_b\right) - k_{\mathrm{off}}[\mathrm{Ca}^{+2}]_b. \tag{2.27}$$

If the diffusion happens in a fast time scale, the whole compartment would have homogeneous Ca^{+2} concentration very rapidly. In that case Ca^{+2} binding to the buffer can be approximated to its steady-state by setting $\mathrm{d}[\mathrm{Ca}^{+2}]_b/\mathrm{d}t = 0$ giving

$$0 = k_{\mathrm{on}}[\mathrm{Ca}^{+2}]_f \left(B_t - [\mathrm{Ca}^{+2}]_b\right) - k_{\mathrm{off}}[\mathrm{Ca}^{+2}]_b, \tag{2.28}$$

which can be rewritten to give a formula for buffered calcium concentration

$$[\mathrm{Ca}^{+2}]_b = \frac{B_t[\mathrm{Ca}^{+2}]_f}{[\mathrm{Ca}^{+2}]_f + k}, \tag{2.29}$$

where $k = k_{\mathrm{off}}/k_{\mathrm{on}}$.

Substituting the result (2.29) into the relation for the total Ca^{+2} concentration (2.25) yields

$$[\mathrm{Ca}^{+2}]_t = [\mathrm{Ca}^{+2}]_f + \frac{B_t[\mathrm{Ca}^{+2}]_f}{[\mathrm{Ca}^{+2}]_f + k}. \tag{2.30}$$

Considering that the calcium concentration change in time as the calcium flow enters and exits the membrane we can derive the rate of change by deriving both sides of the previous equation as

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]_t}{\mathrm{d}t} = \left(1 + \frac{B_t k}{([\mathrm{Ca}^{+2}]_f + k)^2}\right) \frac{\mathrm{d}[\mathrm{Ca}^{+2}]_f}{\mathrm{d}t} \tag{2.31}$$

as the rate of change of total calcium concentration change with the amount of current entering the cell as $\mathrm{d}[\mathrm{Ca}^{+2}]_t/\mathrm{d}t = I_{tot}$, the previous equation implies

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]_f}{\mathrm{d}t} = \left(1 + \frac{B_t k}{([\mathrm{Ca}^{+2}]_f + k)^2}\right)^{-1} I_{tot}. \tag{2.32}$$

2.2.3 Ion Channels

As mentioned along with the description of the Hodgkin-Huxley model in Section 2.1, it was discovered that the conductance of the membrane is controlled by large molecules residing in the cellular membrane [6]. Such molecules are called ion channels and are fundamental elements of electric excitation of the cell.

There are different mechanisms giving rise to the channel permeability. In the simplest view the conformation of the subunits comprising the ion channel allows forming a small pore through the membrane. The pore then lets the ions pass from one side of the membrane to the other. There are various types of ion channels. Some are specific to a particular kind of ions and others allow passage of a few different ionic species (the most important for electric excitation are ions are sodium, potassium and calcium and chlorine).

The channels can be further divided according to the force that drives the transport of the ions into two groups known as passive and active channels.

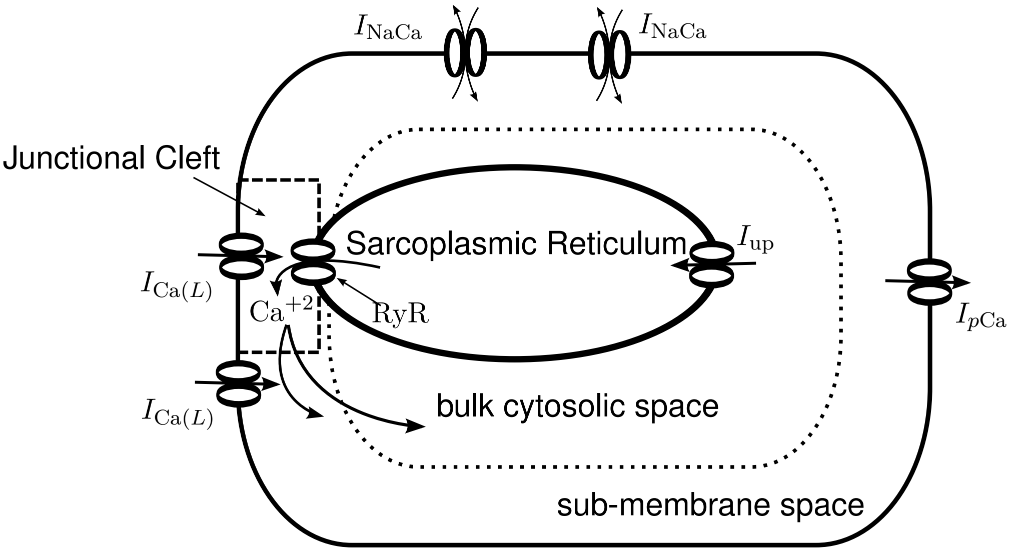


Figure 2.6: Diagram of a cardiac cell that illustrates the calcium dynamics. The calcium within the cell is stored inside the Sarcoplasmic reticulum, from where it is released through the ryanodine receptor (RyR) to the junctional cleft. From the junctional cleft it difuse into sub-membrane space and bulk cytosolic space. From where it is uptaken by I_{up}. Further detail on the kinetics of calcium is described Section .

The passive channels are the simplest. The ionic flow is driven by the concentration gradient on the two sides of the cellular membrane. Two representatives of the passive channels were described in Section 2.1. First, I_{Na} is responsible for inward sodium current which causes increase of membrane voltage from the resting potential and leads to the excitation of the cell. Second, I_{K} allows outward potassium current that restores the resting potential. If the cell had only passive channels, the excitation pulses would alter the physiological concentrations of ions and lead to a pathological situation. To keep the concentrations in the physiological range, the currents caused by passive channels have to be coupled with the reverse movement driven by active channels.

The active channels use energy to act against the concentration gradient. The primary active transport is ensured by so-called pumps. The pumps are driven by the chemical energy released during the dephosporylation of adenosine triphosphate (ATP) to adenosine diphosphate (ADP). The most prominent example of primary active transport is the sodium-potassium pump which moves one sodium ion out of the cell and one potassium ion inside the cell, i.e. both ionic species move against their concentration gradient.

The secondary active transport is mediated by so-called exchangers. In the exchangers one species moves along the electrochemical gradient, while the other moves against it. For instance the sodium-calcium exchanger swaps three sodium ions for one calcium ion. The sodium-calcium exchanger can act in both directions, i.e. in some conditions produce inward and in others outward current.

This functionality helps to restore restore normal concentration of calcium inside the cell.

The ionic flux constitutes an electric current. In Section 2.1 we described the traditional method called voltage-clamp method, which allows to measure membrane voltage and currents through all channels present in the membrane. To measure specific kind of channels all the other channels have to be deactivated by specific chemical agents. Alternatively, we can measure specific kind of channels on genetically engineered cells which form a single type of ion channels.

A similar method to voltage-clamp has been developed for a measurement of the current through a small number of ion channels or even a single channel – so-called patch-clamp technique. Unlike the voltage-clamp method, where a glass pipette containing the measurement electrode is connected with the internal space of the cell, the patch-clamp method seals the end of the glass pipette on a minute patch of membrane. Then the measurements record only the channels contained in the minute patch. As the channels under the patch undergo transitions from non-conductive to conductive state, discrete “flips” between zero and non-zero values of can be observed in the current recording. Those flips are driven by thermal noise, which causes the ion channel molecule to change the conformation in a way that allows the flux of ions [7].

The kinetics of the channel can be modelled using a state model, where the states corresponds to different conformations of the channel. The probability that the channel changes to another state is called transition rate. The probabilistic nature of the transition rate is due to the thermal noise. The usual assumption is that the transition rates do not depend on how the channel reached the state it is in. This is known as a Markov property, which means that the system is “memoryless”.

Simulations of a small population of ion channels by a stochastic state model (also known as a Markov chain) results in traces that are one realisation of such model. Due to the stochasticity, the realisations are not identical in general case. However, the results resamble the experimental traces from patch clamp experiments. As the number of simulated channels increases, the effect of noise averages out and the simulated traces look like a voltage clamp experimental traces on a whole cell. The Markov chain model can be used to derive a master equation corresponding to a situation when the number of channels goes to infinity. Such model is a deterministic system of ODEs.

One possible interpretation of the traditional gate models is that the closing and opening of the channel is controlled by a number of gates. A limiting factor of this approach is, that the movement of different gates (corresponding to different subunits of the molecule) is assumed to be independent. However, because the subunits are linked by atomic forces, this assumption cannot be generally valid. In contrast with the gate models the Markov chains do not necessarily

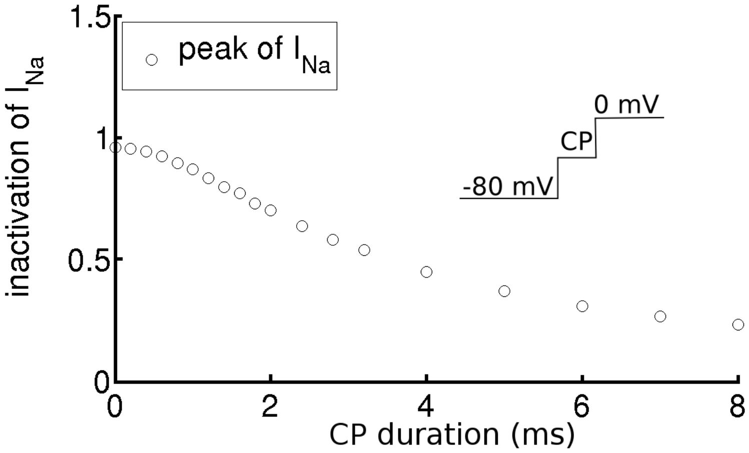


Figure 2.7: Experimental data of I_{Na} channel inactivation (shown by circles) as a function of duration of conditioning pulse (CP) to -35 mV. The CP duration is shown on the horizontal axis. The envelope through the circles corresponds to the inactivation of h-gate. Data for this plot were digitalised from Fig. 3. in [8].

assume the independence of different processes in the channel. Although, an equivalent representation of a gate model is possible using Markov chains, the Markov chains also enable a way of representing more generic situations, which cannot be expressed by gate models. The experimental evidence suggests that more generic models are required to better describe some measurement data (e.g. Figure 2.7). For that reason the Markov chain models have become a popular method of simulation of cardiac ion channel.

In the following section we look into an example, where the gate model fails to reproduce experimental data. Then we introduce the Markov chain model through a conversion between a simple gate model and Markov chain model, and finally we convert the Hodgkin-Huxley models for I_{Na} and I_{K} to their Markov chain equivalents.

2.3 Ion Channel Models

2.3.1 Exposing the Limitations of Gate Model

The sodium current I_{Na} is responsible for the initiation of the action potential. The model used by Hodgkin and Huxley contains three activation gates m and one inactivation gate h which are assumed to be independent (as described in subsection 2.1.2). However, Armstrong and Bezanilla (1977) [8, 9] have shown experimental data which cannot be satisfactorily reproduced by a gate model. They performed several voltage-clamp experiments which support this discovery. We discuss the second one of the three they presented in their paper [8] and highlight the main results in Figure 2.7. First, we describe the experimental protocol, then the theoretical results, and finally explain the discrepancy between the experiments and the theory.

The voltage-clamp experimental protocol is depicted in the inset of the figure and is as follows. First, the “holding” membrane voltage was set to $V_{\mathrm{h}} = -80$ mV. Then, a conditioning pulse (CP) was applied for a variable duration up to 8 ms. The duration of the conditioning pulse is represented on the horizontal axis in the figure. The value of the voltage at the CP was $V_{\mathrm{CP}} = -35$ mV. After the CP a “testing” pulse to $V_{\mathrm{test}} = 0$ mV was applied.

According to the Hodgkin-Huxley model, the channel has two types of gates. The activation m-gate that is closed at low voltages but opens quickly when the voltage increases due to small characteristic time (as can be seen in Figure 2.4, note that the values of voltage in the testing protocol is offset as compared to Hodgkin and Huxley by about -80 mV). The h-gate responsible for inactivation is opened at low voltages, and closes with increasing membrane voltage. Because of relatively large characteristic time, the action of the h-gate is slow.

At the beginning of the experiment, the h-gates are opened $h_{\infty}(V_{\mathrm{h}}) \sim 1$ and m-gates are closed $m_{\infty}(V_{\mathrm{h}}) \sim 0$. Because of fast characteristic time of of m-gate, the m-gates quickly attain to values $m_{\infty}(V_{\mathrm{CP}})$, when the conditioning pulse (CP) is applied. The time evolution of h-gate is slow so the ratio of open h-gates depends on the duration of CP t_{CP} as $h(t_{\mathrm{CP}})$.

So the relation for the current after the onset of testing potential V_{test} is

$$I_{\mathrm{Na}} = \bar{g_{\mathrm{Na}}} m_{\infty}(V_{\mathrm{CP}})^3 h(t_{\mathrm{CP}})(V_{\mathrm{test}} - E_{\mathrm{Na}}). \tag{2.33}$$

The inactivation is determined by comparing the current after the onset of testing potential in two different experiments. This is because, the $h(t_{\mathrm{CP}})$ is the only variable, that is different in these two experiments.

The current is depicted in the figure by circles. The envelope through the circles in the figure represents the time evolution of the inactivation gate $h(t_{\mathrm{CP}})$.

The inactivation in the gate model follows the exponential function as described by equations (2.15b). However, the evolution of experimental results show clearly sigmoid curve of the inactivation. Hence, the gate model is not able to represent the data realistically.

A modification of gate models in order to reproduce such experimental data is not possible. Hence, so-called Markov chain models were suggested in order to address the limitations of gate models.

2.3.2 Deriving Markov Chain Models

The ionic current is derived from an Ohm’s law, where the conductance of a particular group of ion channels depends on a ratio between opened and closed

$$(1-s) \underset{\beta_s}{\overset{\alpha_s}{\rightleftharpoons}} s$$

(a)

$$C \underset{\beta_s}{\overset{\alpha_s}{\rightleftharpoons}} O$$

(b)

Figure 2.8: Diagram of a simple gate model and its Markov chain equivalent. The gate model (a) is determined by its open probablility s, its closing probability $1-s$, and the transition probabilities per unit of time (α_s for a closed channel to open, β_s for an open channel to close). The Markov Chain model (b) can reside in an open and closed state, where O and C are the probabilities of the channel to be in the corresponding state.

channels of that group as

$$g_s = \bar{g}_s P_{\text{open}}(t) \tag{2.34}$$

where the $\bar{g}_s$ is the maximal conductance, which is achieved when all channels are open. The ratio between opened and closed channels, also called an open probability of a channel, is

$$P_{\text{open}}(t) = \prod_{i=1}^{N} s_i(t) \tag{2.35}$$

where $s_i(t)$ represents the open probability of each of the N gates in a channel calculated from equations of the form

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \alpha_s(1-s) - \beta_s s \tag{2.36}$$

where α_s is the transition probability of an open gate to close, and β_s is the transition probability of a closed gate to open per unit of time.

As the first example we consider a gate model of a channel with one gate so that $N = 1$, so the open probability of the channel is $P_{\text{open}} = s$. The model assumes that a channel is either open or closed, so $P_{\text{closed}} = 1 - s$. Then the transition between the states can be expressed as in Figure 2.8(a). Considering the situation with a closed gate as one conformational state of the channel with probability of being “occupied” $C = 1 - s$ and the situation with open gate as another conformational states with probabilty $O = s$, we can convert the model into the form as in Figure 2.8(b). The model can be described by a system of differential equations

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_s C - \beta_s O, \tag{2.37a}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = \beta_s O - \alpha_s C, \tag{2.37b}$$

where the first is obtained by replacing the open and closed state in equation (2.36), and the second by replacing an equation describing a change in complementary probabilities (not shown). Defining a vector of dynamical variables and a coefficient (transition) matrix

$$\vec{u} = \begin{bmatrix} O \\ C \end{bmatrix}, \qquad \boldsymbol{A} = \begin{bmatrix} -\beta_s & \alpha_s \\ \beta_s & -\alpha_s \end{bmatrix} \tag{2.38}$$

we can obtain a general system of linear differential equations describing a Markov chain model as

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}(t)\vec{u} \tag{2.39}$$

where $\vec{u}$ represents the states of the Markov chain and $\boldsymbol{A}$ is the transition matrix.

The states of the Markov chain correspond to the probability that the channel ocupies a particular conformational state. In other words it is the ratio between the number of channels in the particular conformational state and the total number of channels. In biological cells, the total number of channels is a large, but a finite number. However, for practical purposes we assume the number of the channel goes to the limit of infinity, such that the probability is a real number between 0 and 1.

The Markov chain model satisfies the state conservation law, according to which the sum of all states is equal to one. This also implies that the sum of the transition rates in the columns of the transition matrix is zero. Using the state conservation law, we can always reduce the dimensionality of the Markov chain model by one.

The ion channel manifests its behaviour only through the change of conductance, which causes measurable ionic currents. For practical purposes, any model whose behaviour reproduces the experimental evidence is equivalent even when it is mathematially different. In the Markov chain model the conductance is proportional to the sum of probabilities of open states (if there is more than one open state). The closed states affect the conductance indirectly through the transitions to open states.

2.3.3 Conversion from Markov chain to a Gate Model

The conversion between the Markov chain to gate model can be made only in special cases, when the Markov chain has a particular symmetry. An example of such a Markov chain is the model shown in Figure 2.9 with four states and identical transition rates in the horizontal and vertical directions on the diagram.

Here we demonstrate the conversion from the model in Figure 2.9 to a gate model. The probabilities that the model resides in closed C, inactivated I, closed

$$\begin{tikzcd}
\mathrm{IC} \arrow[r, shift left, "\alpha_d"] \arrow[d, shift left, "\beta_f"] & I \arrow[l, shift left, "\beta_d"] \arrow[d, shift left, "\beta_f"] \\
C \arrow[r, shift left, "\alpha_d"] \arrow[u, shift left, "\alpha_f"] & O \arrow[l, shift left, "\beta_d"] \arrow[u, shift left, "\alpha_f"]
\end{tikzcd}$$

Figure 2.9: Simple Markov chain model diagram. C denotes closed state, O denotes open state, IC denotes inactivated closed state and I denotes inactivated state. Greek letters denote transition rates.

inactivated IC and open O states are computed by a system of equations

$$\frac{\mathrm{d}C}{\mathrm{d}t} = \beta_f \mathrm{IC} + \beta_d O - (\alpha_d + \alpha_f)C, \tag{2.40a}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \beta_f I + \alpha_d C - (\alpha_f + \beta_d)O, \tag{2.40b}$$

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \alpha_d \mathrm{IC} + \alpha_f O - (\beta_d + \beta_f)I, \tag{2.40c}$$

$$\frac{\mathrm{d}\,\mathrm{IC}}{\mathrm{d}t} = \beta_d I + \alpha_f C - (\alpha_d + \beta_f)\mathrm{IC} \tag{2.40d}$$

where α_d corresponds to the transition probability from left to right (i.e., from C to O or from IC to I), and β_d corresponds to the transition probability from right to left, while α_f corresponds to the transition probability from bottom to top (i.e., from C to IC or from O to I), and β_f corresponds to the transition probability from top to bottom.

Using the definition of the Markov chain model (2.39), we can write the transition matrix and the state vector as

$$\boldsymbol{A} = \begin{bmatrix} -(\alpha_d + \alpha_f) & \beta_d & 0 & \beta_f \\ \alpha_d & -(\alpha_f + \beta_d) & \beta_f & 0 \\ 0 & \alpha_f & -(\beta_d + \beta_f) & \alpha_d \\ \alpha_f & 0 & \beta_d & -(\alpha_d + \beta_f) \end{bmatrix}, \quad \vec{u} = \begin{bmatrix} C \\ O \\ I \\ \mathrm{IC} \end{bmatrix}. \tag{2.41}$$

The equivalent gate model is obtained by assuming the existence of two gates: gate d with open probability d and close probability $(1-d)$, and gate f with open probability f and closed probability $(1-f)$. Combining the open and closed gates (the order does not matter) leads to four states: $C = f(1-d)$, $O = df$, $I = d(1-f)$ and $\mathrm{IC} = (1-d)(1-d)$.

Introducing these new variables we can add (2.40a) to (2.40b) to get

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \alpha_f(1-f) - \beta_f f \tag{2.42}$$

and adding (2.40b) to (2.40c) we obtain

$$\frac{\mathrm{d}d}{\mathrm{d}t} = \alpha_d(1-d) - \beta_d d \tag{2.43}$$

which has a familiar form of a gate model. In particular it describes an ion channel with one activation gate d and one inactivation gate f which corresponds to a slow inward current I_s [10].

Markov chains can describe behaviour which would not be possible to simulate with a gate model. For instance the transition rates between the states IC and C in Figure 2.9 could be replaced by different values and such a model could not be reduced to a gate model. This comes at the expense of higher computational cost due to larger number of dynamical variables, but often offers an advantage of more realistic reproduction of experimental data.

Finally, we mention that the transition rates of a Markov chain models of ion channels are usually expressed in an exponential form as

$$\alpha = \alpha_0 \exp\left(z_\alpha \frac{\mathrm{V_m} F}{RT}\right), \tag{2.44}$$

$$\beta = \beta_0 \exp\left(-z_\beta \frac{\mathrm{V_m} F}{RT}\right) \tag{2.45}$$

where α_0 and β_0 (ms) are the transition rates at membrane voltage $\mathrm{V_m} = 0$ mV and z_α and z_β are the equivalent charge movements during the state transition. The parameters α_0, β_0, z_α and z_β are typically estimated to fit the experimental data.

2.3.4 Conversion of Hodgkin-Huxley I_{Na} and I_{K} to Markov Chains

In this subsection we describe the conversion of the I_{K} and I_{Na} gate models from Hodgkin and Huxley model to an equivalent Markov chain model. A combination of gate states for each channel correspond to one state of the resulting Markov chain model.

For the I_{K} model, the open probability is given by n^4, i.e. a product of open probabilities of four identical gates n. Each gate can be in two states: either open or closed state. The probability of an open state is n and its complement, the probability of a closed state is $(1-n)$. To avoid confusion in the terminology, we call the states n and $(1-n)$ *gate states*, while the states of the Markov chain are called *channel states*.

We consider two different approaches to convert the gate model into a Markov chain model based on the principles from combinatorics. The first approach is applied if the order, in which the individual gates close, matters. Then each channel state is one of the possible *variations* (in the combinatorial sense) of the

$$C_4 \underset{\beta_n}{\overset{4\alpha_n}{\rightleftharpoons}} C_3 \underset{2\beta_n}{\overset{3\alpha_n}{\rightleftharpoons}} C_2 \underset{3\beta_n}{\overset{2\alpha_n}{\rightleftharpoons}} C_1 \underset{4\beta_n}{\overset{\alpha_n}{\rightleftharpoons}} O_{\mathrm{K}}$$

Figure 2.10: Diagram of the Hodgkin-Huxley I_{K} model as Markov chain. $O_{\mathrm{K}} = C_0$ and C_i is a channel state corresponding to i gates in closed, i.e. in $(1 - n)$ gate state.

two gate states, e.g. channel state $n, n, n, (1 - n)$ is different from $(1 - n), n, n, n$. So there are $2^4 = 16$ channel states. Each channel state constitutes a vertex of a four-dimensional hypercube, and is connected to four other channel states. The transition rate at each connection corresponds to exactly one transition of the gates.

For our purposes, the only state which is ultimately important is the state when all the gates are open. So we do not need to consider states in which any of the gates closes, e.g. channel state $n, n, n, (1 - n)$ is identical to $(1 - n), n, n, n$. Each of the channel states is one of the possible combinations (in the combinatorial sense) of gate states. Thus, the number of channel states is 5 and the Markov chain can be visualised in a simple linear structure as in Figure 2.10.

Unlike in the “variation” case (where order matters), in this case, each transition could be done by any of the gates in the same gate state. So, when all four gates have been closed and the channel resides in C_4, the transition α_n is multiplied by 4 as it is four times more likely per unit of time, that any of the four gates opens, than in a case of a single gate.

In general, the transition rates α_n between two channel states are multiplied by a number i, which corresponds to the number of states on the left in the diagram (i corresponds to the number of closed gates). The β_n between two states are multiplied by $4 - i$ where i corresponds to the number the channel state on the right in the diagram (then $4 - i$ is the number of open gates).

The initial conditions of the channel states are determined from the initial conditions n_0 of the gates. The initial condition of the gate is multiplied by the coefficients found from binomial expansion. The cases when all the channels are open or closed correspond to the power of four of the corresponding initial condition ($C_4(t_0) = (1 - n_0)^4$, and $O_{\mathrm{K}}(t_0) = n_0^4$ respectively). The states when one gate is open or closed have to be multiplied by 4 because it can be any of the four combinations that have one gate open or closed ($C_3(t_0) = 4(1 - n_0)^3 n_0$, and $C_1(t_0) = 4n_0^3(1 - n_0)$ respectively). The state with two open and two closed gates is found as $C_2 = 6n_0^2(1 - n_0)^2$.

The open probability of the Hodgkin-Huxley I_{Na} model is given by a product of the open probability of a combination of three activation gates m and the open probability of one inactivation gate h as $O_{\mathrm{Na}} = m^3 h$. All the activation gates are independent. So the activation part of the channel m^3 can be described by a

$$\begin{tikzcd}
C'_{3\mathrm{Na}} \arrow[r, harpoon, shift left, "3\alpha_m"] & C'_{2\mathrm{Na}} \arrow[l, harpoon, shift left, "\beta_m"] \arrow[r, harpoon, shift left, "2\alpha_m"] & C'_{1\mathrm{Na}} \arrow[l, harpoon, shift left, "2\beta_m"] \arrow[r, harpoon, shift left, "\alpha_m"] & C'_{0\mathrm{Na}} \arrow[l, harpoon, shift left, "3\beta_m"]
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
\mathrm{IC}_{3\mathrm{Na}} \arrow[r, harpoon, shift left, "3\alpha_m"] \arrow[d, harpoon, shift left, "\alpha_h"] & \mathrm{IC}_{2\mathrm{Na}} \arrow[l, harpoon, shift left, "\beta_m"] \arrow[r, harpoon, shift left, "2\alpha_m"] \arrow[d, harpoon, shift left, "\alpha_h"] & \mathrm{IC}_{1\mathrm{Na}} \arrow[l, harpoon, shift left, "2\beta_m"] \arrow[r, harpoon, shift left, "\alpha_m"] \arrow[d, harpoon, shift left, "\alpha_h"] & \mathrm{IC}_{0\mathrm{Na}} \arrow[l, harpoon, shift left, "3\beta_m"] \arrow[d, harpoon, shift left, "\alpha_h"] \\
C_{3\mathrm{Na}} \arrow[r, harpoon, shift left, "3\alpha_m"] \arrow[u, harpoon, shift left, "\beta_h"] & C_{2\mathrm{Na}} \arrow[l, harpoon, shift left, "\beta_m"] \arrow[r, harpoon, shift left, "2\alpha_m"] \arrow[u, harpoon, shift left, "\beta_h"] & C_{1\mathrm{Na}} \arrow[l, harpoon, shift left, "2\beta_m"] \arrow[r, harpoon, shift left, "\alpha_m"] \arrow[u, harpoon, shift left, "\beta_h"] & O_{\mathrm{Na}} \arrow[l, harpoon, shift left, "3\beta_m"] \arrow[u, harpoon, shift left, "\beta_h"]
\end{tikzcd}$$

(b)

Figure 2.11: Diagram of the Hodgkin-Huxley model for I_{Na} as a Markov chain. (a) considering only activation gates, where $C'_{i\mathrm{Na}}$ corresponds to state with i gates closed. Transition rates α_m, β_m are related to the opening and closing of the m gates. (b) shows equivalent Markov chain to I_{Na} model. $C_{i\mathrm{Na}}$ states correspond to the states of with open inactivated gate h and states $\mathrm{IC}_{i\mathrm{Na}}$ correspond to the same with closed inactivation gate h.

$$\begin{tikzcd}
\mathrm{IC}_3 \arrow[r, harpoon, shift left, "\alpha_{11}"] \arrow[d, harpoon, shift left, "\alpha_3"] & \mathrm{IC}_2 \arrow[l, harpoon, shift left, "\beta_{11}"] \arrow[r, harpoon, shift left, "\alpha_{12}"] \arrow[d, harpoon, shift left, "\alpha_3"] & \mathrm{IF} \arrow[l, harpoon, shift left, "\beta_{12}"] \arrow[r, harpoon, shift left, "\alpha_4"] \arrow[d, harpoon, shift left, "\alpha_3"] \arrow[dr, harpoon, shift left, "\alpha_2"] & \mathrm{IM}_1 \arrow[l, harpoon, shift left, "\beta_4"] \arrow[r, harpoon, shift left, "\alpha_5"] & \mathrm{IM}_2 \arrow[l, harpoon, shift left, "\beta_5"] \\
C_3 \arrow[r, harpoon, shift left, "\alpha_{11}"] \arrow[u, harpoon, shift left, "\beta_3"] & C_2 \arrow[l, harpoon, shift left, "\beta_{11}"] \arrow[r, harpoon, shift left, "\alpha_{12}"] \arrow[u, harpoon, shift left, "\beta_3"] & C_1 \arrow[l, harpoon, shift left, "\beta_{12}"] \arrow[r, harpoon, shift left, "\alpha_{13}"] \arrow[u, harpoon, shift left, "\beta_3"] & O \arrow[l, harpoon, shift left, "\beta_{13}"] \arrow[ul, harpoon, shift left, "\beta_2"] &
\end{tikzcd}$$

Figure 2.12: Diagram of the I_{Na} Markov chain model by Clancy and Rudy (2002) [2]

Markov chain in Figure 2.11. In the case of I_{Na}, the activation part has only four states, which are derived analogously to the I_{K} model.

The diagram of activation has to combine with the inactivation described by gate h. The inactivation part of the channel corresponds to only one gate and therefore has only two states – open h and the complementary closed case $(1-h)$. The combination of activation gates m^3 and the inactivation gate h gives us the diagram shown in Figure 2.11(b).

2.4 Popular Markov Chain Ion Channel Models

2.4.1 Fast Sodium Current I_{Na}

Figure 2.12 shows the Clancy and Rudy (2002) I_{Na} model. This model improves a previously published model by the same authors (1999), and was widely cited in the literature. This model includes 3 closed states: C_3, C_2, C_1; 5 inactivated states:

closed inactivated - IC_3, IC_2, fast inactivated - IF and slow inactivated - IM_1 and IM_2; and one open state: O. The open state O contributes to the calculation of the fast sodium current according to

$$I_{\mathrm{Na}} = \bar{g_{\mathrm{Na}}} O(\mathrm{V_m} - E_{\mathrm{Na}}), \tag{2.46}$$

where $\bar{g_{\mathrm{Na}}}$ is the maximal conductance when all the channels are open, and E_{Na} is reverse potential for sodium calculated from Nernst equation (similarly to (2.23)).

The system is described by the following ODEs:

$$\begin{aligned}
\frac{\mathrm{d}O}{\mathrm{d}t} =& \alpha_{13}C_1 + \beta_2\mathrm{IF} - (\beta_{13} + \alpha_2)O, && \text{(2.47a)}\\
\frac{\mathrm{d}C_1}{\mathrm{d}t} =& \alpha_{12}C_2 + \beta_{13}O + \alpha_3\mathrm{IF} - (\beta_{12} + \beta_3 + \alpha_{13})C_1, && \text{(2.47b)}\\
\frac{\mathrm{d}C_2}{\mathrm{d}t} =& \alpha_{11}C_3 + \alpha_3\mathrm{IC}_2 + \beta_{12}C_1 - (\beta_{11} + \alpha_{12} + \beta_3)C_2, && \text{(2.47c)}\\
\frac{\mathrm{d}C_3}{\mathrm{d}t} =& \beta_{11}C_2 + \alpha_3\mathrm{IC}_3 - (\alpha_{11} + \beta_3)C_3, && \text{(2.47d)}\\
\frac{\mathrm{d}\,\mathrm{IC}_3}{\mathrm{d}t} =& \beta_3C_3 + \beta_{11}\mathrm{IC}_2 - (\alpha_{11} + \alpha_3)\mathrm{IC}_3, && \text{(2.47e)}\\
\frac{\mathrm{d}\,\mathrm{IC}_2}{\mathrm{d}t} =& \alpha_{11}\mathrm{IC}_3 + \beta_3C_2 + \beta_{12}\mathrm{IF} - (\beta_{11} + \alpha_3 + \alpha_{12})\mathrm{IC}_2, && \text{(2.47f)}\\
\frac{\mathrm{d}\,\mathrm{IF}}{\mathrm{d}t} =& \alpha_{12}\mathrm{IC}_2 + \beta_4\mathrm{IM}_1 + \beta_3C_1 + \alpha_2O - (\beta_{12} + \alpha_4 + \beta_2 + \alpha_3)\mathrm{IF}, && \text{(2.47g)}\\
\frac{\mathrm{d}\,\mathrm{IM}_1}{\mathrm{d}t} =& \alpha_4\mathrm{IF} + \beta_5\mathrm{IM}_2 - (\beta_4 + \alpha_5)\mathrm{IM}_1, && \text{(2.47h)}\\
\frac{\mathrm{d}\,\mathrm{IM}_2}{\mathrm{d}t} =& \alpha_5\mathrm{IM}_1 - \beta_5\mathrm{IM}_2 && \text{(2.47i)}
\end{aligned}$$

where the transition rates are

$$\begin{aligned}
&C_3 \to C_2 && \alpha_{11} = \frac{3.802}{0.1027e^{-\mathrm{V_m}/17.0} + 0.20\exp\left(-\mathrm{V_m}/150\right)}, && \text{(2.48a)}\\
&C_2 \to C_1 && \alpha_{12} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/15.0\right) + 0.23\exp\left(-\mathrm{V_m}/150\right)}, && \text{(2.48b)}\\
&C_1 \to O && \alpha_{13} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/12.0\right) + 0.25\exp\left(-\mathrm{V_m}/150\right)}, && \text{(2.48c)}
\end{aligned}$$

$$\begin{aligned}
&\mathrm{IC}_3 \to \mathrm{IC}_2 && \alpha_{11} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/17.0\right) + 0.20\exp\left(-\mathrm{V_m}/150\right)}, && \text{(2.48d)}\\
&\mathrm{IC}_2 \to \mathrm{IF} && \alpha_{12} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/15.0\right) + 0.23\exp\left(-\mathrm{V_m}/150\right)}, && \text{(2.48e)}
\end{aligned}$$

$$\begin{aligned}
&C_2 \to C_3 && \beta_{11} = 0.1917\exp\left(-\mathrm{V_m}/20.3\right), && \text{(2.48f)}\\
&C_1 \to C_2 && \beta_{12} = 0.20\exp\left(-(\mathrm{V_m} - 5)/20.3\right), && \text{(2.48g)}\\
&O \to C_1 && \beta_{13} = 0.22\exp\left(-(\mathrm{V_m} - 10)/20.3\right), && \text{(2.48h)}
\end{aligned}$$

$$\mathrm{IF} \rightarrow C_1 \qquad \alpha_3 = 3.7933 \cdot 10^{-7} \exp\left(-\mathrm{V_m}/7.7\right), \tag{2.48i}$$

$$\mathrm{IC}_2 \rightarrow C_2 \qquad \alpha_3 = 3.7933 \cdot 10^{-7} \exp\left(-\mathrm{V_m}/7.7\right), \tag{2.48j}$$

$$\mathrm{IC}_3 \rightarrow C_3 \qquad \alpha_3 = 3.7933 \cdot 10^{-7} \exp\left(-\mathrm{V_m}/7.7\right), \tag{2.48k}$$

$$C_1 \rightarrow \mathrm{IF} \qquad \beta_3 = 8.4 \cdot 10^{-3} + 2 \cdot 10^{-5}\mathrm{V_m}, \tag{2.48l}$$

$$C_2 \rightarrow \mathrm{IC}_2 \qquad \beta_3 = 8.4 \cdot 10^{-3} + 2 \cdot 10^{-5}\mathrm{V_m}, \tag{2.48m}$$

$$C_3 \rightarrow \mathrm{IC}_3 \qquad \beta_3 = 8.4 \cdot 10^{-3} + 2 \cdot 10^{-5}\mathrm{V_m}, \tag{2.48n}$$

$$O \rightarrow \mathrm{IF} \qquad \alpha_2 = 9.178 \exp\left(\mathrm{V_m}/29.68\right), \tag{2.48o}$$

$$\mathrm{IF} \rightarrow O \qquad \beta_2 = \frac{\alpha_{13}\alpha_2\alpha_3}{\beta_{13}\beta_3}, \tag{2.48p}$$

$$\mathrm{IF} \rightarrow \mathrm{IM}_1 \qquad \alpha_4 = \alpha_2/100, \tag{2.48q}$$

$$\mathrm{IM}_1 \rightarrow \mathrm{IF} \qquad \beta_4 = \alpha_3, \tag{2.48r}$$

$$\mathrm{IM}_1 \rightarrow \mathrm{IM}_2 \qquad \alpha_5 = \alpha_2/(9.5 \cdot 10^4), \tag{2.48s}$$

$$\mathrm{IM}_2 \rightarrow \mathrm{IM}_1 \qquad \beta_5 = \alpha_3/50. \tag{2.48t}$$

2.4.2 L-type Calcium Current $I_{\mathrm{Ca}(L)}$

The calcium current $I_{\mathrm{Ca}(L)}$ plays a crucial role in forming the action potential morphology and duration in cardiac cells. Its importance is increased by providing negative feedback to control the intracellular Ca^{2+} concentration [11]. The channel is opened by the movement of four voltage-sensitive subunits and can inactivate due to voltage-dependent inactivation.

In this channel another type of inactivation due to ionic binding to the channel is observed in the presence of calcium ions Ca^{2+}. In the case of calcium binding, no current can flow through the channel, however, the movement of the voltage sensor remains unaffected. This calcium inactivation was discovered in experiments with barium ions Ba^{2+}, which behave similarly to calcium, however, as they do not bind to the channel, this type of inactivation is not observed.

A Markovian model that includes calcium-dependent inactivation was proposed by Imredy and Yue [12]. This model was improved to reflect the molecular structure proposed by Jafri *et al.* (2008) [13]. They define a normal mode, and a Ca-mode when the channel is inactivated by Ca^{+2}. Each mode contains five closed states with possible Ca^{+2} dependent transitions between the Ca-mode and the $\mathrm{V_m}$-mode; and one open state. The transition rates between the Ca-mode to the $\mathrm{V_m}$-mode were assumed to be very slow.

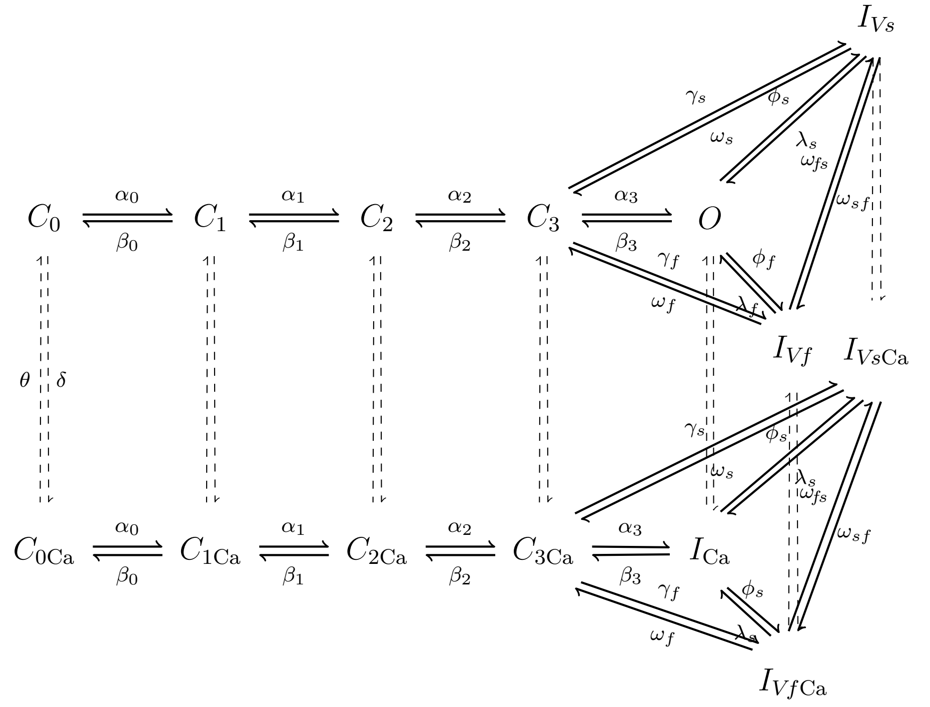


Figure 2.13: Markov chain model of $I_{\mathrm{Ca}(L)}$ [1]

In the model proposed by Faber *et al.* [1] the $I_{\mathrm{Ca}(L)}$ can operate in two modes as shown in Figure 2.13. The states shown in the bottom are states of the channel with Ca dependent inactivation, the states in the top corresponds to states without Ca^{2+} inactivation, i.e. in $\mathrm{V_m}$-mode. The only conductive state is the state O. The channel $I_{\mathrm{Ca}(L)}$ allows flow of calcium, sodium and potassium ions. The currents are proportional to the open probability according to

$$I_{\mathrm{Ca}(L)} = \bar{I}_{\mathrm{Ca}}(\ldots)O, \tag{2.49a}$$

$$I_{\mathrm{Ca,Na}} = \bar{I}_{\mathrm{Ca,Na}}(\ldots)O, \tag{2.49b}$$

$$I_{\mathrm{Ca,K}} = \bar{I}_{\mathrm{Ca,K}}(\ldots)O, \tag{2.49c}$$

where $\bar{I}_{\mathrm{Ca}}(\ldots)$, $\bar{I}_{\mathrm{Ca,Na}}(\ldots)$ and $\bar{I}_{\mathrm{Ca,K}}(\ldots)$ are function of other dynamical variables such as ionic concentrations in the subspace and membrane voltage, and O is the open state of $I_{\mathrm{Ca}(L)}$ Markov chain model.

The states of the model are described by the following system of ODEs [1]

$$\frac{\mathrm{d}C_0}{\mathrm{d}t} = \theta C_{0\mathrm{Ca}} + \beta_0 C_1 - (\delta + \alpha_0)C_0, \tag{2.50a}$$

$$\frac{\mathrm{d}C_1}{\mathrm{d}t} = \theta C_{1\mathrm{Ca}} + \beta_1 C_2 + \alpha_0 C_0 - (\delta + \beta_0 + \alpha_1)C_1, \tag{2.50b}$$

$$\frac{\mathrm{d}C_2}{\mathrm{d}t} = \theta C_{2\mathrm{Ca}} + \beta_2 C_3 + \alpha_1 C_1 - (\delta + \beta_1 + \alpha_2)C_2, \tag{2.50c}$$

$$\frac{\mathrm{d}C_3}{\mathrm{d}t} = \theta C_{3\mathrm{Ca}} + \beta_3 O + \alpha_2 C_2 + \omega_f I_{Vf} + \omega_s I_{Vs} - (\delta + \beta_2 + \alpha_3 + \gamma_f + \gamma_s) C_3, \tag{2.50d}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \theta I_{\mathrm{Ca}} + \alpha_3 C_3 + \lambda_f I_{Vf} + \lambda_s I_{Vs} - (\delta + \beta_3 + \phi_f + \phi_s) O, \tag{2.50e}$$

$$\frac{\mathrm{d}I_{Vf}}{\mathrm{d}t} = \gamma_f C_3 + \phi_f O + \omega_{sf} I_{Vs} + \theta I_{Vf\mathrm{Ca}} - (\omega_f + \omega_{fs} + \lambda_f + \delta) I_{Vf}, \tag{2.50f}$$

$$\frac{\mathrm{d}I_{Vs}}{\mathrm{d}t} = \gamma_s C_3 + \phi_s O + \omega_{fs} I_{Vf} + \theta I_{Vs\mathrm{Ca}} - (\omega_s + \omega_{sf} + \lambda_s + \delta) I_{Vs}, \tag{2.50g}$$

$$\frac{\mathrm{d}C_{0\mathrm{Ca}}}{\mathrm{d}t} = \delta C_0 + \beta_0 C_{1\mathrm{Ca}} - (\theta + \alpha_0) C_{0\mathrm{Ca}}, \tag{2.50h}$$

$$\frac{\mathrm{d}C_{1\mathrm{Ca}}}{\mathrm{d}t} = \delta C_1 + \beta_1 C_{2\mathrm{Ca}} + \alpha_0 C_{0\mathrm{Ca}} - (\theta + \beta_0 + \alpha_1) C_{1\mathrm{Ca}}, \tag{2.50i}$$

$$\frac{\mathrm{d}C_{2\mathrm{Ca}}}{\mathrm{d}t} = \delta C_2 + \beta_2 C_{3\mathrm{Ca}} + \alpha_1 C_{1\mathrm{Ca}} - (\theta + \beta_1 + \alpha_2) C_{2\mathrm{Ca}}, \tag{2.50j}$$

$$\frac{\mathrm{d}C_{3\mathrm{Ca}}}{\mathrm{d}t} = \delta C_3 + \beta_3 I_{\mathrm{Ca}} + \alpha_2 C_{2\mathrm{Ca}} + \omega_f I_{Vf} + \omega_s I_{Vs} - (\theta + \beta_2 + \alpha_3 + \gamma_f + \gamma_s) C_{3\mathrm{Ca}}, \tag{2.50k}$$

$$\frac{\mathrm{d}I_{\mathrm{Ca}}}{\mathrm{d}t} = \delta O + \alpha_3 C_{3\mathrm{Ca}} + \lambda_f I_{Vf\mathrm{Ca}} + \lambda_s I_{Vs\mathrm{Ca}} - (\theta + \beta_3 + \phi_f + \phi_s) I_{\mathrm{Ca}}, \tag{2.50l}$$

$$\frac{\mathrm{d}I_{Vf\mathrm{Ca}}}{\mathrm{d}t} = \gamma_f C_{3\mathrm{Ca}} + \phi_f O + \omega_{sf} I_{Vf\mathrm{Ca}} + \delta I_{Vf} - (\omega_f + \omega_{fs} + \lambda_f + \theta) I_{Vf\mathrm{Ca}}, \tag{2.50m}$$

$$\frac{\mathrm{d}I_{Vs\mathrm{Ca}}}{\mathrm{d}t} = \gamma_s C_{3\mathrm{Ca}} + \phi_s O + \omega_{fs} I_{Vf\mathrm{Ca}} + \delta I_{Vs} - (\omega_s + \omega_{sf} + \lambda_s + \theta) I_{Vs\mathrm{Ca}} \tag{2.50n}$$

where the parameters are

$$\alpha = 0.925 \exp(\mathrm{V_m}/30), \tag{2.51a}$$

$$\beta = 0.39 \exp(-\mathrm{V_m}/40), \tag{2.51b}$$

$$\alpha_0 = 4\alpha, \tag{2.51c}$$

$$\alpha_1 = 3\alpha, \tag{2.51d}$$

$$\alpha_2 = 2\alpha, \tag{2.51e}$$

$$\alpha_3 = \alpha, \tag{2.51f}$$

$$\beta_0 = \beta, \tag{2.51g}$$

$$\beta_1 = 2\beta, \tag{2.51h}$$

$$\beta_2 = 3\beta, \tag{2.51i}$$

$$\beta_3 = 4\beta, \tag{2.51j}$$

$$\gamma_f = 0.245 \exp(\mathrm{V_m}/10), \tag{2.51k}$$

$$\gamma_s = 0.005 \exp(-\mathrm{V_m}/40), \tag{2.51l}$$

$$\phi_f = 0.02 \exp(\mathrm{V_m}/500), \tag{2.51m}$$

$$\phi_s = 0.03 \exp(-\mathrm{V_m}/280), \tag{2.51n}$$

$$\lambda_f = 0.035 \exp(-\mathrm{V_m}/300), \tag{2.51o}$$

$$\lambda_s = 0.0011 \exp(V/500), \tag{2.51p}$$

$$\omega_f = (\beta_3\lambda_f\gamma_f)/(\alpha_3\phi_f), \tag{2.51q}$$

$$\omega_s = (\beta_3\lambda_s\gamma_s)/(\alpha_3\phi_s), \tag{2.51r}$$

$$\omega_{sf} = (\lambda_s\phi_f)/\lambda_f, \tag{2.51s}$$

$$\omega_{fs} = \phi_s, \tag{2.51t}$$

$$\delta = \frac{1}{1 + 1/[\mathrm{Ca}^{2+}]_{\mathrm{ss}}}, \tag{2.51u}$$

$$\theta = 0.01. \tag{2.51v}$$

2.4.3 Calcium Current of the Sarcoplasmic Reticulum I_{rel}

The calcium dynamics control the mechanical activity of the cardiac cell. The cell is divided into four compartments as shown in Figure 2.6. Those compartments are junctional cleft – a disk space by the cellular membrane and sarcoplasmic reticulum; sub-membrane space – a narrow compartment close to the cellular membrane; a sarcoplasmic reticulum – intracellular organelle; bulk cytosolic space – the remaining intracellular space.

The Sarcoplasmic reticulum acts as a storage of Ca^{2+} ions inside of the cell. Ryanodine receptor (RyR) is an ion channel in the membrane of the intracellular organelle called sarcoplasmic reticulum. The flux of calcium from the sarcoplasmic reticulum is triggered by increasing calcium concentration in the junctional cleft or inside the sarcoplasmic reticulum. The release of calcium due to increased concentration in the junctional cleft is known as calcium-induced calcium release (CICR). If the sarcoplasmic calcium concentration $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ increases above a threshold, the calcium is released. This mechanism may cause arrhythmogenic phenomena called early after depolarisation.

The calcium release from sarcoplasmic reticulum into the subspace is given by the formula

$$I_{\mathrm{rel}} = G_{\mathrm{rel}} \cdot O_{\mathrm{ryr}} \cdot ([\mathrm{Ca}]_{\mathrm{JSR}} - [\mathrm{Ca}]_{\mathrm{ss}}), \tag{2.52}$$

where $[\mathrm{Ca}]_{\mathrm{JSR}}$ and $[\mathrm{Ca}]_{\mathrm{ss}}$ are the calcium concentration in junctional sarcoplasmic reticulum (JSR), and in subspace respectively, O_{ryr} is open probability of RyR model that is described below, and conductance is given by

$$G_{\mathrm{rel}} = 250 \cdot \left(\frac{\gamma_{\mathrm{rel}}}{\nu_{\mathrm{rel}}} + S_{\mathrm{rel}}\right) \tag{2.53}$$

where

$$\gamma_{\mathrm{rel}} = \left(1 + \exp\left(\frac{20 + I_{\mathrm{Ca(L)}}}{6}\right)\right)^{-1} - 0.034445, \tag{2.54a}$$

$$\nu_{\mathrm{rel}} = 1 + \exp\left(\frac{0.015 \cdot \bar{I}_{\mathrm{Ca}} + 1.25}{0.75}\right), \tag{2.54b}$$

$$\begin{tikzcd}
I_1 \arrow[r, shift left, "\alpha_{1R}"] \arrow[d, shift right, "\delta_{1R}"'] & I_2 \arrow[l, shift left, "\beta_{1R}"] \arrow[r, shift left, "\alpha_{2R}"] \arrow[d, shift right, "\delta_{2R}"'] & I_3 \arrow[l, shift left, "\beta_{2R}"] \arrow[r, shift left, "\alpha_{3R}"] \arrow[d, shift right, "\delta_{3R}"'] & I_4 \arrow[l, shift left, "\beta_{3R}"] \arrow[r, shift left, "\alpha_{4R}"] \arrow[d, shift right, "\delta_{4R}"'] & I_5 \arrow[l, shift left, "\beta_{4R}"] \arrow[d, shift right, "\delta_{5R}"'] \\
C_1 \arrow[r, shift left, "\alpha_{1R}"] \arrow[u, shift right, "\gamma_{1R}"'] & C_2 \arrow[l, shift left, "\beta_{1R}"] \arrow[r, shift left, "\alpha_{2R}"] \arrow[u, shift right, "\gamma_{2R}"'] & C_3 \arrow[l, shift left, "\beta_{2R}"] \arrow[r, shift left, "\alpha_{3R}"] \arrow[u, shift right, "\gamma_{3R}"'] & C_4 \arrow[l, shift left, "\beta_{3R}"] \arrow[r, shift left, "\alpha_{4R}"] \arrow[u, shift right, "\gamma_{4R}"'] & O_1 \arrow[l, shift left, "\beta_{4R}"] \arrow[u, shift right, "\gamma_{5R}"']
\end{tikzcd}$$

Figure 2.14: State diagram of RyR Markov chain model [1].

where $I_{\mathrm{Ca(L)}}$ is defined in (2.49a) and $\bar{I}_{\mathrm{Ca}}$ is maximal calcium current through $I_{\mathrm{Ca}(L)}$ channel.

The variable S_{rel} is omitted in the model equations in the paper, however in the author's code is defined as a dynamical variable dependent on time and calcium concentration in JSR ($[\mathrm{Ca}]_{\mathrm{JSR}}$).

Figure 2.14 shows the Markov chain diagram of the RyR by Faber *et al.* (2007). This model contains 10 interconnected states located in 2 rows – corresponding to inactivated and activated states. The state O_1 is the only conductive state. The release of Ca^{2+} from the RyR is measured in mM, unlike in membrane currents, where the current is measured in $\mu\mathrm{A}/\mu\mathrm{F}$.

According to Faber *et al.* (2007) [1] the RyR model is described by the following system of ordinary differential equations (ODEs)

$$\frac{\mathrm{d}C_1}{\mathrm{d}t} = \beta_{1R}C_2 + \delta_{1R}I_1 - (\alpha_{1R} + \gamma_{1R})C_1, \tag{2.55a}$$

$$\frac{\mathrm{d}C_2}{\mathrm{d}t} = \alpha_{1R}C_1 + \beta_{2R}C_3 + \delta_{2R}I_2 - (\beta_{1R} + \alpha_{2R} + \gamma_{2R})C_2, \tag{2.55b}$$

$$\frac{\mathrm{d}C_3}{\mathrm{d}t} = \alpha_{2R}C_2 + \beta_{3R}C_4 + \delta_{3R}I_3 - (\beta_{2R} + \alpha_{3R} + \gamma_{3R})C_3, \tag{2.55c}$$

$$\frac{\mathrm{d}C_4}{\mathrm{d}t} = \beta_{4R}O + \alpha_{3R}C_3 + \delta_{4R}I_4 - (\beta_{3R} + \alpha_{4R} + \gamma_{4R})C_4, \tag{2.55d}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{4R}C_4 + \delta_{5R}I_5 - (\gamma_{5R} + \beta_{4R})O, \tag{2.55e}$$

$$\frac{\mathrm{d}I_1}{\mathrm{d}t} = \gamma_{1R}C_1 + \beta_{1R}I_2 - (\delta_{1R} + \alpha_{1R})I_1, \tag{2.55f}$$

$$\frac{\mathrm{d}I_2}{\mathrm{d}t} = \gamma_{2R}C_2 + \alpha_{1R}I_1 + \beta_{2R}I_3 - (\delta_{2R} + \beta_{1R} + \alpha_{2R})I_2, \tag{2.55g}$$

$$\frac{\mathrm{d}I_3}{\mathrm{d}t} = \gamma_{3R}C_3 + \alpha_{2R}I_2 + \beta_{3R}I_4 - (\delta_{3R} + \beta_{2R} + \alpha_{3R})I_3, \tag{2.55h}$$

$$\frac{\mathrm{d}I_4}{\mathrm{d}t} = \gamma_{4R}C_4 + \alpha_{3R}I_3 + \beta_{4R}I_5 - (\delta_{4R} + \beta_{3R} + \alpha_{4R})I_4, \tag{2.55i}$$

$$\frac{\mathrm{d}I_5}{\mathrm{d}t} = \gamma_{5R}O + \alpha_{4R}I_4 - (\delta_{5R} + \beta_{4R})I_5 \tag{2.55j}$$

where the transition rates are described by functions dependent on two calcium concentrations as

$$\alpha_{1R} = 1750[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56a}$$

$$\alpha_{2R} = 5600[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56b}$$

$$\alpha_{3R} = 5600[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56c}$$

$$\alpha_{4R} = 5600[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56d}$$

$$\beta_{1R} = 5, \tag{2.56e}$$

$$\beta_{2R} = 2.62, \tag{2.56f}$$

$$\beta_{3R} = 1, \tag{2.56g}$$

$$\beta_{4R} = 6.25, \tag{2.56h}$$

$$\gamma_{1R} = 0.4[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56i}$$

$$\gamma_{2R} = 1.2[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56j}$$

$$\gamma_{3R} = 2.8[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56k}$$

$$\gamma_{4R} = 5.2[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56l}$$

$$\gamma_{5R} = 8.4[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{2.56m}$$

$$\delta_{1R} = \frac{0.01}{1+\left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{2.56n}$$

$$\delta_{2R} = \frac{0.001}{1+\left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{2.56o}$$

$$\delta_{3R} = \frac{0.0001}{1+\left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{2.56p}$$

$$\delta_{4R} = \frac{0.00001}{1+\left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{2.56q}$$

$$\delta_{5R} = \frac{0.000001}{1+\left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9} \tag{2.56r}$$

where $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$ is the calcium concentration in the subspace and CSQN is calsequestrin buffered calcium concentration in junctional space, and $\overline{\mathrm{CSQN}} = 10\ \mathrm{mM}$ is the maximal amount of calcium that can be buffered by calsequestrin.

2.4.4 Calcium Dynamics in Faber et al. (2007) Model

In the Faber et al. (2007) model the cell is divided into compartments. The intracellular space contains the sarcoplasmic reticulum which stores the calcium, and intracellular space. The sarcoplasmic reticulum is further divided into two compartments called junctional sarcoplasmic reticulum (JSR) and network sarcoplasmic reticulum (NSR).

The calcium channels controlling the current $I_{\mathrm{Ca}(L)}$ from the extracellular space are concentrated in so-called T-tubules, which increase the surface area to allow a quick pathway for entry of calcium. The JSR membrane is located close to the T-tubules from the inside of the cellular membrane in so-called subspace which comprises 2% of the myocyte.

The RyR receptors lead to the subspace and sense the increase in the calcium concentration and react by releasing the calcium from the sarcoplasmic reticulum. This process is known as calcium induced calcium release (CICR). The calcium concentration in the subspace is much greater than those observed in bulk myoplasm, and so diffuses into the myoplasm where it affects intracellular calcium concentration.

The calcium in the intracellular subspace binds to buffer molecules of troponin and calmodulin. The calcium in JSR binds to buffer mollecules of calsequestrin. Only free calcium is available for the current through the calcium channels.

The calcium is then removed from the intracellular to extracellular space by calcium pump and sodium-calcium exchanger.

The concentration corresponding to the junctional space of sarcoplasmic reticulum (JSR) is given as

$$\frac{\mathrm{d}[\mathrm{Ca}]_{\mathrm{JSR}}}{\mathrm{d}t} = \beta_{\mathrm{JSR}} \cdot (I_{\mathrm{tr}} - I_{\mathrm{rel}}), \tag{2.57}$$

where I_{tr} is a current from NSR into JSR defined as

$$I_{\mathrm{tr}} = \frac{[\mathrm{Ca}]_{\mathrm{NSR}} - [\mathrm{Ca}]_{\mathrm{JSR}}}{\tau_{\mathrm{tr}}}, \tag{2.58}$$

where $\tau_{\mathrm{tr}} = 120$ ms is a time constant of calcium transfer from NSR to JSR, I_{rel} is given by (2.52) and coefficient

$$\beta_{\mathrm{JSR}} = \left(1 + \frac{[\overline{csqn}] \cdot K_{\mathrm{m,csqn}}}{(K_{\mathrm{m,csqn}} + [\mathrm{Ca}]_{\mathrm{JSR}})^2}\right)^{-1}, \tag{2.59}$$

where $[\overline{csqn}] = 10$ mM is total concentration of calsequestrin in the SR and $K_{\mathrm{m,csqn}} = 0.8$ mM is equilibrium constant for calsequestrin.

The calcium concentration in the subspace and is given by

$$\frac{\mathrm{d}[\mathrm{Ca}]_{\mathrm{ss}}}{\mathrm{d}t} = -\beta_{\mathrm{ss}} \cdot \left((I_{\mathrm{Ca(L)}} - 2 \cdot I_{\mathrm{NaCa,ss}}) \frac{A_{\mathrm{cap}}}{2 \cdot V_{\mathrm{ss}} F} - I_{\mathrm{rel}} \frac{V_{\mathrm{JSR}}}{V_{\mathrm{ss}}} + I_{\mathrm{diff,Ca}}\right) \tag{2.60}$$

where F is Faraday constant, $A_{\mathrm{cap}} = 1.534 \cdot 10^{-4}$ cm^2 is the capacitive membrane area, $V_{\mathrm{ss}} = 7.602 \cdot 10^{-7}$ μL and $V_{\mathrm{JSR}} = 1.824 \cdot 10^{-7}$ μL are the volumes of sub-space and JSR respectively. The $I_{\mathrm{Ca(L)}}$ was defined in (2.49a). The $I_{\mathrm{NaCa,ss}}$ is sodium-calcium exchanger in the subspace. The diffusion between the subspace and bulk intracellular space (myoplasm) is defined as

$$I_{\mathrm{diff,Ca}} = \frac{[\mathrm{Ca}]_{\mathrm{ss}} - [\mathrm{Ca}]_{\mathrm{i}}}{\tau_{\mathrm{diff}}} \tag{2.61}$$

where $\tau_{\mathrm{diff}} = 0.1$ ms is a time constant of ion transfer from subspace to myoplasm, and $[\mathrm{Ca}]_{\mathrm{i}}$ is myoplasmic calcium concentration. The coefficient in (2.60) is defined as

$$\beta_{\mathrm{ss}} = \left(1 + \frac{\overline{\mathrm{BSR}} \cdot K_{\mathrm{m,BSR}}}{(K_{\mathrm{m,BSR}} + [\mathrm{Ca}]_{\mathrm{ss}})^2} + \frac{\overline{\mathrm{BSL}} \cdot K_{\mathrm{m,BSL}}}{(K_{\mathrm{m,BSL}} + [\mathrm{Ca}]_{\mathrm{ss}})^2}\right)^{-1}, \tag{2.62}$$

where $\overline{\mathrm{BSR}} = 0.047$ mM, $K_{\mathrm{m,BSR}} = 0.00087$ mM, $\overline{\mathrm{BSL}} = 2.124$ mM, and $K_{\mathrm{m,BSL}} = 0.127$ mM.

The calcium concentration in the NSR space of SR is

$$\frac{\mathrm{d}[\mathrm{Ca}]_{\mathrm{NSR}}}{\mathrm{d}t} = I_{\mathrm{up}} - I_{\mathrm{leak}} - I_{\mathrm{tr}} \frac{V_{\mathrm{JSR}}}{V_{\mathrm{NSR}}} \tag{2.63}$$

where $V_{\mathrm{NSR}} = 2.098 \cdot 10^{-6}$ μL and the currents

$$I_{\mathrm{leak}} = \frac{0.00875}{15}[\mathrm{Ca}]_{\mathrm{NSR}}, \tag{2.64a}$$

$$I_{\mathrm{up}} = \frac{[\mathrm{Ca}]_{\mathrm{i}}}{[\mathrm{Ca}]_{\mathrm{i}} + K_{\mathrm{m,up}}} \overline{I_{\mathrm{up}}} \tag{2.64b}$$

where $K_{\mathrm{m,up}} = 0.00092$ mM is half-saturation concentration of I_{up}, and $\overline{I_{\mathrm{up}}} = 0.017325$ mM/ms is maximal current through I_{up} channel.

The calcium concentration in bulk myoplasm is given by

$$\begin{aligned}\frac{\mathrm{d}[\mathrm{Ca}]_{\mathrm{i}}}{\mathrm{d}t} = -\beta_{\mathrm{myo}}\Bigg(& (I_{\mathrm{total,Ca}} - I_{lca} + 2I_{\mathrm{NaCa,ss}}) \frac{A_{\mathrm{cap}}}{V_{\mathrm{myo}} z_{\mathrm{Ca}} F} \\ & + (I_{\mathrm{up}} - I_{\mathrm{leak}}) \frac{V_{\mathrm{NSR}}}{V_{\mathrm{myo}}} - I_{\mathrm{diff,Ca}} \frac{V_{\mathrm{ss}}}{V_{\mathrm{myo}}}\Bigg)\end{aligned} \tag{2.65}$$

where $V_{\mathrm{myo}} = 2.584 \cdot 10^{-5}$ μL is the volume of myoplasm, $z_{\mathrm{Ca}} = 2$ is valence of calcium ions, $I_{\mathrm{total,Ca}}$ total calcium ion flow, and coefficient

$$\beta_{\mathrm{myo}} = \left(1 + \frac{([\overline{trpn}] K_{\mathrm{m,trpn}})}{(K_{\mathrm{m,trpn}} + [\mathrm{Ca}]_{\mathrm{i}})^2} + \frac{[\overline{cmdn}] K_{\mathrm{m,cmdn}}}{(K_{\mathrm{m,cmdn}} + [\mathrm{Ca}]_{\mathrm{i}})^2}\right)^{-1} \tag{2.66}$$

where $[\overline{cmdn}] = 0.050$ mM, and $[\overline{trpn}] = 0.070$ mM are maximal calcium concentrations buffered in calmodulin and troponin respectively, $K_{\mathrm{m,cmdn}} = 0.00238$ mM, and $K_{\mathrm{m,trpn}} = 0.0005$ mM are equilibrium constants of buffering for calmodulin and troponin respectively.

Chapter **3**

Asymptotic and Numerical Methods

This chapter introduces numerical and asymptotic methods which are used in this thesis. Section 3.1 discusses asymptotic methods and describes the reduction of dimensionality of a system based on the division into fast and slow subsystems. The theory follows the ideas described in Tikhonov (1952) [14], Fenichel (1979) [15] and Biktashev (2003) [16]. The section is divided into dimensionality reduction for the leading-order of linear non-autonomous system, then a theory for developing a first-order correction term which aims to improve the accuracy of the leading-order approximation. The last part develops theory for application of the dimensionality reduction on a Markov chain.

Section 3.2 presents numerical integration methods for systems of ordinary differential equations (ODEs) and analysis of the order of numerical errors which arise.

3.1 Asymptotic methods

3.1.1 Toy Example of Dimensionality Reduction

This subsection introduces the of dimensionality reduction that will later be applied to the Markov chains on a simple toy model of two dimensions. This model reads as

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = -\left(1 + \frac{2\varepsilon}{3}\right) x_1 + \left(2 - \frac{2\varepsilon}{3}\right) x_2, \tag{3.1}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = \left(1 - \frac{\varepsilon}{3}\right) x_1 - \left(2 + \frac{\varepsilon}{3}\right) x_2, \tag{3.2}$$

which has a form

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = (\boldsymbol{A}_0 + \varepsilon \boldsymbol{A}_1)\,\vec{x}, \tag{3.3}$$

where

$$\boldsymbol{A}_0 = \begin{bmatrix} -1 & 2 \\ 1 & -2 \end{bmatrix}, \qquad \boldsymbol{A}_1 = \frac{1}{3}\begin{bmatrix} -2 & -2 \\ -1 & -1 \end{bmatrix}, \qquad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{3.4}$$

First, we consider a case where $\varepsilon = 0$ then the system (3.3) writes as

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = -x_1 + 2x_2, \tag{3.5a}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = x_1 - 2x_2. \tag{3.5b}$$

The matrix $\boldsymbol{A}_0$ has eigenvalues $\lambda_1 = 0$, $\lambda_2 = -3$ and corresponding right $\vec{v}_k$ and left (adjoint) eigenvectors $\vec{w}_k$ (for $k = 1, 2$) as

$$\vec{v}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \qquad \vec{w}_1 = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{3.6a}$$

$$\vec{v}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \qquad \vec{w}_2 = \frac{1}{3}\begin{bmatrix} -1 \\ 2 \end{bmatrix}. \tag{3.6b}$$

Then the system has a solution

$$\vec{x}(t) = \sum_{k=1}^{2} c_k \vec{v}_k \exp(\lambda_k t) = \begin{bmatrix} \vec{v}_1 | \vec{v}_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \exp(\lambda_2 t) \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \exp(-3t) \end{bmatrix}, \tag{3.7}$$

where c_1, c_2 are constants determined by solving the initial value problem for the initial point $P = [x_1(0), x_2(0)]$.

This can be characterised by the phase portrait shown in Figure 3.1(a). So the system can be explicitly described as a superposition of two modes, one of which decays with time. The matrix has one zero eigenvalue, i.e. it is degenerate. The eigenvector corresponding to the zero eigenvalue $\vec{v}_1$ determines the centre manifold. The other eigenvector $\vec{v}_2$ corresponds to the negative eigenvalue which means that the solution decays with time and approaches the central manifold.

Now we to obtain a equivalent system in its orthogonal basis. We transform the original system (3.5) to the new system by multiplying by a transpose of adjoint eigenvalue matrix $\boldsymbol{W}^T = [\vec{w}_1 | \vec{w}_2]^T$, which satisfies $\boldsymbol{W}^T \boldsymbol{A}_0 = \boldsymbol{W}^T \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ contains the eigenvalues on its diagonal ($\lambda_1 = 0, \lambda_2 = -3$), so we have eigenvector

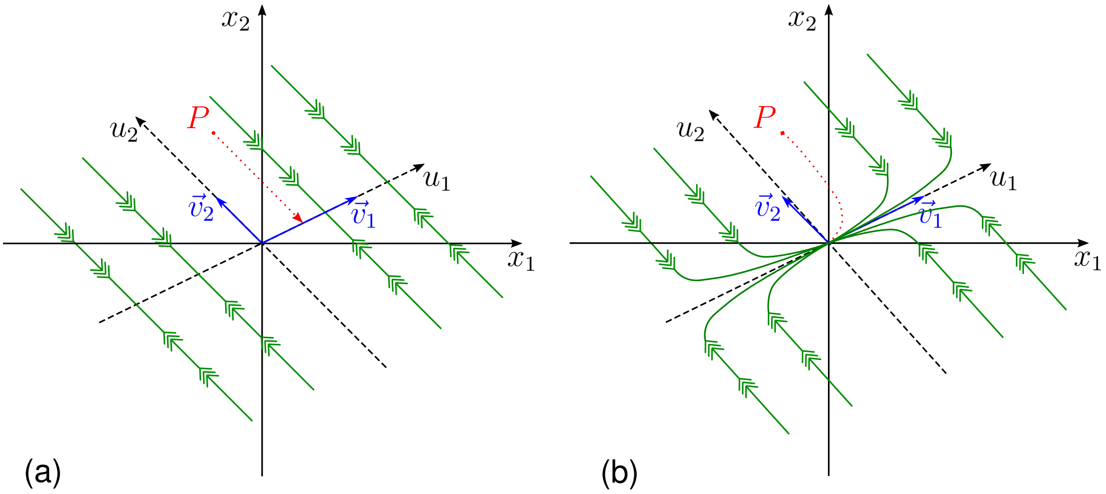


Figure 3.1: Phase portrait of the system defined by (3.3) and (3.5). The axis x_1 and x_2 are the original coordinates of the system. The axis u_1 and u_2 correspond to the new coordinates of the system. The vectors $\vec{v}_1$ and $\vec{v}_2$ correspond to the eigenvectors of the system. The green lines represent the flow. Given initial conditions of a point P the trajectory will be represented by the red line. Panel (a) shows the situation when the $\varepsilon = 0$ as shown in (3.5) and panel (b) shows situation when $\varepsilon \neq 0$ as described in (3.15).

and eigenvalue matrices as

$$\boldsymbol{V} = \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix}, \qquad \boldsymbol{W}^T = \frac{1}{3}\begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix}, \qquad \boldsymbol{\Lambda} = \begin{bmatrix} 0 & 0 \\ 0 & -3 \end{bmatrix}. \tag{3.8}$$

Then from the eigenvalue identity, we obtain

$$\boldsymbol{W}^T \frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \boldsymbol{W}^T \boldsymbol{A}_0 \vec{x} = \boldsymbol{\Lambda} \boldsymbol{W}^T \vec{x}. \tag{3.9}$$

We define a transformation from old to new state variables and the other way round as

$$\vec{u} = \boldsymbol{W}^T \vec{x}, \qquad\qquad \vec{x} = \boldsymbol{V}\vec{u}. \tag{3.10}$$

which gives us the system

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{\Lambda}\vec{u}, \tag{3.11}$$

which expands as

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} =0, \tag{3.12a}$$

$$\frac{\mathrm{d}u_2}{\mathrm{d}t} =-3u_2. \tag{3.12b}$$

The solution of this system is given by

$$\vec{u}(t) = \begin{bmatrix} c_1 \\ c_2 \exp(-3t) \end{bmatrix}, \tag{3.13}$$

where $\vec{u} = [u_1, u_2]^T$. This system has one zero eigenvalue, which is reflected by a zero derivative of u_1. This means that there is no flow around the corresponding axis. From this relation we see, that at the limit $t \to \infty$ the $\vec{u} \to [c_1, 0]^T$ and the transformation (3.10) rewrites as

$$\vec{x} = \vec{v}_1 u_1 = \begin{bmatrix} 2c_1 \\ c_1 \end{bmatrix}. \tag{3.14}$$

The equations describing the system are uncoupled and from the equation (3.12a) we see, that u_1 remains constant. The evolution of the u_2 variable can be characterised as a decay to the line given by the eigenvector u_1. This is the simplest case of the reduction which allows us to reduce the dimensionality of the system.

More interesting behaviour can be observed for $\varepsilon \neq 0$ in (3.3). Then we obtain a system

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \left(\begin{bmatrix} -1 & 2 \\ 1 & -2 \end{bmatrix} + \frac{\varepsilon}{3}\begin{bmatrix} -2 & -2 \\ -1 & -1 \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{3.15}$$

Using the same approach as in the first example, we pre-multiply by the eigenvector matrix $\boldsymbol{W}^T$ and transform the variables according to (3.10) to get

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \left(\boldsymbol{W}^T \boldsymbol{A}_0 \boldsymbol{V} + \varepsilon \boldsymbol{W}^T \boldsymbol{A}_1 \boldsymbol{V} \right) \vec{u}, \tag{3.16a}$$

the first term on the RHS is equivalent to (3.9). The second term is computed as

$$\varepsilon \boldsymbol{W}^T \boldsymbol{A}_1 \boldsymbol{V} = \varepsilon \frac{1}{3} \begin{bmatrix} 1 & 1 \\ -1 & 2 \end{bmatrix} \frac{1}{3} \begin{bmatrix} -2 & -2 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} \varepsilon & 0 \\ 0 & 0 \end{bmatrix}, \tag{3.17}$$

collecting these two terms together yields a system

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = \varepsilon u_1, \tag{3.18a}$$

$$\frac{\mathrm{d}u_2}{\mathrm{d}t} = -3u_2. \tag{3.18b}$$

That is exactly equivalent to the original system (3.15).

Again as in (3.12), the equations describing the system are uncoupled. Unlike the system (3.12) now, u_1 varies slowly for a small ε. This can be better characterised by a reformulation of the (3.18) using a slow-time variable $T = \varepsilon t$ as

$$\frac{\mathrm{d}u_1}{\mathrm{d}T} = u_1, \tag{3.19a}$$

$$\varepsilon\frac{\mathrm{d}u_2}{\mathrm{d}T} = -3u_2. \tag{3.19b}$$

The systems (3.18) and (3.19) are equivalent to each other for $\varepsilon > 0$. Considering the limit $\varepsilon \to 0$ in (3.19) we obtain

$$\frac{\mathrm{d}u_1}{\mathrm{d}T} = u_1, \tag{3.20a}$$

$$0 = -3u_2. \tag{3.20b}$$

In other words, if the parameter ε is small, the dynamics of the u_2 compared to u_1 will be much faster, and will decay with time. Hence, the variable u_2 can be neglected in the time-scale of u_1 and instead of the original system we get a single dynamical equation (3.20a). The transformation to the original coordinates is done using (3.14).

3.1.2 Classical Formulations of Singular Perturbation Theory

The ideas illustrated on the toy model in the previous section and which will be used throughout this and the following chapter are based on singular perturbation theory. These methods have been traditionally used to eliminate the complexity in chemical reaction by quasi-stationary approximation, in physics as adiabatic elimination, and other areas as celestial mechanics. Tikhonov (1952) has systematised the procedures and introduced them as a mathematical theory. Below we give a brief exposition of his results [14].

Tikhonov introduced a system similar to (3.19), which he called a full system, as

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{f}(\vec{x}, \vec{z}^{(1)}, \ldots, \vec{z}^{(m)}, t), \qquad (j = 1, 2, \ldots, m), \tag{3.21a}$$

$$\mu^{(j)}\frac{\mathrm{d}\vec{z}^{(j)}}{\mathrm{d}t} = \vec{F}^{(j)}(\vec{x}, \vec{z}^{(1)}, \ldots, \vec{z}^{(m)}, t). \tag{3.21b}$$

The main interest of the paper was to study the solutions of the initial value problem in a case when $\mu_j \to 0$. (The μ_j corresponds to ε in the previous subsection.)

In the simplest case Tikhonov considered and a system with one parameter μ_1 at its zero limit. From (3.21b) we solve $\vec{F}(\vec{x}, \vec{z}, t) = 0$ for $\vec{z}$, which defines *manifold of equilibria*, and let the root to be $\vec{z} = \vec{\phi}(\vec{x}, t)$. This leads to the following degenerate system

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{f}(\vec{x}, \vec{z}, t), \qquad \vec{x}(t_0) = \vec{x}_0, \tag{3.22a}$$

$$\vec{z} = \vec{\phi}(\vec{x}, t). \tag{3.22b}$$

The degenerate system has an *attached system* which is considered in a fast time $\tau = t/\mu$ as

$$\frac{\mathrm{d}\vec{z}}{\mathrm{d}\tau} = \vec{F}(\vec{x}, \vec{z}, t), \tag{3.23}$$

where $\vec{x}$ and t are considered as parameters.

Assuming that all functions are continuous and differential equations have unique solutions, Tikhonov has proven that the solution of the full system approaches the solutions of the degenerate system for $\mu_1 \to 0$, if

1. the root $\vec{z} = \vec{\phi}(\vec{x}, t)$ is a stable root of the attached system;
2. the initial conditions $\vec{z}_0$ are within the basin of attraction of the root $\vec{z} = \vec{\phi}(\vec{x}, t)$ at the initial values $\vec{x}_0$ and t_0.

The implication of this result is that for a small enough values of μ_1 the full system can be replaced by the degenerate system which has fewer dimensions. Hence the procedure of solving the system simplifies.

The simplification is most useful in cases, when the full system cannot be solved analytically and the degenerate system has an analytical solution. Other convenient application is when the computational time spend for solving the system numerically is lower for the degenerate system than for the full system.

In our application to the Markov chain we hypothesise that the numerical solution of the full system is more time consuming than a numerical solution of some particular degenerate system. In the text in the following chapter, we call the degenerate system of a Markov chain a “reduced” model.

Tikhonov also considers a situation with more than one parameter μ_j, however, this is not discussed here, as we did not consider such case in the following text.

The limitation of the Tikhonov approach is that the system can be only degenerate along one of the axis of the system, i.e. variables $\vec{z}^{(j)}$. This was addressed in the paper by Fenichel (1979) [15], where the flow can have Tikhonov structure locally around manifold in properly chosen coordinate system. Also, Fenichel discusses higher order terms, which improve asymptotic accuracy. The Fenichel's paper is not directly used in this text, however, we refer to it for completeness, as a relevant prominent work in the area of singular perturbation theory. For our purposes we will follow the convention used by Biktashev (2003) [16] as it depends on fewer technical details and presents a simpler notation.

In the case of the Markov chain, the Tikhonov approach has to be modified because Markov chains do not have Tikhonov structure as the fast and slow variables are both present within each dynamical equation. An attempt of a direct application of this theory would lead to a violation of a *conservation law* observed in Markov chain models. The conservation law constrains the sum of the states to be equal to one. This also implies that the sum of the right hand sides must be zero in order to maintain the conservation law for the time evolution of the system.

Perturbing dynamical equations of a Markov chain in the Tikhonov manner, destroys the balance as the sum of the right hand sides cease to be equal to zero. Therefore, the perturbed system violates the conservation law.

To address this issue, we transform the system in the base of eigenvectors. This is done by selecting specific fast transition rates in the original system. Then the resulting system can be divided into a part that varies in a slow time scale and part that corresponds to the selected fast transition rates. The differential equations of the transformed system allow direct perturbation according to Tikhonov.

In the following text we reproduce some of the ideas from the cited papers. However, for the readers convenience we present the derivation of the formulas in a such way that no further knowledge of the cited papers is required.

In Section 3.1.3 we describe a derivation of leading-order formulas for a system of linear non-homogeneous differential equations. In this case we think of a part of a Markov chain model, that contains fast transition rates. The homogeneous part of the model corresponds to the dynamical variables linked with the fast transition rates. The non-homogeneous part correspond to the connections of those states to the adjacent states in the topology of the model. The fast subsystem is then replaced by a single differential equation.

In Section 3.1.4 we present a generalised approach to develop higher-order terms, to improve the asymptotic accuracy of the solution. In that section we use the notation as introduced by Biktashev (2003) [16].

The Section 3.1.5 presents the application of the theory developed in the previous Section 3.1.4 to a general Markov chain model. Unlike Section 3.1.3, Section 3.1.5 develops equations up to first-order of the small parameter ε.

3.1.3 Leading-Order Reduction for Linear Systems

Definition of the System

Consider a linear non-homogeneous system of η ordinary differential equations

$$\frac{\mathrm{d}\vec{X}}{\mathrm{d}t} = \boldsymbol{A}(t,\varepsilon)\vec{X}(t) + \vec{H}(t), \quad \vec{X}(t), \vec{H}(t) \in \mathbb{R}^{\eta}, \boldsymbol{A}(t,\varepsilon) \in \mathbb{R}^{\eta\times\eta}, \tag{3.24}$$

where $\vec{X}(t)$ is the vector of dynamical variables, $\boldsymbol{A}(t,\varepsilon)$ is a matrix of coefficients, $\vec{H}$ is a vector of non-homogeneous terms accounting for contribution from variables not included in $\vec{X}$, and ε is a small parameter. This parameter characterises the separation between the slow and the fast time scales.

Division to Time Scales

To analyse the fast behaviour of this system in the vicinity of time point t_0, we introduce fast time variable τ such that $t = t_0 + \varepsilon\tau$, where $\varepsilon \to 0$ is a small parameter. Now the dynamical variable $\vec{X}(t)$ is expanded using slow and fast time as $\vec{X}(t) = \vec{Y}(t, (t - t_0)/\varepsilon) = \vec{Y}(t, \tau)$ where $\vec{Y}(t, \tau) \in \mathrm{R}^{\eta}$.

The system (3.24) is reformulated in terms of parameter ε using the chain derivative rule as

$$\varepsilon\left(\frac{\partial\vec{Y}}{\partial t} + \frac{1}{\varepsilon}\frac{\partial\vec{Y}}{\partial t}\right) = \varepsilon\boldsymbol{A}(t,\varepsilon)\vec{Y}(t,\tau) + \varepsilon\vec{H}(t). \tag{3.25}$$

The coefficient matrix $\boldsymbol{A}(t, \varepsilon)$ and dynamical variable $\vec{Y}(t, \tau)$ are expanded as a power series in ε as

$$\boldsymbol{A}(t,\varepsilon) = \sum_{\ell}\varepsilon^{\ell-1}\boldsymbol{A}_{\ell}(t) = \frac{1}{\varepsilon}\left(\boldsymbol{A}_0(t) + \varepsilon\boldsymbol{A}_1(t) + \varepsilon^2\boldsymbol{A}_2(t) + \ldots\right), \tag{3.26a}$$

$$\vec{Y}(t,\tau) = \sum_{\ell}\varepsilon^{\ell}\vec{Y}_{\ell}(t,\tau) = \vec{Y}_0(t,\tau) + \varepsilon\vec{Y}_1(t,\tau) + \varepsilon^2\vec{Y}_2(t,\tau) + \ldots \tag{3.26b}$$

where each matrix $\boldsymbol{A}_{\ell}(t)$ describes the behaviour of the system in a time scale $\varepsilon^{\ell-1}t$. The expansion of $\boldsymbol{A}$ starts from $\mathcal{O}(\varepsilon^{-1})$ to account for fast processes. After substitution of (3.26) into (3.25) we obtain

$$\begin{aligned}\varepsilon\frac{\partial\vec{Y}_0}{\partial t} + \frac{\partial\vec{Y}_0}{\partial t} + \varepsilon\frac{\partial\vec{Y}_1}{\partial t} =& \\ =& \boldsymbol{A}_0(t)\vec{Y}_0(t,\tau) + \varepsilon\boldsymbol{A}_1(t)\vec{Y}_0(t,\tau) + \varepsilon\boldsymbol{A}_0(t)\vec{Y}_1(t,\tau) + \varepsilon\vec{H}(t) + \mathcal{O}(\varepsilon^2).\end{aligned} \tag{3.27}$$

Collecting the terms according to orders of ε gives us

$$\mathcal{O}(\varepsilon^0): \qquad \frac{\partial\vec{Y}_0}{\partial t} = \boldsymbol{A}_0(t)\vec{Y}_0(t,\tau), \tag{3.28a}$$

$$\mathcal{O}(\varepsilon^1): \qquad \frac{\partial\vec{Y}_0}{\partial t} + \frac{\partial\vec{Y}_1}{\partial t} = \boldsymbol{A}_1(t)\vec{Y}_0(t,\tau) + \boldsymbol{A}_0(t)\vec{Y}_1(t,\tau) + \vec{H}(t), \tag{3.28b}$$

$$\mathcal{O}(\varepsilon^2): \qquad \vdots$$

where the $\mathcal{O}(\varepsilon^0)$ defines the model in the fast time scale τ, and $\mathcal{O}(\varepsilon^1)$ in the slow time scale t.

Selection of Eigenvectors and Eigenvalues

For the solution of our system we will need to find the eigenvalues of the matrix $\boldsymbol{A}_0$. The eigenvectors and eigenvalues of matrix $\boldsymbol{A}_0$ have to satisfy the condition

$$\boldsymbol{A}_0(t)\vec{u}_k(t) = \lambda_k(t)\vec{u}_k(t), \qquad \boldsymbol{A}_0(t) \in \mathbb{R}^{\eta\times\eta}, \vec{u}_k(t) \in \mathbb{R}^{\eta}, \lambda_k(t)\mathbb{R}, \tag{3.29}$$

where λ are the eigenvalues and u_k are the eigenvectors.

We assume that the eigenvalues of the matrix $\boldsymbol{A}_0$ are real (as will be proven below). We also assume that one zero eigenvalue $\lambda = 0$, and all the remaining eigenvalues are negative.

A Markov chain has always at least one zero eigenvalue due to the structure of the transition matrix which satisfies the state conservation law, i.e. the sum of entries in each column is zero. The state conservation law is expressed as

$$\boldsymbol{A}_0^T \hat{\imath} = 0, \tag{3.30}$$

where $\hat{\imath}$ is a unit vector with 1 in all entries. This means that $\lambda_0 = 0$ must be an eigenvalue of the system because the $\boldsymbol{A}_0$ and its transpose $\boldsymbol{A}_0^T$ have the same set of eigenvalues. We assign an index $j = 0$ to the zero eigenvalue $\lambda_0(t) = 0$. Further, we assume that the multiplicity of all eigenvalues is $m = 1$.

The equation for a single dynamical state can be expressed in the following form [18]

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = \sum_{j \neq i} (\alpha_{ji} x_j - \alpha_{ij} x_i), \tag{3.31}$$

where $\alpha_{ji} = A_{ji}$ and the sum is over all indices j, when $j \neq i$.

A detailed balance conditions state that in an equilibrium $\vec{x} = \vec{x}^{(eq)}$ each of the elementary sums in (3.31) is equal to zero. This can be also expressed as

$$\alpha_{ji} x_j^{(\mathrm{eq})} = \alpha_{ij} x_i^{(\mathrm{eq})}. \tag{3.32}$$

Assuming that the detailed balance condition is satisfied, we can rescale the transition rates matrix as

$$\tilde{A}_{ij} = \frac{\alpha_{ij} x_j^{(\mathrm{eq})}}{\sqrt{x_i^{(\mathrm{eq})} x_j^{(\mathrm{eq})}}}. \tag{3.33}$$

which means that the the rescaled eigenvalue matrix is symmetric and hence has real eigenvalues. Now, it remains to prove, that the eigenvalues of the rescaled matrix $\tilde{\boldsymbol{A}}$ are identical to the original matrix $\boldsymbol{A}$.

The eigenvalue problem of the rescaled transition matrix $\tilde{\boldsymbol{A}}$ is defined as

$$\sum_j \tilde{A}_{ij} \vec{\gamma}_j = \lambda \vec{\gamma}_i \tag{3.34}$$

Rescaling the eigenvectors as $\vec{\gamma}_i = \vec{v}_i/\sqrt{x_i^{(\mathrm{eq})}}$ and substituting the $\tilde{\boldsymbol{A}}$ from (3.33) to (3.34) we get

$$\text{RHS: } \sum_j \tilde{A}_{ij}\vec{\gamma}_j = \sum_j \frac{\alpha_{ij}x_j^{(\mathrm{eq})}}{\sqrt{x_i^{(\mathrm{eq})}x_j^{(\mathrm{eq})}}}\frac{\vec{v}_j}{\sqrt{x_j^{(\mathrm{eq})}}} = \sum_j \frac{\alpha_{ij}}{\sqrt{x_i^{(\mathrm{eq})}}}\vec{v}_j, \tag{3.35}$$

$$\text{LHS: } \lambda\vec{\gamma}_i = \lambda\frac{\vec{v}_i}{\sqrt{x_i^{(\mathrm{eq})}}}. \tag{3.36}$$

Dividing both sides by $\sqrt{x_i^{(\mathrm{eq})}}$ we get the eigenvalue problem of the matrix $\boldsymbol{A}$, and therefore both matrices $\boldsymbol{A}$ and $\tilde{\boldsymbol{A}}$ have the same set of eigenvalues, and hence also entries of $\boldsymbol{A}$ are real.

Having found all η eigenvalues and eigenvectors of the $\boldsymbol{A}_0$, we can write the diagonal eigenvalue matrix as

$$\boldsymbol{\Lambda}(t) = \begin{bmatrix} \lambda_0(t) & 0 & \dots & 0 \\ 0 & \lambda_1(t) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_\eta(t) \end{bmatrix} \tag{3.37}$$

and the eigenvector matrix, by concatenating the eigenvectors in columns, as

$$\tilde{\boldsymbol{P}}(t) = [\vec{u}_0(t), \vec{u}_1(t), \dots, \vec{u}_\eta(t)] \tag{3.38}$$

where the corresponding eigenvalues and eigenvectors are represented by the same indices. We can then write $\boldsymbol{A}_0(t)\tilde{\boldsymbol{P}}(t) = \tilde{\boldsymbol{P}}(t)\boldsymbol{\Lambda}(t)$.

We find the inverse matrix of $\tilde{\boldsymbol{P}}(t)$ which corresponds to the adjoint vectors to the eigenvectors. The rows of such a matrix are denoted dual basis $\vec{\omega}_j^{\mathrm{T}}(t)$ of eigenvectors $\vec{u}_j(t)$ as

$$\tilde{\boldsymbol{P}}^{-1}(t) = \begin{bmatrix} \vec{\omega}_0^{\mathrm{T}}(t) \\ \vec{\omega}_1^{\mathrm{T}}(t) \\ \vdots \\ \vec{\omega}_\eta^{\mathrm{T}}(t) \end{bmatrix}. \tag{3.39}$$

$\tilde{\boldsymbol{P}}^{-1}(t)\tilde{\boldsymbol{P}}(t) = \mathbf{I}$ where $\mathbf{I}$ is identity matrix. Then

$$\vec{\omega}_j^{\mathrm{T}}(t)\vec{u}_k(t) = \delta_{jk} \tag{3.40}$$

are the entries of matrix $\mathbf{I}$ which satisfy $\delta_{kk} = 1$ and $\delta_{jk} = 0\ \forall j \neq k$.

The eigenvector corresponding to a specific eigenvalue λ_k can be chosen arbitrarily from a space of vectors with the same direction

$$\vec{v}_k(t) = s_k(t)\vec{u}_k(t) \tag{3.41}$$

because any of them satisfies the relation

$$\boldsymbol{A}_0(t)s_k(t)\vec{u}_k(t) = \lambda_k(t)s_k(t)\vec{u}_k(t), \tag{3.42a}$$

$$\boldsymbol{A}_0(t)\vec{v}_k(t) = \lambda_k(t)\vec{v}_k(t). \tag{3.42b}$$

Using the normalisation factor $s_k(t)$ we can also scale the dual basis as $\vec{w}_k^{\mathrm{T}}(t) = s_k^{-1}(t)\vec{\omega}_k^{\mathrm{T}}(t)$.

It is convenient to choose the eigenvectors such that they satisfy the condition of so-called *dynamical orthogonality*. This means that the derivative of the eigenvectors is orthogonal to the corresponding adjoint vector as

$$\vec{w}_k^{\mathrm{T}} \frac{\mathrm{d}\vec{v}_k}{\mathrm{d}t} = \xi_{kk} = 0. \tag{3.43}$$

Using the chain rule for the derivation of $\vec{v}_k$ we find

$$0 = \vec{w}_k^{\mathrm{T}}(t)\frac{\mathrm{d}\vec{v}_k}{\mathrm{d}t} = \frac{s_k(t)}{s_k(t)}\vec{\omega}_k^{\mathrm{T}}(t)\frac{\mathrm{d}\vec{u}_k}{\mathrm{d}t} + \frac{\delta_{kk}}{s_k(t)}\frac{\mathrm{d}s_k}{\mathrm{d}t} = \vec{\omega}_k^{\mathrm{T}}(t)\frac{\mathrm{d}\vec{u}_k}{\mathrm{d}t} + s_k^{-1}(t)\frac{\mathrm{d}s_k}{\mathrm{d}t}. \tag{3.44}$$

This can be satisfied if we can find $s_k(t) \neq 0$ which is defined and differentiable in the interval $t \in I = [0, \infty)$

$$\int \frac{1}{s_k(t)}\mathrm{d}s_k(t) = -\int \vec{\omega}_k^{\mathrm{T}}(t)\mathrm{d}\vec{u}_k(t). \tag{3.45}$$

Then $s_k(t)$ has solution

$$s_k(t) = e^{-\int \vec{\omega}_k^{\mathrm{T}}(t)\mathrm{d}\vec{u}_k(t)}. \tag{3.46}$$

Solution of Leading-Order Term – $\mathcal{O}(1)$

The equation (3.28a) gives the solution in $\mathcal{O}(\varepsilon^0)$ as

$$\vec{Y}_0(t,\tau) = \sum_k a_k(t)\vec{v}_k(t)e^{\lambda_k(t)\tau}. \tag{3.47}$$

For now we assume that $a_k(t)$ is an arbitrary constant, and $\lambda_k(t)$ and $\vec{v}_k(t)$ are the eigenvalues and eigenvectors of matrix $\boldsymbol{A}_0(t)$.

To find the solution in the slow time scale t we let it go to the limit in the slow time $\tau \to \infty$ to get

$$\lim_{\tau\to\infty} \vec{Y}_0(t,\tau) = a_0(t)\vec{v}_0(t). \tag{3.48}$$

We rewrite (3.28b) in a non-homogeneous form

$$\frac{\partial \vec{Y}_1}{\partial t} = \boldsymbol{A}_0(t)\vec{Y}_1(t,\tau) + \vec{F}(t,\tau), \tag{3.49}$$

where the term

$$\vec{F}(t,\tau) = \boldsymbol{A}_1(t)\vec{Y}_0(t,\tau) - \frac{\partial \vec{Y}_0}{\partial t} + \vec{H}(t). \tag{3.50}$$

Using the solution for $\vec{Y}_0$ given in equation (3.47) we expand

$$\begin{aligned}\vec{F}(t,\tau) =& \sum_k \left[\boldsymbol{A}_1(t)a_k(t)\vec{v}_k(t) - \frac{\mathrm{d}a_k}{\mathrm{d}t}\vec{v}_k(t) - a_k(t)\frac{\mathrm{d}\vec{v}_k}{\mathrm{d}t} - \right.\\ &\left. - a_k(t)\vec{v}_k(t)\frac{\mathrm{d}\lambda_k}{\mathrm{d}t}\tau\right] e^{\lambda_k(t)\tau} + \vec{H}(t).\end{aligned} \tag{3.51}$$

The solution of the system of equations (3.49) is obtained by using the method of diagonalisation. Multiplying the system from the left by $\boldsymbol{P}^{-1}$ gives

$$\frac{\partial \vec{Z}}{\partial t} = \boldsymbol{\Lambda}(t)\vec{Z}(t,\tau) + \vec{G}(t,\tau) \tag{3.52}$$

where $\boldsymbol{\Lambda}(t)$ is eigenvalue matrix as defined in (3.37) and

$$\vec{Z}(t,\tau) = \boldsymbol{P}^{-1}(t)\vec{Y}_1(t,\tau), \tag{3.53a}$$

$$\vec{G}(t,\tau) = \boldsymbol{P}^{-1}(t)\vec{F}(t,\tau), \tag{3.53b}$$

which can be written by elements as

$$z_j(t,\tau) = \vec{w}_j^{\mathrm{T}}(t)\vec{Y}_1(t,\tau), \tag{3.54a}$$

$$g_j(t,\tau) = \vec{w}_j^{\mathrm{T}}(t)\vec{F}(t,\tau). \tag{3.54b}$$

Substituting (3.51) into (3.54b) and using the orthogonality (3.40) and dynamical orthogonality (3.43) properties for eigenvectors, we obtain

$$\begin{aligned}g_j(t,\tau) =& \sum_k \left[\vec{w}_j^{\mathrm{T}}(t)\boldsymbol{A}_1(t)a_k(t)\vec{v}_k(t) - \xi_{jk}a_k(t)\right] e^{\lambda_k(t)\tau} - \\ & - \left[\frac{\mathrm{d}a_j}{\mathrm{d}t} + a_j(t)\frac{\mathrm{d}\lambda_j}{\mathrm{d}t}\tau\right] e^{\lambda_j(t)\tau} + h_j(t)\end{aligned} \tag{3.55}$$

where $h_j(t) = \vec{w}_j^{\mathrm{T}}(t)\vec{H}(t)$.

The system (3.52) is uncoupled, and each individual equation can be written in a form

$$\frac{\partial z_j}{\partial \tau} =\lambda_j(t)z_j(t,\tau) + g_j(t,\tau). \tag{3.56}$$

Using the integrating factor method we get the solution

$$z_j(t,\tau) =e^{\lambda_j(t)\tau}\int e^{-\lambda_j(t)\tau}g_j(t,\tau)\mathrm{d}\tau \tag{3.57}$$

that, using g_j from the equation (3.55), rewrites as

$$\begin{aligned} z_j(t,\tau) = e^{\lambda_j(t)\tau}\Bigg\{ &\left[\vec{w}_j^{\mathrm{T}}(t)\boldsymbol{A}_1(t)a_j(t)\vec{v}_j(t) - \frac{\mathrm{d}a_j}{\mathrm{d}t}\right]\int \mathrm{d}\tau - a_j(t)\frac{\mathrm{d}\lambda_j}{\mathrm{d}t}\int \tau\mathrm{d}\tau + \\ + h_j(t)\int e^{-\lambda_j(t)\tau}\mathrm{d}\tau + \sum_{k\neq j}&\left[\vec{w}_j^{\mathrm{T}}(t)\boldsymbol{A}_1(t)a_k(t)\vec{v}_k(t) - \xi_{jk}a_k(t)\right]\int e^{(\lambda_k(t)-\lambda_j(t))\tau}\mathrm{d}\tau\Bigg\}. \end{aligned} \tag{3.58}$$

For the integration we have to consider the structure of the eigenvalues of our system. That is, there are no multiple eigenvalues, and $j = 0$ corresponds to a zero eigenvalue. For zero eigenvalue $j = 0$ we get the result

$$\begin{aligned} z_0(t,\tau) = &\left[\vec{w}_0^{\mathrm{T}}(t)\boldsymbol{A}_1(t)a_0(t)\vec{v}_0(t) - \frac{\mathrm{d}a_0}{\mathrm{d}t} + h_0(t)\right]\tau + \\ &+\sum_{k>0}\left[\vec{w}_0^{\mathrm{T}}(t)\boldsymbol{A}_1(t)a_k(t)\vec{v}_k(t) - \xi_{0k}a_k(t)\right]\frac{e^{\lambda_k(t)\tau}}{\lambda_k(t)} + K_0(t), \end{aligned} \tag{3.59}$$

where $K_0(t)$ is an integration constant in τ that is found using given initial values.

As $\tau \to \infty$ the terms in the sum of equation (3.59) decay to 0 because of the assumption that all the eigenvalues are non-positive. Therefore, we can write (3.59) in a form

$$z_0(t,\tau) = \left[\vec{w}_0^{\mathrm{T}}(t)\boldsymbol{A}_1(t)a_0(t)\vec{v}_0(t) - \frac{\mathrm{d}a_0}{\mathrm{d}t} + h_0(t)\right]\tau + K_0(t). \tag{3.60}$$

We know that $\vec{X}(t)$ is bounded, so $\vec{Y}_0(t,\tau)$ is bounded as well, and hence the solution $z_j(t,\tau)$ must be bounded (i.e. $z_j(t,\tau) \neq \pm\infty, \forall \tau \in [0,\infty)$). This means that the coefficient of the first term containing τ must be zero, which yields the following differential equation

$$\frac{\mathrm{d}a_0}{\mathrm{d}t} =a_0(t)\vec{w}_0^{\mathrm{T}}(t)\boldsymbol{A}_1(t)\vec{v}_0(t) + \vec{w}_0^{\mathrm{T}}(t)\vec{H}(t). \tag{3.61}$$

This equation defines a point on invariant manifold of the system (3.28a). The relation to the original variables $\vec{Y}_0(t)$ and the initial conditions are calculated from

(3.48) at the limit $\tau \to \infty$ giving

$$\vec{Y_0} = \vec{v}_0(t)a_0(t), \tag{3.62}$$

which implies

$$a_0(t) = \vec{w}_0^{\mathrm{T}}(t)\vec{Y_0}(t). \tag{3.63}$$

3.1.4 First-Order Correction Term for General Systems

Definition of the System

In this section we describe the perturbation theory for autonomous systems. Although Markov chains are non-autonomous, they can be converted to autonomous as will be described in Section 3.1.5.

The dynamical behaviour of autonomous systems is described by a system of ODEs

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \vec{f}(\vec{u}) + \varepsilon\vec{h}(\vec{u}), \tag{3.64}$$

where $\vec{u}, \vec{f}(\vec{u}), \vec{h}(\vec{u}) \in \mathbb{R}^{\eta+1}$. We define $\vec{u}$ as

$$\vec{u} = \vec{U}(\vec{a}(t)) + \varepsilon\vec{v}(t), \tag{3.65}$$

where the state of the system is given by a combination of a solution on an invariant attracting manifold with coordinates $\vec{U}(\vec{a}(t)) \in \mathbb{R}^{\eta+1}$ and a small perturbation $\vec{v}(t) \in \mathbb{R}^{\eta+1}$, which is called first-order correction term. Vector $\vec{a}(t) \in \mathbb{R}^{m+1}, m < \eta + 1$, denotes the coordinates on the invariant manifold such that $\vec{f}(\vec{U}(\vec{a}(t))) = 0$.

Eigenvalues of the Jacobian Matrix

To analyse the system (3.64) we have to find the Jacobian matrix

$$\boldsymbol{F}(\vec{a}(t)) = \frac{\partial\vec{f}}{\partial\vec{u}(t)}.$$

The eigenvectors $\vec{V_i}(\vec{a})$ and corresponding adjoint vectors $\vec{W^T}(\vec{a})$ of the Jacobian $\boldsymbol{F}(\vec{a})$ are found according to the following relation

$$\boldsymbol{F}(\vec{a}(t))\vec{V_i}(\vec{a}) = \Lambda_i\vec{V_i}(\vec{a}), \tag{3.66a}$$

$$\vec{W^T}_i(\vec{a})\boldsymbol{F}(\vec{a}(t)) = \Lambda_i\vec{W^T}_i(\vec{a}), \tag{3.66b}$$

$$W_i^T V_i = \delta_{ji} \tag{3.66c}$$

where $\Lambda_k = 0$ for $k < m + 1$ and $\Lambda_j \neq 0$ for $j \geq m + 1$.

We require that $\vec{v}(t)$ is always orthogonal to $\vec{U}(\vec{a})$ at the point $\vec{a}(t)$, which means that

$$\vec{W^T}_k(\vec{a})\vec{v}(t) = 0 \tag{3.67}$$

for $k = 0, 1, \ldots, m$.

From the definition of $\vec{u}$ we find that $\partial\vec{U}(\vec{a}) = \partial\vec{u} - \varepsilon\partial\vec{v}(t)$ that gives $\partial\vec{U}(\vec{a}) \approx \partial\vec{u}$ as $\varepsilon \to 0$ which leads to

$$\frac{\partial\vec{f}(\vec{U}(\vec{a}))}{\partial a_k} = \frac{\partial\vec{f}(\vec{U}(\vec{a}))}{\partial\vec{U}(\vec{a})}\frac{\partial\vec{U}(\vec{a})}{\partial a_k} = \frac{\partial\vec{f}(\vec{U}(\vec{a}))}{\partial\vec{u}}\frac{\partial\vec{U}(\vec{a})}{\partial a_k} = \boldsymbol{F}(\vec{a})\frac{\partial\vec{U}(\vec{a})}{\partial a_k}. \tag{3.68}$$

We choose $\vec{V_k}$ such that

$$\vec{V_k} = \frac{\partial\vec{U}}{\partial a_k} \tag{3.69}$$

for $k = 0, \ldots, m$ are the eigenvectors corresponding to zero eigenvalues $\Lambda_k = 0$. The eigenvalues in this section are no longer for only the fast part of the system as in the previous section. However, as the conservation law is still satisfied, so we assume an existence of at least one zero eigenvalue. The eigenvalues are also assumed to be real and non-positive. The assumption of the multiplicity $m = 1$ for the eigenvalues, no longer applies here, however we assume, that the system has a complete set of eigenvectors.

In the subsection 3.1.3 we show a way to choose the left and right eigenvectors corresponding to the same eigenvalue, such that they satisfy the requirement of dynamical orthogonality.

The multiplication of the left and right eigenvalue matrix gives a Kronecker delta function $\vec{W^T}_i\vec{V_j} = \delta_{ij}$ as defined in (3.40), which is a constant and therefore its derivative is

$$0 = \frac{\partial\delta_{ij}}{\partial a_k} = \vec{W^T}_i\frac{\partial\vec{V_j}(\vec{a}(t))}{\partial a_k} + \frac{\partial\vec{W^T}_i}{\partial a_k}\vec{V_j}(\vec{a}(t)). \tag{3.70}$$

We define

$$K_{ijk} = -\vec{W^T}_i\frac{\partial\vec{V_j}(\vec{a}(t))}{\partial a_k} = \frac{\partial\vec{W^T}_i}{\partial a_k}\vec{V_j}(\vec{a}(t)). \tag{3.71}$$

When $K_{ijk} = 0$ the eigenvectors $\vec{W^T}_i$ and $\vec{V_j}$ are dynamically orthogonal with respect to a_k. This was already established for $j = k$ by a particular choice of eigenvectors, as described in (3.43). The variable K_{ijk} is used for $i = 0, \ldots, m$ and $j = m + 1, \ldots, \eta$ from different sets, while k is only defined for $k = 0, \ldots, m$. This was shown to be true for the cases we will consider in the next chapter, but we have not found a proof for the general case.

Expansion of the First-Order Correction Term

We expand the first-order correction term in the basis of eigenvectors as

$$\vec{v}(t) = \sum_{j=m+1}^{\eta} \vec{V}_j(\vec{a}(t))b_j(t), \tag{3.72}$$

where $\vec{V}_j(\vec{a}(t))$ are the eigenvectors of the Jacobian $\boldsymbol{F}(\vec{a}(t))$ and $b_j \in \mathbb{R}$.

We define a notation for the flow on the invariant manifold as

$$\frac{\mathrm{d}a_k}{\mathrm{d}t} = \varepsilon \mathcal{A}_k \tag{3.73}$$

for $k = 0, \dots, m$. This is substituted into expression (3.72) to find the derivative of the first-order correction term,

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \varepsilon \mathcal{A}_k b_j + \sum_{j=m+1}^{\eta} \vec{V}_j(\vec{a}(t))\frac{\mathrm{d}b_j}{\mathrm{d}t}. \tag{3.74}$$

Taylor Expansion in Multiple Variables

We recall the Taylor series expansion of function $\vec{f}(\vec{u})$ in multiple variables which is done around the point $\vec{U} = [U_0, \dots, U_\eta]$ as

$$\begin{aligned}
&\vec{f}(u_0, \dots, u_\eta) = \\
=&\sum_{i_1=0}^{\infty} \cdots \sum_{i_\eta=0}^{\infty} \left(\frac{\partial^{i_1+\ldots+i_\eta} \vec{f}(U_0, \dots, U_\eta)}{\partial u_0^{i_1} \dots \partial u_\eta^{i_\eta}} \right) \frac{(u_0 - U_0)^{i_1} \dots (u_\eta - U_\eta)^{i_\eta}}{i_1! \dots i_\eta!} = \\
=&\vec{f}(U_0, \dots, U_\eta) + \sum_{j=0}^{\eta} \frac{\partial \vec{f}(U_0, \dots, U_\eta)}{\partial u_j}(u_j - U_j) + \\
&+ \frac{1}{2!} \sum_{j=0}^{\eta} \sum_{k=0}^{\eta} \frac{\partial^2 \vec{f}(U_0, \dots, U_\eta)}{\partial u_j \partial u_k}(u_j - U_j)(u_k - U_k) + \\
&+ \frac{1}{3!} \sum_{j=0}^{\eta} \sum_{k=0}^{\eta} \sum_{l=0}^{\eta} \frac{\partial^3 \vec{f}(U_0, \dots, U_\eta)}{\partial u_j \partial u_k \partial u_l}(u_j - U_j)(u_k - U_k)(u_l - U_l) + \dots = \\
=&\vec{f^U} + \sum_{j=0}^{\eta} \vec{f^U}_j (u_j - U_j) + \sum_{j=0}^{\eta} \sum_{k=0}^{\eta} \vec{f^U}_{jk}(u_j - U_j)(u_k - U_k) + \\
&+ \sum_{j=0}^{\eta} \sum_{k=0}^{\eta} \sum_{l=0}^{\eta} \vec{f^U}_{jk\ell}(u_j - U_j)(u_k - U_k)(u_l - U_l) + \dots .
\end{aligned} \tag{3.75}$$

The Taylor coefficients of the function $\vec{f}(u_1, \dots, u_\eta)$ at point $\vec{U}$ are

$$\vec{f^U} = \vec{f}(U_0, \dots, U_\eta), \tag{3.76a}$$

$$\vec{f^U}_j = \frac{\partial \vec{f}(U_0,\dots,U_\eta)}{\partial u_j}, \tag{3.76b}$$

$$\vec{f^U}_{jk} = \frac{1}{2!}\frac{\partial^2 \vec{f}(U_0,\dots,U_\eta)}{\partial u_j \partial u_k}, \tag{3.76c}$$

$$\vec{f^U}_{jk\ell} = \frac{1}{3!}\frac{\partial^3 \vec{f}(U_0,\dots,U_\eta)}{\partial u_j \partial u_k \partial u_\ell}. \tag{3.76d}$$

Analogously we do Taylor expansion of function $\vec{h}(\vec{u})$ with the coefficients $\vec{h^U}, \vec{h^U}_j$ and $\vec{h^U}_{jk}$.

For our purposes we need only two derivatives of $\vec{f^U}$ and one derivative of $\vec{h}$. Hence, we assume that those functions have the required number of continuous derivatives.

Expanding the System around the Invariant Manifold

We substitute (3.65) into (3.64) and rewrite the left hand side as

$$\begin{aligned}\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} =& \frac{\mathrm{d}\vec{U}}{\mathrm{d}t} + \varepsilon\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \sum_{k=0}^{m}\frac{\partial \vec{U}}{\partial a_k}\frac{\mathrm{d}a_k}{\mathrm{d}t} + \varepsilon\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \\ =& \varepsilon\left(\sum_{k=0}^{m}\vec{V}_k(\vec{a}(t))\mathcal{A}_k + \frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right) = \\ =& \varepsilon\sum_{k=0}^{m}\vec{V}_k(\vec{a}(t))\mathcal{A}_k + \varepsilon\sum_{j=m+1}^{\eta}\vec{V}_j(\vec{a}(t))\frac{\mathrm{d}b_j}{\mathrm{d}t} + \varepsilon^2\sum_{j=m+1}^{\eta}\sum_{k=0}^{m}\frac{\partial \vec{V}_j}{\partial a_k}\mathcal{A}_k b_j,\end{aligned} \tag{3.77}$$

and right hand side by using Taylor expansion (3.75) as

$$\begin{aligned}&\vec{f}(\vec{u}) + \varepsilon\vec{h}(\vec{u}) = \vec{f}(\vec{U}(\vec{a}(t)) + \varepsilon\vec{v}(t)) + \varepsilon\vec{h}(\vec{U}(\vec{a}(t)) + \varepsilon\vec{v}(t)) = \\ =& \vec{f^U} + \varepsilon\sum_{j=0}^{\eta}\vec{f^U}_j v_j(t) + \varepsilon^2\sum_{j=0}^{\eta}\sum_{k=0}^{\eta}\vec{f^U}_{jk}v_j(t)v_k(t) + \varepsilon\vec{h^U} + \varepsilon^2\sum_{j=0}^{\eta}\vec{h^U}_j v_j(t) + \mathcal{O}(\varepsilon^3).\end{aligned} \tag{3.78}$$

Here, v_j denotes j-th coordinate of the correction vector $\vec{v}$, and the function $\vec{f}(\vec{U}(a(t)))$ on the invariant manifold is $\vec{f^U} = 0$.

Putting (3.77) and (3.78) together we obtain

$$\begin{aligned}&\sum_{k=0}^{m}\vec{V}_k(\vec{a}(t))\mathcal{A}_k + \sum_{j=m+1}^{\eta}\vec{V}_j(\vec{a}(t))\frac{\mathrm{d}b_j}{\mathrm{d}t} + \varepsilon\sum_{j=m+1}^{\eta}\sum_{k=0}^{m}\frac{\partial \vec{V}_j}{\partial a_k}\mathcal{A}_k b_j = \\ =& \sum_{j=0}^{\eta}\vec{f^U}_j v_j + \varepsilon\sum_{j=0}^{\eta}\sum_{k=0}^{\eta}\vec{f^U}_{jk}v_j(t)v_k(t) + \vec{h^U} + \varepsilon\sum_{j}^{\eta}\vec{h^U}_j v_j(t) + \mathcal{O}(\varepsilon^2).\end{aligned} \tag{3.79}$$

We are going to pre-multiply this system by an adjoint vector $\vec{W^T}_i$ to the eigenvector $\vec{V}_i$ of the Jacobian. For that we simplify the first term on the right hand side as

$$\vec{W^T}_i \sum_{j=0}^{\eta} \vec{f^U}_j v_j(t) = \vec{W^T}_i \sum_{j=0}^{\eta} \frac{\partial \vec{f}(\vec{U}(a))}{\partial u_j} v_j(t) = \vec{W^T}_i \boldsymbol{F}(\vec{a})\vec{v}(t) = \\ = \Lambda_i \vec{W^T}_i \vec{v}(t) = \Lambda_i \vec{W^T}_i \sum_{j=0}^{\eta} \vec{V}_j(\vec{a}(t)) b_j(t) = \Lambda_i b_i(t), \tag{3.80}$$

and define

$$H_i(t) = \vec{W^T}_i \vec{h^U}. \tag{3.81}$$

We do the multiplication of (3.79) to get

$$\frac{\mathrm{d}b_i}{\mathrm{d}t} = \Lambda_i b_i(t) + H_i - \mathcal{A}_i + \varepsilon \Bigg\{ \sum_{j=0}^{\eta} \vec{W^T}_i \vec{h^U}_j v_j(t) + \sum_{j=m+1}^{\eta} \sum_{k=0}^{\eta} \Big[K_{ijk} \mathcal{A}_k b_j(t) + \\ + \vec{W^T}_i \vec{f^U}_{jk} v_j(t) v_k(t) \Big] \Bigg\} + \mathcal{O}(\varepsilon^2), \tag{3.82}$$

where subscripts of v_k denote the entries of the correction vector (3.72), and K_{ijk} was defined in (3.71).

We write the system as

$$\frac{\mathrm{d}b_i}{\mathrm{d}t} = \Lambda_i b_i(t) + H_i - \mathcal{A}_i + \varepsilon \mathcal{F}_i(\vec{b}, \vec{\mathcal{A}}) + \mathcal{O}(\varepsilon^2), \tag{3.83}$$

with the new variable $\mathcal{F}_i(\vec{b}, \vec{\mathcal{A}})$ collecting the terms of order $\mathcal{O}(\varepsilon)$ as

$$\mathcal{F}_i(\vec{b}, \vec{\mathcal{A}}) = \sum_{j=0}^{\eta} \sum_{k=m+1}^{\eta} \vec{W^T}_i \vec{h^U}_j V_{kj}(\vec{a}) b_k(t) + \sum_{j=m+1}^{\eta} \sum_{k=0}^{m} K_{ijk} \mathcal{A}_k b_j(t) + \\ + \sum_{j,k=0}^{\eta} \sum_{l,q=m+1}^{\eta} \vec{W^T}_i \vec{f^U}_{jk} V_{lj}(\vec{a}) b_l(t) V_{qk}(\vec{a}) b_q(t). \tag{3.84}$$

Solution of the Reduced General System

We impose the condition of orthogonality (3.67) which says that $b_j = 0$ for $j = 0, \dots, m$. We also recall that the flow $\mathcal{A}_j$ is only defined for for $j = 0, 1, \dots m$. Then we split the system (3.64) into two sets for $i < m+1$ and $i \geq m+1$ as

$$\mathcal{A}_i = H_i + \varepsilon \mathcal{F}_i(\vec{b}, \vec{\mathcal{A}}) + \mathcal{O}(\varepsilon^2) \qquad \text{for } i = 0, 1, \dots, m, \text{ and} \tag{3.85a}$$

$$\frac{\mathrm{d}b_i}{\mathrm{d}t} = \Lambda_i b_i(t) + H_i + \mathcal{O}(\varepsilon) \qquad \text{for } i = m+1, m+2, \dots, \eta. \tag{3.85b}$$

To solve $\mathcal{A}_i$ up to order $\mathcal{O}(\varepsilon^2)$, it is sufficient to solve b_i only up to order $\mathcal{O}(\varepsilon)$. This is because the term b_i in equation (3.85a) is only present in terms which are already multiplied by the parameter ε.

To find b_i we use an integrating factor with a limit t and an arbitrary chosen limit r as

$$\exp\left(-\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right) \tag{3.86}$$

that multiplies the equation (3.85b) to get

$$b_i(t) = \int_{s_i}^t \exp\left(\int_\tau^t \Lambda_i(\xi)\mathrm{d}\xi\right) H_i(\tau)\mathrm{d}\tau + D_i(t) + \mathcal{O}(\varepsilon), \tag{3.87}$$

where s_i is an arbitrary chosen limit (which will vanish in later calculations),

$$D_i(t) = C_i \exp\left(\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right) \tag{3.88}$$

and C_i is an arbitrary constant.

In order to obtain the result we will now expand the terms in the integrand of equation (3.87) to

$$\Lambda_i(\xi) = \Lambda_i(t^*) + \mathcal{O}(\varepsilon), \tag{3.89a}$$

$$H_i(\tau) = H_i(t^*) + \mathcal{O}(\varepsilon). \tag{3.89b}$$

This is substituted into (3.87) and after integration yields the following formula,

$$b_i(t) = -\frac{H_i(t^*)}{\Lambda_i(t^*)}\Big\{1 - \exp\left[\Lambda_i(t^*)(t - s_i)\right]\Big\} + D_i(t) + \mathcal{O}(\varepsilon). \tag{3.90}$$

The integrating factor in (3.88) is valid for any C. For convenience we choose

$$C = -\frac{H_i(t^*)}{\Lambda_i(t^*)} \tag{3.91}$$

such that the exponentials in (3.90) cancel out. Then the final equation for the coordinates across the invariant manifold has the form

$$b_i(t) = -\frac{H_i(t)}{\Lambda_i(t)} + \mathcal{O}(\varepsilon) = -\frac{\vec{W^T}_i \vec{h^U}(t)}{\Lambda_i(t)} + \mathcal{O}(\varepsilon). \tag{3.92}$$

To find the flow on the invariant manifold we substitute the result (3.92) into (3.85a), and using the $\mathrm{d}a_i/\mathrm{d}t = \varepsilon\mathcal{A}_i$ we obtain

$$\begin{aligned}
&\frac{\mathrm{d}a_i}{\mathrm{d}t} = \varepsilon \vec{W^T}_i \vec{h^U} + \\
&\varepsilon^2 \Bigg\{ -\sum_{j=0}^{\eta}\sum_{k=m+1}^{\eta} \vec{W^T}_i \vec{h^U}_j V_{kj}(\vec{a}) \frac{\vec{W^T}_k \vec{h^U}(t)}{\Lambda_k(t)} - \sum_{j=m+1}^{\eta}\sum_{k=0}^{m} K_{ijk} \vec{W^T}_k \vec{h^U}(t) \frac{\vec{W^T}_j \vec{h^U}(t)}{\Lambda_j(t)} + \\
&+ \sum_{j,k=0}^{\eta}\sum_{\ell,q=m+1}^{\eta} \vec{W^T}_i \vec{f^U}_{jk} V_{lj}(\vec{a}) \frac{\vec{W^T}_\ell \vec{h^U}(t)}{\Lambda_\ell(t)} V_{qk}(\vec{a}) \frac{\vec{W^T}_q \vec{h^U}(t)}{\Lambda_q(t)} \Bigg\} + \mathcal{O}(\varepsilon^3).
\end{aligned} \tag{3.93}$$

3.1.5 First-Order Correction Term for Markov Chains

Autonomisation of Markov Chains

Here we apply the dimensionality reduction using the first-order correction term for a Markov chain model, i.e. a system of linear ordinary differential equations. The Markov chain models of ion channels are normally non-autonomous as they depend explicitly on membrane voltage $\mathrm{V_m}(t)$ which is not part of the Markov chain model. As the theory was developed for autonomous system of ODEs, we autonomise the Markov chain using an additional variable σ as

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \left[\frac{1}{\varepsilon}\boldsymbol{A}_0(\sigma) + \boldsymbol{A}_1(\sigma)\right]\vec{x}, \tag{3.94a}$$

$$\frac{\mathrm{d}\sigma}{\mathrm{d}t} =1, \tag{3.94b}$$

with vectors $\vec{x}, \vec{H} \in \mathbb{R}^{\eta}$; matrices $\boldsymbol{A}_0(\sigma), \boldsymbol{A}_1(\sigma) \in \mathbb{R}^{\eta\times\eta}$; parameters $\sigma, \varepsilon \in \mathbb{R}$; $\varepsilon \to 0$ is a small number. The assumption of the number of smooth and continuous derivatives introduced earlier is automatically satisfied here, because the system is linear, as long as the entries of $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$ satisfy that requirement as well.

Time is re-scaled as analogously to (3.25) by introducing fast time $\tau = t/\varepsilon$.

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}\tau} =[\boldsymbol{A}_0(\sigma) + \varepsilon\boldsymbol{A}_1(\sigma)]\vec{x}, \tag{3.95a}$$

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\tau} =\varepsilon. \tag{3.95b}$$

We group the terms according to orders of ε. This allows us to rewrite the system in a form of (3.64), in which

$$\vec{f}(\vec{u}) = \begin{bmatrix}\boldsymbol{A}_0(\sigma)\vec{x}\\ 0\end{bmatrix}, \qquad \vec{h}(\vec{u}) = \begin{bmatrix}\boldsymbol{A}_1(\sigma)\vec{x}\\ 1\end{bmatrix}, \text{and} \qquad \vec{u} = \begin{bmatrix}\vec{x}\\ \sigma\end{bmatrix}. \tag{3.96}$$

The solution in the leading-order lies on the invariant m-dimensional manifold such that $\vec{f}(\vec{U}) = 0$. This condition is satisfied for a vector $\vec{U}$ which lies on the manifold. This vector is found as a linear combination of the vectors projected to a zero vector by matrix $\boldsymbol{A}_0(\sigma)$ (i.e. kernel of $\boldsymbol{A}_0$). We have proven the existence of an eigenvector from the state conservation law in equation (3.30) which corresponds to the zero eigenvalue $\lambda_j(\sigma) = 0$.

We have ordered those eigenvalues as $\lambda_k(\sigma) = 0$ for $k = 1, \ldots, m$, and $\lambda_j(\sigma) \neq 0$ for $j = m + 1, \ldots, \eta$, where m denotes the multiplicity of the zero eigenvalue of the matrix $\boldsymbol{A}_0$. The eigenvalue of the dimension liked with time has an index zero.

The solution can be written in a form equivalent to equation (3.47) that gives the transformation from the coordinates of the new system $\vec{a}$ to the coordinates of the original system $\vec{x}$ in the $\mathcal{O}(1)$ as

$$\vec{x}^{\mathcal{O}(1)} = \sum_{k=1}^{m} a_k \vec{v}_k(\sigma). \tag{3.97}$$

Then coordinates of the system on the invariant manifold (3.97) can be used to write down the coordinates for the extended phase space (including the time-like variable) as

$$\vec{U} = \begin{bmatrix} \sum_{j=1}^{m} a_j \vec{v}_j(\sigma) \\ \sigma \end{bmatrix}. \tag{3.98}$$

Including the first-order which is orthogonal to the invariant manifold, we get an expression for the original (non-extended) phase space as

$$\vec{x} = \sum_{k=1}^{m} a_k \vec{v}_k(\sigma) + \varepsilon \sum_{j=m+1}^{\eta} \vec{v}_j(\vec{a}(t)) b_j(t). \tag{3.99}$$

The Jacobian $\boldsymbol{F}(\vec{u})$ of the function $\vec{f}(\vec{u})$ from (3.96) evaluated at $\vec{u} = \vec{U}$ is

$$\boldsymbol{F}(\vec{u}) = \frac{\partial \vec{f}(\vec{u})}{\partial u} = \begin{bmatrix} \frac{\partial}{\partial x}(\boldsymbol{A}_0(\sigma)\vec{x}) & \frac{\partial}{\partial \sigma}(\boldsymbol{A}_0(\sigma)\vec{x}) \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_0(\sigma) & \frac{\partial \boldsymbol{A}_0(\sigma)}{\partial \sigma}\vec{x} \\ 0 & 0 \end{bmatrix}. \tag{3.100}$$

Eigenvalues and Eigenvectors

We find the eigenvalues and eigenvectors according to equation (3.66a). We look for eigenvectors in a form

$$\vec{V}_i = \begin{bmatrix} \vec{\kappa}_i \\ \mu_i \end{bmatrix}. \tag{3.101}$$

$\vec{\kappa} \in \mathbb{R}^{\eta}$ is the component of eigenvector $\vec{V}_i$ corresponding to the space $\vec{x}$, and $\mu \in \mathbb{R}$ corresponds to the extended space σ.

The eigenvalues are obtained solving the following system of equations,

$$\boldsymbol{A}_0(\sigma)\vec{\kappa}_i + \frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\vec{x}\mu_i = \Lambda_i \vec{\kappa}_i, \tag{3.102a}$$

$$0\vec{\kappa}_i + 0\mu_i = \Lambda_i \mu_i. \tag{3.102b}$$

If $\mu_i = 0$ then we are solving the eigenvalue problem for the matrix $\boldsymbol{A}_0(\sigma)$, which results in

$$\vec{V}_i = \begin{bmatrix} \vec{v}_i(\sigma) \\ 0 \end{bmatrix}, \qquad \Lambda_i = \lambda_i, \tag{3.103}$$

where $i = 1, \ldots, \eta$. This way we obtain η solutions. Because the system was extended by the variable σ, the dimension of the Jacobian and its eigenspace is η.

It is necessary to find one more linearly independent solution, which corresponds to the extended space when $\mu \neq 0$. Then equation (3.102b) implies $\Lambda_k = 0$ (we let the index for this solution to be $k = 0$). For $\Lambda_0 = 0$ equation (3.102a) gets the form

$$\boldsymbol{A}_0(\sigma)\vec{\kappa}_0 + \frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\vec{x}\mu_0 = 0, \tag{3.104}$$

and expanding $\vec{x}$ according to its definition (3.99) we get

$$\boldsymbol{A}_0(\sigma)\vec{\kappa}_0 + \mu_0\frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\sum_{k=1}^{m} a_k\vec{v}_k = 0. \tag{3.105}$$

We know that $\boldsymbol{A}_0(\sigma)\vec{v}_k(\sigma) = 0$ is satisfied because of the zero eigenvalues $\lambda_k = 0$ for $k = 1, \ldots, m$, therefore $\frac{\mathrm{d}}{\mathrm{d}\sigma}(\boldsymbol{A}_0(\sigma)\vec{v}_k) = 0$. Then, by differentiating it with respect to σ, we see that according to the product rule we obtain the following relation

$$\frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\vec{v}_k(\sigma) = -\boldsymbol{A}_0(\sigma)\frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma}. \tag{3.106}$$

Equation (3.105) then yields

$$\boldsymbol{A}_0(\sigma)\left(\vec{\kappa}_0 - \mu_0\sum_{k=1}^{m} a_k\frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma}\right) = 0, \tag{3.107}$$

we choose a specific vector that satisfies this relation as

$$\vec{\kappa}_0 = \mu_0\sum_{k=1}^{m} a_k\frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma}. \tag{3.108}$$

For convenience we choose $\mu_0 = 1$ in order to get the remaining eigenvector of $\boldsymbol{F}(\sigma)$ for eigenvalue $\Lambda_0 = 0$ as

$$\vec{V}_0 = \begin{bmatrix} \sum_{k=1}^{m} a_k\frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma} \\ 1 \end{bmatrix}. \tag{3.109}$$

The adjoint vectors are defined according to equation (3.66b) and satisfy the Kronecker delta function as $\vec{W^T}_j\vec{V}_i = \delta_{ji}$. Hence, we find the adjoint vectors for the eigenvectors (3.103) and (3.109) to be

$$\vec{W^T}_0 = \begin{bmatrix} 0, & 1 \end{bmatrix}, \tag{3.110a}$$

$$\vec{W^T}_i = \begin{bmatrix} \vec{w}_i, & -\sum_{j=1}^{m} a_j\vec{w}_i\frac{\mathrm{d}\vec{v}_j}{\mathrm{d}\sigma} \end{bmatrix}, \tag{3.110b}$$

where $i = 1, \dots, \eta$.

Term K_{ijk} defined by (3.71) is found by substituting the adjoint vectors and eigenvectors. The possible combinations are then $K_{0jk} = K_{0j0} = 0$ (because W_0^T is a constant vector), and

$$K_{ij0} = -\,\vec{W^T}_i \frac{\partial \vec{V_j}}{\partial a_0} = \frac{\partial \vec{W^T}_i}{\partial a_0} \vec{V_j}, \tag{3.111a}$$

$$K_{ijk} = -\,\vec{W^T}_i \frac{\partial \vec{V_j}}{\partial a_k} = \frac{\partial \vec{W^T}_i}{\partial a_k} \vec{V_j}, \tag{3.111b}$$

where $i = 1, \dots, m$, $j = m+1, \dots, \eta$ and $k = 1, \dots, m$.

Taylor Coefficients

The Taylor coefficients for the function $\vec{h}(\vec{u})$ are

$$\vec{h^U}_j = \frac{\partial \vec{h^U}}{\partial u_j} = \frac{\partial}{\partial u_j} \begin{bmatrix} \boldsymbol{A}_1(\sigma)\vec{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial u_j} \left(\boldsymbol{A}_1(\sigma)\vec{x}\right) \\ 0 \end{bmatrix}, \tag{3.112}$$

$$\vec{h^U}_{jk} = \frac{\partial^2 \vec{h^U}}{\partial u_j \partial u_k} = \frac{\partial^2}{\partial u_j \partial u_k} \begin{bmatrix} \boldsymbol{A}_1(\sigma)\vec{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\partial^2}{\partial u_j \partial u_k} \left(\boldsymbol{A}_1(\sigma)\vec{x}\right) \\ 0 \end{bmatrix}. \tag{3.113}$$

The Taylor coefficients for the function $\vec{f^U}(\vec{u})$ are

$$\vec{f^U}_j = \frac{\partial \vec{f^U}}{\partial u_j} = \begin{bmatrix} \frac{\partial}{\partial u_j} (\boldsymbol{A}_0(\sigma)\vec{x}) \\ 0 \end{bmatrix}, \tag{3.114a}$$

$$\vec{f^U}_{jk} = \frac{\partial^2 \vec{f^U}}{\partial u_j \partial u_k} = \begin{bmatrix} \frac{\partial^2}{\partial u_j \partial u_k} (\boldsymbol{A}_0(\sigma)\vec{x}) \\ 0 \end{bmatrix}, \tag{3.114b}$$

$$\vec{f^U}_{jk\ell} = \frac{\partial^3 \vec{f^U}}{\partial u_j \partial u_k \partial u_\ell} = \begin{bmatrix} \frac{\partial^2}{\partial u_j \partial u_k} (\boldsymbol{A}_0(\sigma)\vec{x}) \\ 0 \end{bmatrix}. \tag{3.114c}$$

Solution of the Reduced Markov Chain

Now we substitute into (3.93) for $i = 0$ to get

$$\begin{aligned} \frac{\mathrm{d}a_0}{\mathrm{d}t} &= \vec{W^T}_0 \vec{h^U} + \\ &+ \varepsilon \Bigg\{ -\sum_{j=0}^{\eta} \sum_{k=m+1}^{\eta} \vec{W^T}_0 \vec{h^U}_j V_{kj}(\vec{a}) \frac{\vec{W^T}_k \vec{h^U}(t)}{\Lambda_k(t)} - \sum_{j=m+1}^{\eta} \sum_{k=0}^{m} K_{0jk} \vec{W^T}_k \vec{h^U}(t) \frac{\vec{W^T}_j \vec{h^U}(t)}{\Lambda_j(t)} + \\ &+ \sum_{j,k=0}^{\eta} \sum_{\ell,q=m+1}^{\eta} \vec{W^T}_0 \vec{f^U}_{jk} V_{lj}(\vec{a}) \frac{\vec{W^T}_\ell \vec{h^U}(t)}{\Lambda_\ell(t)} V_{qk}(\vec{a}) \frac{\vec{W^T}_q \vec{h^U}(t)}{\Lambda_q(t)} \Bigg\} + \mathcal{O}(\varepsilon^2) = 1. \end{aligned} \tag{3.115}$$

Then we substitute the terms for $i \neq 0$ into (3.93). Using expression (3.99) (notice that $\mathcal{O}(\varepsilon)$ vanishes to higher order terms) this eventually gives

$$\begin{aligned}\frac{\mathrm{d}a_i}{\mathrm{d}t} =& L_i(\sigma) + \varepsilon \Bigg\{ -\sum_{j=0}^{\eta} \sum_{k=m+1}^{\eta} \vec{w}_i \left(\frac{\partial}{\partial u_j} \left(\boldsymbol{A}_1(\sigma)\vec{x} \right) \right) \frac{V_{kj}(\vec{a}) L_k(\sigma)}{\Lambda_k(t)} + \\ &+ \sum_{j,k=0}^{\eta} \sum_{\ell,q=m+1}^{\eta} \left[\vec{w}_i \frac{\partial^2}{\partial u_j \partial u_k} (\boldsymbol{A}_0(\sigma)\vec{x}) \right] \frac{V_{\ell j}(\vec{a}) L_\ell(\sigma)}{\Lambda_\ell(t)} \frac{V_{qk}(\vec{a}) L_q(\sigma)}{\Lambda_q(t)} \Bigg\} + \mathcal{O}(\varepsilon^2),\end{aligned} \tag{3.116}$$

where the variable for the leading-order term is

$$L_i(\sigma) = \vec{w}_i \left(\boldsymbol{A}_1(\sigma)\vec{x} \right) - \sum_{j=1}^{m} a_j \vec{w}_i \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}\sigma}. \tag{3.117}$$

We denote the j-th column of the matrices $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$ as $\vec{A}_0^j$ and $\vec{A}_1^j$. The derivative $\frac{\partial}{\partial u_j} \left(\boldsymbol{A}_1(\sigma)\vec{x} \right)$ is split for u_j when $j = 0, \ldots, \eta - 1$. In this case $u_j = x_j$. For $j = \eta$, $u_\eta = \sigma$. We obtain the relation

$$\frac{\partial}{\partial u_j} \left(\boldsymbol{A}_1(\sigma)\vec{x} \right) = \vec{A}_1^j(\sigma), \qquad \text{for } j = 0, \ldots, \eta - 1, \tag{3.118a}$$

$$\frac{\partial}{\partial u_\eta} \left(\boldsymbol{A}_1(\sigma)\vec{x} \right) = \frac{\partial A_1}{\partial \sigma} \vec{x}. \tag{3.118b}$$

Accordingly, we obtain the relation for the second derivatives as

$$\frac{\partial^2}{\partial u_j \partial u_k} \left(\boldsymbol{A}_0(\sigma)\vec{x} \right) = \frac{\partial \vec{A}_0^j}{\partial u_k} = 0, \qquad \text{for } j,k = 0, \ldots, \eta - 1, \tag{3.119a}$$

$$\frac{\partial^2}{\partial u_j \partial u_\eta} \left(\boldsymbol{A}_0(\sigma)\vec{x} \right) = \frac{\partial}{\partial u_j} \left(\frac{\partial A_0}{\partial \sigma} \vec{x} \right) = \frac{\partial \vec{A}_0^j}{\partial \sigma}, \qquad \text{for } j = 0, \ldots, \eta - 1, \tag{3.119b}$$

$$\frac{\partial^2}{\partial {u_\eta}^2} \left(\boldsymbol{A}_0(\sigma)\vec{x} \right) = \frac{\partial^2 A_0(\sigma)}{\partial \sigma^2} \vec{x}. \tag{3.119c}$$

We substitute the intermediate calculations into equation (3.116) to get

$$\begin{aligned}\frac{\mathrm{d}a_i}{\mathrm{d}t} =& L_i(\sigma) + \varepsilon \Bigg\{ -\sum_{j=0}^{\eta-1} \sum_{k=m+1}^{\eta} \vec{w}_i \vec{A}_1^j(\sigma) \frac{V_{kj}(\vec{a}) L_k(\sigma)}{\Lambda_k(t)} - \\ &- \sum_{k=m+1}^{\eta} \vec{w}_i \vec{A}_1^\eta(\sigma) \frac{V_{k\eta}(\vec{a}) L_k(\sigma)}{\Lambda_k(t)} - \\ &+ \sum_{\ell,q=m+1}^{\eta} \vec{w}_i \frac{\partial^2 A_0(\sigma)}{\partial \sigma^2} \vec{x} \frac{V_{\ell\eta}(\vec{a}) L_\ell(\sigma)}{\Lambda_\ell(t)} \frac{V_{q\eta}(\vec{a}) L_q(\sigma)}{\Lambda_q(t)} + \\ &+ \sum_{k=0}^{\eta-1} \sum_{\ell,q=m+1}^{\eta} \vec{w}_i \frac{\partial \vec{A}_0^k}{\partial \sigma} \frac{V_{\ell\eta}(\vec{a}) L_\ell(\sigma)}{\Lambda_\ell(t)} \frac{V_{qk}(\vec{a}) L_q(\sigma)}{\Lambda_q(t)} +\end{aligned}$$

$$+\sum_{j=0}^{\eta-1}\sum_{\ell,q=m+1}^{\eta}\vec{w}_i\frac{\partial\vec{A}_0^j}{\partial\sigma}\frac{V_{\ell j}(\vec{a})L_\ell(\sigma)}{\Lambda_\ell(t)}\frac{V_{q\eta}(\vec{a})L_q(\sigma)}{\Lambda_q(t)}\Bigg\}+\mathcal{O}(\varepsilon^2). \tag{3.120}$$

According to equation (3.103) that defines the value of the η-th entry of eigenvector, V_j is $V_{j\eta}=0$. The other entries of the eigenvectors correspond to the components of the eigenvectors of $\boldsymbol{A}_0$ (i.e. $V_{kj}=v_{kj}$ for $j=1,\dots,\eta-1$). Using those values we obtain

$$\frac{\mathrm{d}a_i}{\mathrm{d}t}=L_i(\sigma)-\varepsilon\sum_{j=0}^{\eta-1}\sum_{k=m+1}^{\eta}\vec{w}_i\vec{A}_1^j(\sigma)\frac{v_{kj}(\vec{a})L_k(\sigma)}{\lambda_k(t)}+\mathcal{O}(\varepsilon^2), \tag{3.121}$$

which substitutes into system (3.94). To convert the coordinates on the invariant manifold to the original coordinate system, we use equation (3.99).

3.2 Numerical Integration Methods

3.2.1 Order of Approximation

An analytic solution of a system is not always possible. In such cases we can approximate the solution by solving the system numerically. In this section we introduce methods used for the numerical approximation.

The numerical simulations imply an error which rises from the approximation of the solution. In this subsection we describe how to find the order of approximation of a numerical method.

A generic form of an ODE system is given as

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t}=\vec{f}(t,\vec{x}(t)), \tag{3.122}$$

where $\vec{f},\vec{x}\in\mathbb{R}^p$.

Using the Taylor series expansion we can represent the function in a vicinity of time point t_0 with initial conditions $\vec{x}(t_0)=\vec{x}_0$ as a sum of an infinite number of terms according to

$$\vec{f}(t,\vec{x})=\sum_{j=0}^{\infty}\frac{\vec{f}^{(j)}(t_0,\vec{x}_0)}{j!}\Delta t^j \tag{3.123}$$

where $\Delta t=t-t_0$ is the time step, and the coefficients are found as

$$\vec{f}^{(j)}(t_0,\vec{x}_0)=\left.\frac{\mathrm{d}^j\vec{f}}{\mathrm{d}t^j}\right|_{t=t_0}. \tag{3.124}$$

The consecutive terms of the series become smaller due to the division of an increasing factorial of j. The series can be written as

$$\vec{f}(t,\vec{x}) = \sum_{j=0}^{k-1} \frac{\vec{f}^{(j)}(t_0,\vec{x}_0)}{j!}\Delta t^j + \mathcal{O}(\Delta t^k) \tag{3.125}$$

where $\mathcal{O}(\Delta t^k)$ are terms of order k.

The Taylor expansion is a useful tool to estimate the accuracy of a numerical method. For this purpose we find a local truncation error by comparing equation (3.125) with the formula of the numerical method. Then the highest order of Δt used in the formula is called the local order of the method ($\mathcal{O}(\Delta t^k)$).

To estimate the global accuracy for a fixed time interval $I = [t_0, t_N]$, we realise that the number of steps to achieve the solution is $N = (t_N - t_0)/\Delta t$, i.e. inversely proportional of the time step. The order of global truncation error is then found as

$$\mathcal{O}(1/\Delta t)\mathcal{O}(\Delta t^k) = \mathcal{O}(\Delta t^{k-1}). \tag{3.126}$$

3.2.2 Explicit Methods – Forward Euler

The explicit methods are used to calculate the future state of the system from the state at current time. The simplest and most widely used explicit method is the forward Euler method. This method can be derived from the first-order approximation of the Taylor series of the system (3.122) which gives an iterative scheme

$$\vec{x}_{n+1} = \vec{x}_n + \vec{f}(t_n,\vec{x}_n)\Delta t, \qquad\qquad \vec{x}_0 = x_0(t_0). \tag{3.127}$$

The numerical solution $\vec{x}_{n+1} \approx \vec{x}(t_{n+1})$ converges to the exact solution as $\Delta t \to 0$, i.e. the smaller the time step is, the more accurate the result it provides. On the other hand, small time steps require calculating the state of the system more often. So, the computational demands of forward Euler method are inversly proportional to the selected time step size Δt.

Increasing the time step size allows to obtain the solution faster. However, the maximal time step is limited by so-called *numerical instability*. The numerical instability is an artefact observed as oscillations around the exact solution. Those oscillations often cause the simulation to crash.

3.2.3 Rush-Larsen Technique for a Gate Model

Dynamical equations describing the evolution of a gate model are often the most stiff parts of the cellular model, i.e. contain relatively fast processes. For that reason a small time step is required to conserve numerical stability.

The Rush-Larsen technique is an algorithm for an efficient solution of the gate model [17]. The gating variables are described by an equation in a form

$$\frac{\mathrm{d}w}{\mathrm{d}t} = \alpha(\mathrm{V_m}(t))(1-w) - \beta(\mathrm{V_m}(t))w \tag{3.128}$$

where α and β are transition rates, as in previously defined equations for (2.10) and (2.14b), and $w(t)$ is a gating variable. Assuming that the membrane voltage $\mathrm{V_m}$ does not change much during a short time step Δt, we can approximate the transition rates as

$$\alpha(\mathrm{V_m}(t)) \approx \alpha(\mathrm{V_m}(t_n)) = \alpha_n, \tag{3.129a}$$

$$\beta(\mathrm{V_m}(t)) \approx \beta(\mathrm{V_m}(t_n)) = \beta_n. \tag{3.129b}$$

Then the equation (3.128) gives a solution

$$w(t) = \frac{\alpha_n}{\alpha_n+\beta_n} - \left(\frac{\alpha_n}{\alpha_n+\beta_n} - w(t_n)\right)\exp\Big(-(t-t_n)(\alpha_n+\beta_n)\Big) \tag{3.130}$$

that is more convenient to write in a form

$$w_{n+1} = w_{\infty,n} - (w_{\infty,n} - w_n)\exp\left(-\frac{\Delta t}{\tau_{w,n}}\right) \tag{3.131}$$

where $\Delta t = t_{n+1} - t_n$ is the time step, the solution $w_n \approx w(t_n)$,

$$w_{\infty,n} = \frac{\alpha_n}{\alpha_n+\beta_n} \tag{3.132}$$

is a steady-state solution which is obtained from (3.128) by setting $\mathrm{d}w/\mathrm{d}t = 0$, and

$$\tau_{w,n} = \frac{1}{\alpha_n+\beta_n} \tag{3.133}$$

is a time constant.

The Rush-Larsen algorithm [17] adjusts the time step size according to the rate of change of the other variables (e.g. $\mathrm{V_m}$). The algorithm monitors the value of the slope of the potential $\mathrm{dV_m}/\mathrm{d}t$ and adapts the time step size (Δt). During the stimulus or if $|\mathrm{dV_m}/\mathrm{d}t| > 5$ mV/ms the $\Delta t = 0.01$ ms, otherwise the step $\Delta t = 0.01 \cdot [5/(\mathrm{dV_m}/\mathrm{d}t)]$.

Chapter 4

Dimensionality Reduction of I_{Na} Markov Chain

In this chapter, we are going to apply the asymptotic methods described in Section 3.1 of the previous chapter. These methods were developed for the reduction of dimensionality of Markov chains by elimination of fast processes. Our target for the reduction is a Markov chain model of sodium current I_{Na} which has been chosen for two main reasons. First, the I_{Na} is an important current from the electrophysiological point of view. It is responsible for the course of the cellular depolarisation, which initiates the cellular excitation called action potential. Second, the Markov chain definition of the channel contains fast processes. As a result, a numerical solution by forward Euler method requires relatively small values of time steps which leads to a high computational cost.

4.1 Analysis of I_{Na} Markov Chain

4.1.1 Formulation of I_{Na} Markov Chain

A Markov chain of I_{Na} channel published by Clancy and Rudy [2] was introduced in Section 2.3.1. For convenience we reformulate the notation of the states and transition rates. The new states are denoted by single letters from O to W. The name of the O state remains unchanged. The other states are denoted clock-wise by the letters in alphabetical order starting from the state O (Figure 4.1.1(b)). The transition rates are now consistently marked as α_{jk} for transition from state j to state k, e.g. α_{OP} denotes transition probability from state O to state P per unit of time.

The system of ordinary differential equations with the new notation is

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{UO}U - (\alpha_{OP} + \alpha_{OU})O, \tag{4.1a}$$
$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{UP}U + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PU} + \alpha_{PO})P, \tag{4.1b}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{4.1c}$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{4.1d}$$
$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, \tag{4.1e}$$
$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, \tag{4.1f}$$
$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - (\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV})U, \tag{4.1g}$$
$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{4.1h}$$
$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.1i}$$

The advantage of this type of notation is that we can readily check that

- Positive transition α_{jk} has a corresponding negative transition α_{kj} in the same equation,
- A certain transition appearing in one equation will also appear in another one with negative sign, which follows from the conservation law, and
- The transition rates (e.g. α_{jk}) are multiplied by a state corresponding to the first index (here state j).

The transition rates are redefined according to the new notation as

$$\alpha_{RQ} = \alpha_{11}, \qquad \alpha_{QR} = \beta_{11}, \tag{4.2a}$$
$$\alpha_{QP} = \alpha_{12}, \qquad \alpha_{PQ} = \beta_{12}, \tag{4.2b}$$
$$\alpha_{PO} = \alpha_{13}, \qquad \alpha_{OP} = \beta_{13}, \tag{4.2c}$$

$$\alpha_{ST} = \alpha_{11}, \qquad \alpha_{TS} = \beta_{11}, \tag{4.2d}$$
$$\alpha_{TU} = \alpha_{12}, \qquad \alpha_{UT} = \beta_{12}, \tag{4.2e}$$

$$\alpha_{UP} = \alpha_{3}, \qquad \alpha_{PU} = \beta_{3}, \tag{4.2f}$$
$$\alpha_{TQ} = \alpha_{3}, \qquad \alpha_{QT} = \beta_{3}, \tag{4.2g}$$
$$\alpha_{SR} = \alpha_{3}, \qquad \alpha_{RS} = \beta_{3}, \tag{4.2h}$$

$$\alpha_{OU} = \alpha_{2}, \qquad \alpha_{UO} = \beta_{2}, \tag{4.2i}$$

$$\alpha_{UV} = \alpha_4, \qquad \alpha_{VU} = \beta_4, \tag{4.2j}$$

$$\alpha_{VW} = \alpha_5, \qquad \alpha_{WV} = \beta_5. \tag{4.2k}$$

Notice that in the old notation some transition rates were present in more than one place on the diagram. Using the new notation each transition rate presents a specific transition between two states, but some of them are identical.

Figure 4.1.1(a) shows the dependence of the transition rates of the system on the membrane voltage. The transition rates can be divided according to their speeds, fast transition rates (top panel) and slow transition rates (bottom panel).

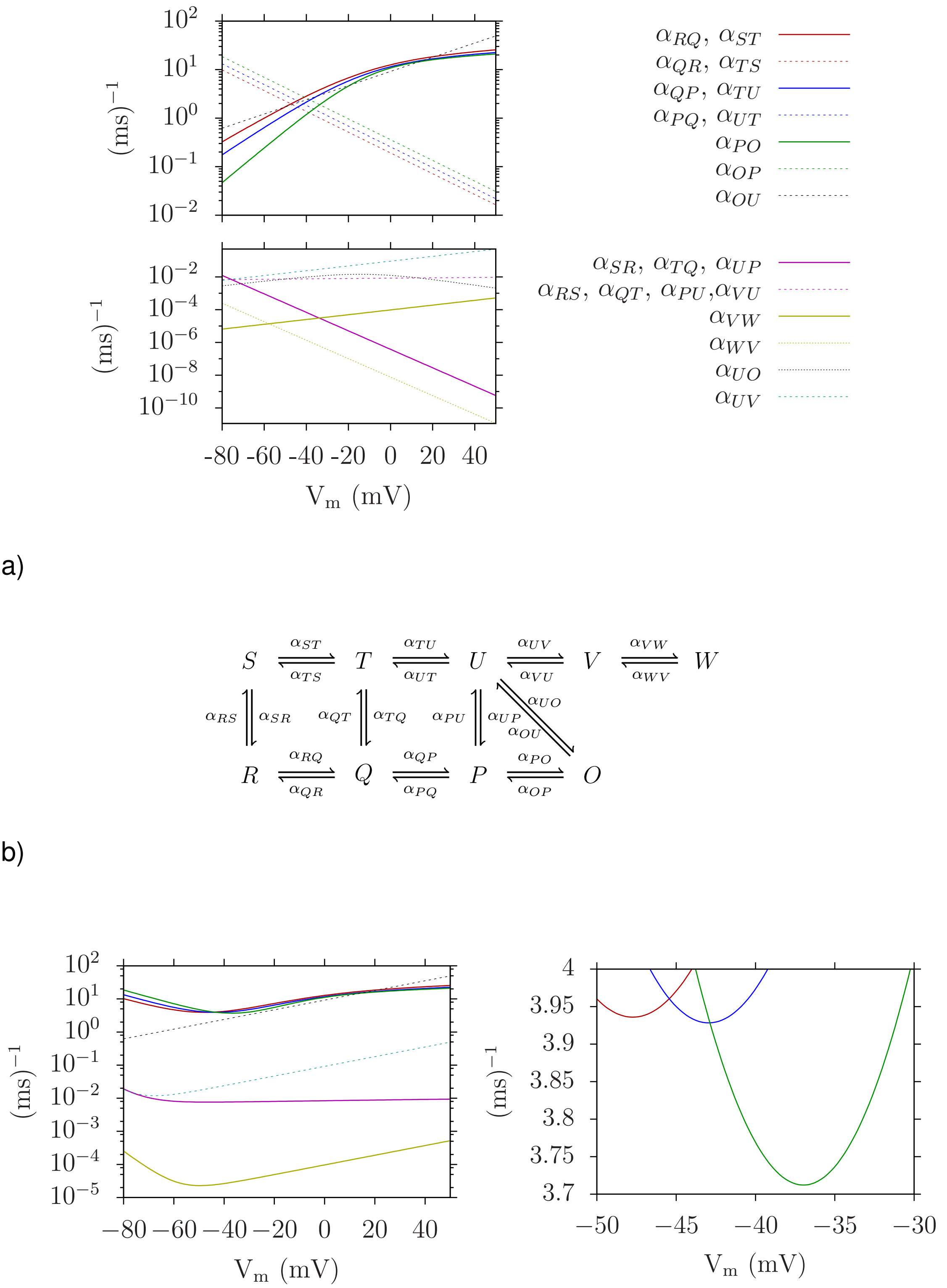
The sum of transition rates in both directions (e.g. $\alpha_{RQ} + \alpha_{QR}$) determines the probability of transitioning between two states, given that the state occupancy of both states is identical. The inverse of this sum is also called a time constant. The time constant can be used to determine how fast the system varies in time. Figure 4.1.1 shows the dependency of the speed of transition rates as a function of membrane voltage – panel (c) shows the dependency for the whole range of the membrane voltage, while panel (d) shows the fastest pairs of the transition rates in more detail at the minimum sum of their values. In our system the fastest transitions are found between states RQ, ST, QP, TU, and PO. Using the perturbation theory from the previous chapter, we will "speed-up" the transition rates in between those states. This procedure is also known as *transition rates embedding*. We will show the results of the system with transition rates embeddings in Section 4.1.1 and an example of the procedure of transition rates embedding in Section 4.2.1.

Figure 4.2 shows the eigenvalues and eigenvectors of the transition matrix at a fixed value of membrane voltage $\mathrm{V_m} = -30$ mV, which allows for analysing the dynamic properties of the system. We see that all the eigenvalues are non-positive with one zero eigenvalue, which agrees with our assumptions. The zero eigenvalue appears due to the conservation law that the Markov chain models satisfy.

4.1.2 Embeddings of I_{Na} Markov chain

Section 3.1.1 contains a demonstration of a dimensionality reduction on a simple toy model. In the example presented there, the system contains a small parameter ε, see equation (3.19). The small parameter divides the system into fast and slow time scale. Considering the case where $\varepsilon \to 0$ the dynamic behaviour of the system can be simplified. The fast variables decay almost instantaneously and we obtain a simpler system characterised by the slow variables. Such system then contains fewer dynamical variables.

The complication in Markov chain models is the lack of dependence on parameters, as they only contain transition rates determined experimentally. To apply the singular perturbation theory in such system we have to introduce the small



(c) (d)

Figure 4.1: Characteristics of I_{Na} Markov chain model: (a) individual transition rates for the range of membrane voltages (transitions between two states are denoted by the same colour); (b) a diagram of the model (new notation); (c,d) sum of pairs of transition rates between two states for the range of membrane voltages (as denoted in the legend on the right from the panels), (d) shows detail of the sum of the pairs in both directions as shown in panel (c).

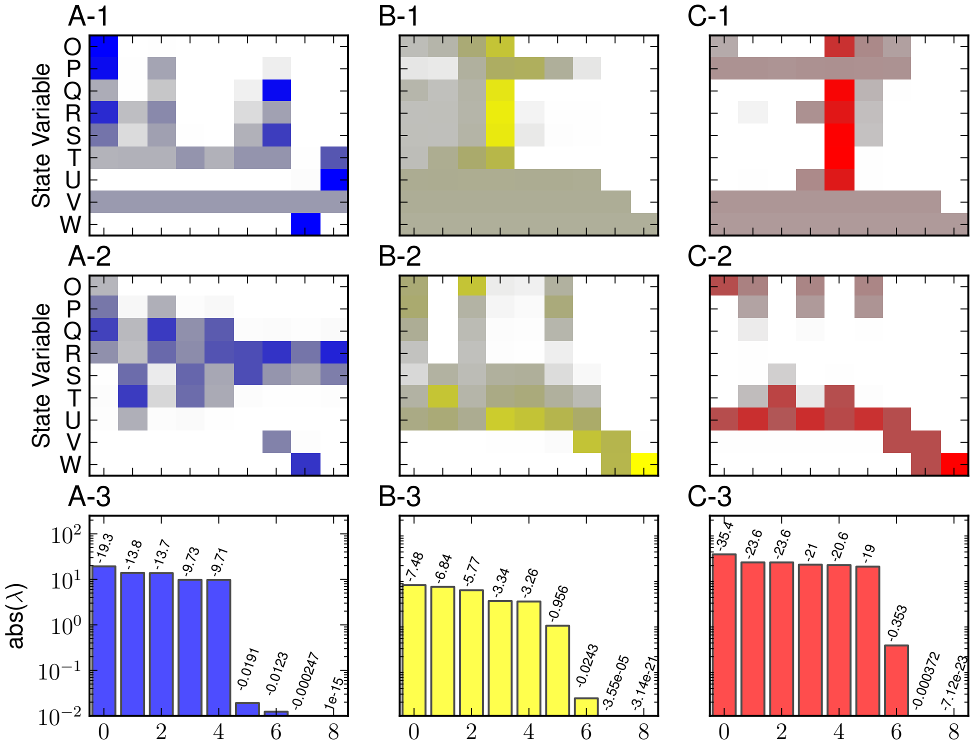


Figure 4.2: Eigenvalues and eigenvectors of I_{Na} model. First two rows show the entries in the matrices of left (1$^{\mathrm{st}}$ row) and right (2$^{\mathrm{nd}}$ row) eigenvectors, and the bottom row shows the absolute value of the eigenvalues (3$^{\mathrm{rd}}$ row) in the I_{Na} model under constant voltage of $\mathrm{V_m} = -80$ mV – blue (column A), $\mathrm{V_m} = -30$ mV – yellow (column B), and $\mathrm{V_m} = 40$ mV – red (column C). The brighter colour denotes higher absolute value of corresponding entry in the eigenvector matrix. The ordering of the eigenvalues corresponds to the ordering of to the eigenvectors. The number above each box corresponds to the eigenvalue.

parameters artificially. This procedure known as *parametric embedding* can be formalised as described in the following quotation from [19].

> *We will call a system*
>
> $$\frac{\mathrm{d}x}{\mathrm{d}t} = F(x, \varepsilon), \qquad\qquad x \in \mathbb{R}^d \tag{4.3}$$
>
> *a parametric embedding of*
>
> $$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x), \qquad\qquad x \in \mathbb{R}^d \tag{4.4}$$
>
> *if* $F(x, 1) = f(x)$ *for all* $x \in \mathbb{R}^d$*. . . .*
>
> *The typical use of this procedure has the form of a replacement of a small constant with a small parameter. If a system contains a dimensionless constant a which is "much smaller than 1", then replacement*

> *of a with εa constitutes a parametric embedding; and then the limit $\varepsilon \to 0$ can be considered. In practice, constant a would more often be replaced with parameter ε but in the context of εa and $a = \mathrm{const}$ this, of course, does not make any difference from εa.*

In the case of a Markov chain we do parametric embedding for a transition rates between two states i and j. If those transition rates fast, we can formally write that $\alpha_{ij} \to \infty, \alpha_{ji} \to \infty$, then the transition between those states happens almost instantaneously. To stress that those transition rates are big quantities, we can replace them by α_{ij}/ε and by α_{ji}/ε respectively. This way the model extends to a whole family of models a different parameter ε. In the subsequent text, we will call such system ij-embedded model as we always consider particular value of ε.

Because our interest concentrates on the asymptotic behaviour of the system, we can change the value of ε, in order to analyse the behaviour of the system after the initial transient. The embedding is “good”, if the quantitative features of the original model are approximated satisfactorily after the initial transient. Then the model reduced according to the singular perturbation theory will provide a good approximation to the original model.

Setting the parameter $\varepsilon \to 0$ means, that both embedded transition rates between the two states are speed-up at the same time. Our hypothesis is that we obtain a “good” embedding, if the transition rates are already fast. However, we cannot assume it to be always true when using complex models, such as Markov chains. This way we can test many different combinations of transition rates embeddings numerically, and save the time required for the manual derivation of the formulas for each possible reduction as the embeddings that fail to provide a satisfactory approximation are no longer considered for a reduction. Instead, we only do the reduction is for the embeddings that seem to lead to a good approximation. In other words, the transition rates embedding is an experimental tool that helps us to speed up the analysis of the system and identify the combination of transition rates, that are appropriate candidates for the reduction.

In general a Markov chain is described by a linear system of ODEs of a form

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{M}(t)\vec{u}, \tag{4.5}$$

using the parameter embedding procedure, we replace the matrix $\boldsymbol{M}$ with a matrix with transition rates embedding as

$$\boldsymbol{A}(t,\varepsilon) = \frac{1}{\varepsilon}\hat{\boldsymbol{A}}_0(t) + \boldsymbol{A}_1(t), \tag{4.6}$$

where the $\hat{\boldsymbol{A}}_0$ contains transition rates are selected for the embedding. The contribution from thematrix $\hat{\boldsymbol{A}}_0$ in the system then satisfies the conservation law.

For $\varepsilon = 1$ the system satisfies the relation $\boldsymbol{M}(t) = \boldsymbol{A}(t,1) = \boldsymbol{A}_0(t) + \boldsymbol{A}_1(t)$ which is consistent with the definition of embedding.

To ensure, that such embedding will result in a good approximation, we have to choose at least one pair of transition rates, that are fast. This is done after an analysis of the I_{Na} Markov chain (see Figure 4.1.1). The fast transition rates are likely to give a satisfactory embedding and hence are good candidate for the reduction. We verify such combinations of the transition rates embeddings by numerical simulations. For practical purposes we use $\varepsilon = 0.1$ which means that the corresponding transition rates are sped up by a factor of ten.

In the embeddings we have considered in this chapter the matrix $\hat{\boldsymbol{A}}_0$ for the full system is sparse. For convenience we reformulate the system such that all the embedded elements are in a dense part of the matrix, for instance

$$\hat{\boldsymbol{A}}_0 = \left[\begin{array}{c|c} \boldsymbol{A}_0 & 0 \\ \hline 0 & 0 \end{array}\right]. \tag{4.7}$$

This way we can consider only low dimensional problem of the matrix $\boldsymbol{A}_0$ for the diagonalisation. The remaining parts of the system (with zero entries) are trivial to solve, as the eigenvalues are all zeros and we choose the part of the eigenvector matrix corresponding to the trivial system as having entries equal to 1 on the diagonal.

As an example of an embedding, the transition rates between the states O and P are multiplied by $1/\varepsilon$, i.e. $\alpha_{PO}(t)$ and $\alpha_{OP}(t)$ in system (4.1). We write the OP-embedding as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{PO}(t)P + \alpha_{UO}(t)U - \left(\frac{1}{\varepsilon}\alpha_{OP}(t) + \alpha_{OU}(t)\right)O, \tag{4.8a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}(t)Q + \frac{1}{\varepsilon}\alpha_{OP}(t)O + \alpha_{UP}(t)U - \left(\frac{1}{\varepsilon}\alpha_{PO}(t) + \alpha_{PU}(t) + \alpha_{PQ}(t)\right)P. \tag{4.8b}$$

The rest of the system does not have any transition rates embedded and reads as

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{4.9a}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{4.9b}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, \tag{4.9c}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, \tag{4.9d}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - (\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV})U, \tag{4.9e}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{4.9f}$$

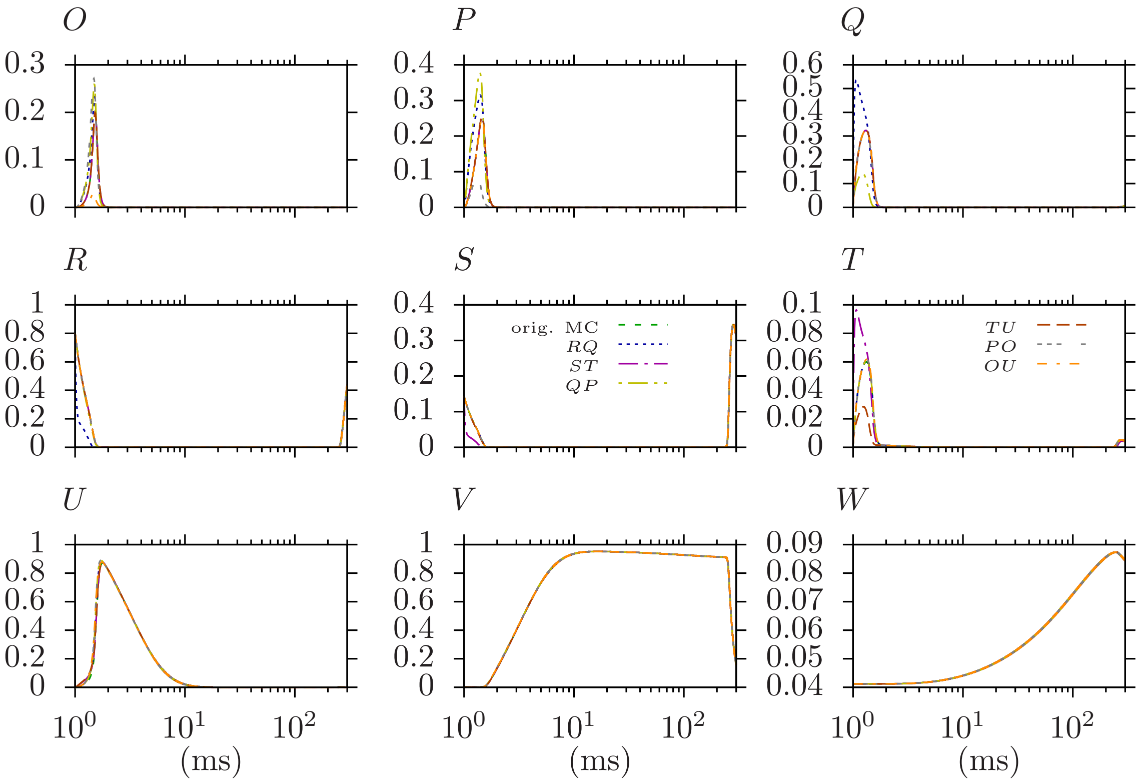


Figure 4.3: State occupancy of I_{Na} system with one pair of transition rates embedded with $\varepsilon = 0.1$ as described by (4.6). The state occupancy of states from O to W is shown from the top left to the bottom right the panel. The legend to the lines on all panels is shown on panels S and T. The horizontal axis denotes time (in milliseconds) shown in a logarithmic scale. The vertical axis denotes the state occupancy, i.e. the probability that the channel resides a particular state. The embedded I_{Na} models were driven by a simulated recording of action potential initiated at $t_0 = 1$. All the traces overlap after an initial transient of about 1 ms.

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.9g}$$

For brevity we call “embedding” also a system with embedded transition rates.

Analogously, we embed other combinations of transition rates. Figure 4.3 shows the states occupancy under an action potential. We recorded the action potential from model of the whole cell by Clancy, Rudy (2002). The recorded traces of action potential then drived the extracted I_{Na} model. The definition of the cellular model can be found in Appendix A.

We used embeddings of the transition rates with the highest sum in both directions (between states RQ, ST, QP, TU, PO, and OU) as shown in Figure 4.1.1. Although the solution of those embeddings provide a good approximation for the long term evolution of the system, the accuracy during the initial phase that corresponds to the action potential onset varies largely in different embeddings. Accurate simulation of the action potential onset is crucial because the I_{Na} current effectively controls the depolarisation of the cell, which causes the action potential.

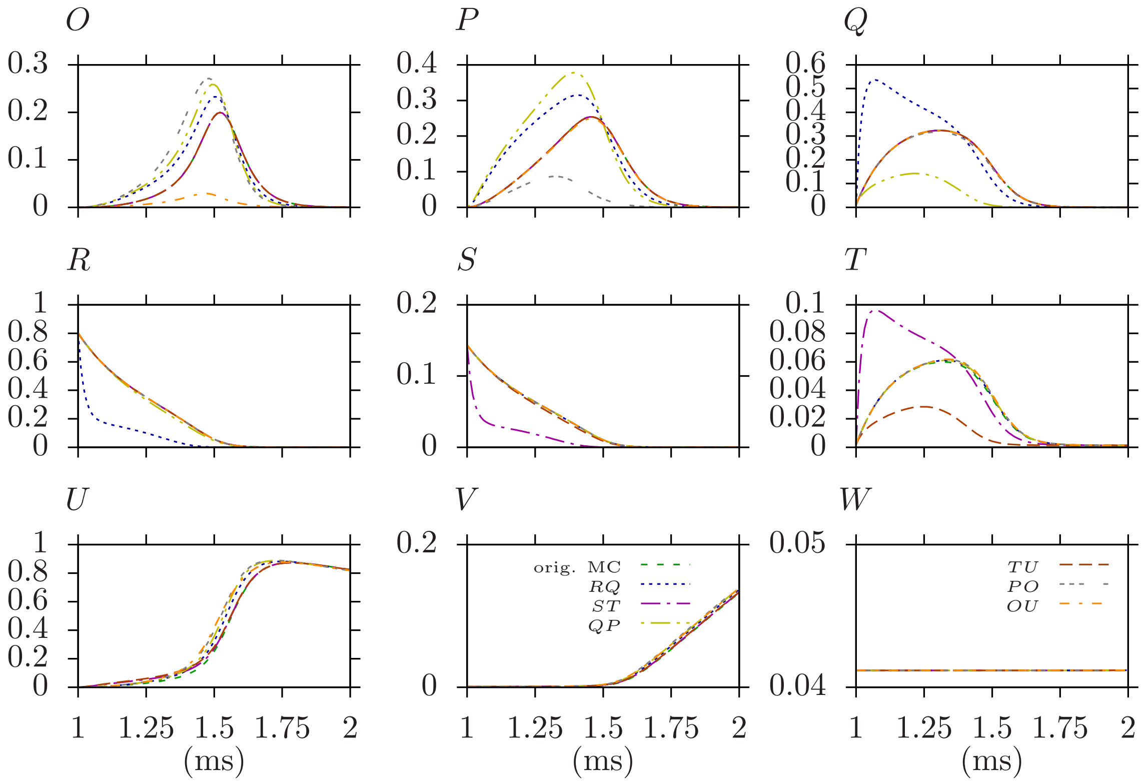


Figure 4.4: State occupancy of I_{Na} system with one pair of transition rates embedded with $\varepsilon = 0.1$ as described by (4.6). The figure shows the detail of the first 1 ms after the initiation of action potential. The legend to the lines on all panels is shown in panels V and W. The horizontal axis denotes time (in milliseconds) shown in a linear scale. The vertical axis denotes the state occupancy, i.e. the probability that the channel resides in a particular state. The embedded I_{Na} models were driven by a simulated recording of action potential initiated at $t_0 = 1$.

Figure 4.4 shows the first millisecond of the states occupancy under the action potential. The figure provides a detail of the part of the action potential which is affected by the transition rates embedding, so it is straightforward to compare the quality of the transition rates embeddings to each other.

In comparing the results for different embeddings, we have to focus mainly on the traces for the occupancy in the state O. This is because the model of I_{Na} ion channel is coupled with the remaining parts of the cellular model through the open probability given by the occupancy of state O. The occupancy of the rest of the states affects the remaining parts of the cellular model only indirectly via their transitions to the open state.

The detailed view shows that the solution of embeddings PO, RQ, QP and OU have an important deviation from the original model especially for the state O. The solution of embeddings TU and ST visually overlap with the occupancy of the state O. These embeddings affect some closed states, but this should not have a significant effect on the I_{Na}.

We perform double embeddings of the pairs of transition rates between two states ($ST + TU$, $RQ + QP$, $ST + RQ$, $TU + QP$) to find the next suitable reduction.

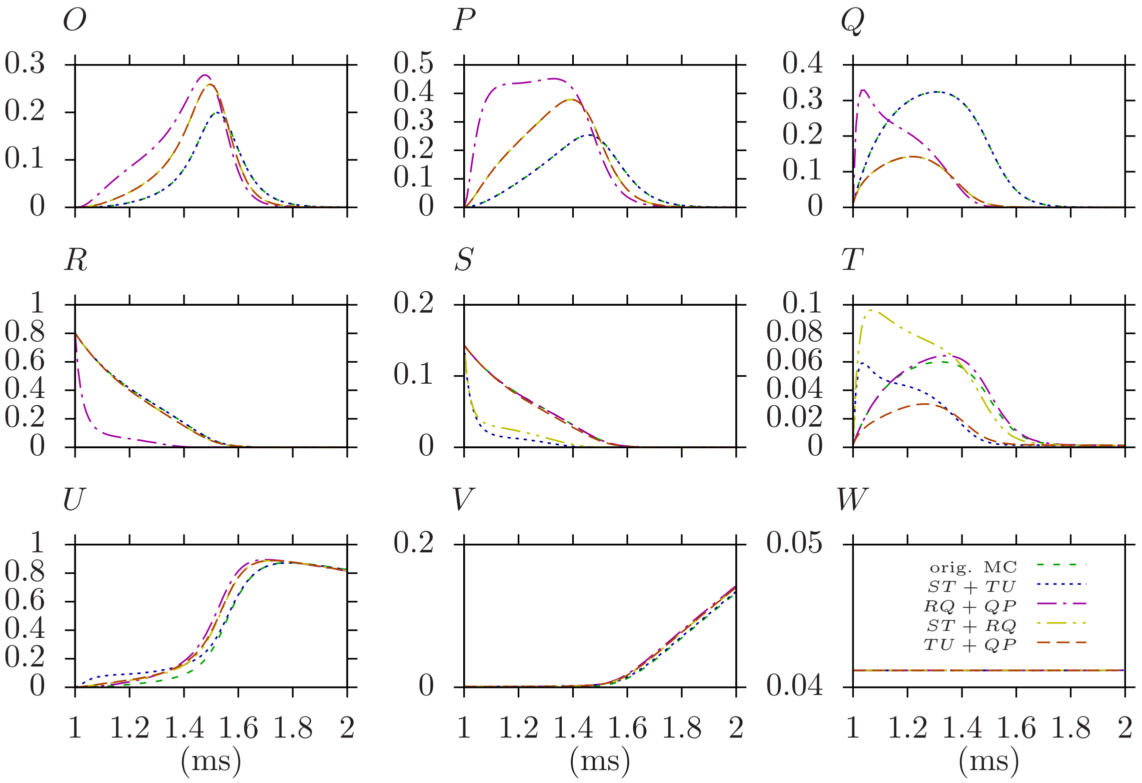


Figure 4.5: State occupancy of I_{Na} system with two pairs of transition rates embedded with $\varepsilon = 0.1$ as described by (4.6). The figure shows the detail of the first 1 ms after the initiation of action potential. The legend to the lines on all panels is shown in W. The horizontal axis denotes time (in milliseconds) shown in a linear scale. The vertical axis denotes the state occupancy, i.e. the probability that the channel resides a particular state. The embedded I_{Na} models were driven by a simulated recording of action potential initiated at $t_0 = 1$.

Figure 4.5 shows the first millisecond of the states occupancy under the action potential. The best approximation is given by the embedding $ST + TU$ which give a good results for most of the states, including state O, but not for the states T and U. The embeddings $ST + RQ$ and $TU + QP$ provide similar results in most states except the S and T states. However, both of them overshoot the open probability O. The embedding $RQ + QP$ provides the least accurate results of the of two-pairs of transition rates embeddings shown in the figure. It is also possible to perform other combinations of embeddings, but the ones in the figure are the most accurate.

To reduce the system further we do an embedding of more than two pairs of transition rates as shown in Figure 4.6. The best result is achieved by the reduction of transition rates between states $ST + TU + RQ$.

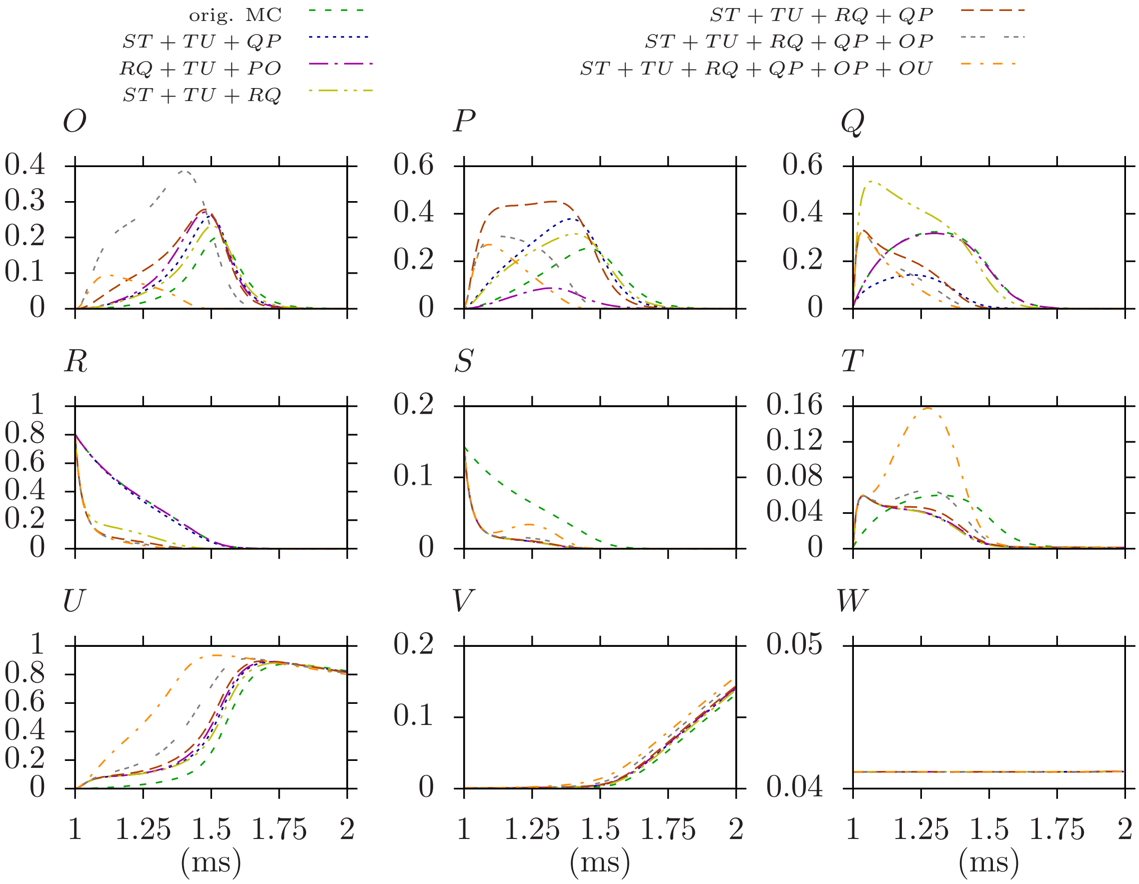


Figure 4.6: State occupancy of I_{Na} system with three or more pairs of transition rates embedded with $\varepsilon = 0.1$ as described by (4.6). The figure shows the detail of the first 1 ms after the initiation of action potential. The legend to the lines on all panels is shown above the figure. The horizontal axis denotes time (in milliseconds) shown in a linear scale. The vertical axis denotes the state occupancy, i.e. the probability that the channel resides a particular state. The embedded I_{Na} models were driven by a simulated recording of action potential initiated at $t_0 = 1$.

4.2 Leading-Order OP-Reduction in I_{Na} Markov Chain

4.2.1 Embedding of OP States

In this section we apply the perturbation theory to the I_{Na} channel as described by (4.1). According to (3.24) we write the parameters of the system (4.8) as

$$\vec{x}(t) = \begin{bmatrix} O(t) \\ P(t) \end{bmatrix}, \tag{4.10a}$$

$$\vec{H}(t) = \begin{bmatrix} \alpha_{UO}(t)U(t) \\ \alpha_{QP}(t)Q(t) + \alpha_{UP}(t)U(t) \end{bmatrix}, \tag{4.10b}$$

$$\boldsymbol{A}(t) = \frac{1}{\varepsilon}\left(\boldsymbol{A}_0(t) + \varepsilon \boldsymbol{A}_1(t)\right), \tag{4.10c}$$

$$\boldsymbol{A}_0(t) = \frac{1}{\varepsilon}\begin{bmatrix} -\alpha_{OP}(t) & \alpha_{PO}(t) \\ \alpha_{OP}(t) & -\alpha_{PO}(t) \end{bmatrix}, \tag{4.10d}$$

$$\boldsymbol{A}_1(t) = \begin{bmatrix} -\alpha_{OU}(t) & 0 \\ 0 & -(\alpha_{PU}(t) + \alpha_{PQ}(t)) \end{bmatrix}. \tag{4.10e}$$

Now we have all the ingredients to follow the procedure of dimensionality reduction according to Section 3.1.3.

4.2.2 Choice of Eigenvectors

To obtain the solution in form (3.61) we have to identify the eigenvalues of $\boldsymbol{A}_0(t)$ such that $\boldsymbol{A}_0(t)\vec{u}_k(t) = \lambda_k(t)\vec{u}_k(t)$. Let $k = 1$ be the index of the zero eigenvalue of the matrix $\boldsymbol{A}_0$. We also find the other required components and compile the full list as

$$\begin{aligned}
&\lambda_1(t) = 0, \\
&\lambda_2(t) = -(\alpha_{PO}(t) + \alpha_{OP}(t)) && \text{(4.11a)} \\
&\vec{u}_1(t) = \begin{bmatrix} \alpha_{PO}(t) \\ \alpha_{OP}(t) \end{bmatrix}, \\
&\vec{u}_2(t) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, && \text{(4.11b)} \\
&\vec{\omega}_1(t) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \\
&\vec{\omega}_2(t) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} -\alpha_{OP}(t) \\ \alpha_{PO}(t) \end{bmatrix}, && \text{(4.11c)} \\
&\tilde{\boldsymbol{P}}(t) = \begin{bmatrix} \alpha_{PO}(t) & -1 \\ \alpha_{OP}(t) & 1 \end{bmatrix}, \\
&\tilde{\boldsymbol{P}}^{-1}(t) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} 1 & 1 \\ -\alpha_{OP}(t) & \alpha_{PO}(t) \end{bmatrix}. && \text{(4.11d)}
\end{aligned}$$

The eigenvectors $\vec{u}_k(t)$ have to be normalised to satisfy the requirement of diagonal orthogonality. The normalisation coefficients were found according to (3.46) as

$$\begin{aligned}
s_1(t) &= \exp\left(-\int (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} 1 & 1 \end{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t} \left(\begin{bmatrix} \alpha_{PO}(t) \\ \alpha_{OP}(t) \end{bmatrix} \right) \mathrm{d}t \right) \\
&= \exp\left(-\ln(\alpha_{PO} + \alpha_{OP})\right) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1}, && \text{(4.12a)} \\
s_2(t) &= \exp\left(-\int (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} -\alpha_{OP}(t) & \alpha_{PO}(t) \end{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t} \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) \mathrm{d}t \right) = 1. && \text{(4.12b)}
\end{aligned}$$

Using the values of s_1 and s_2 and multiplying with the "trial eigenvectors" matrices (4.11d) we obtain the new eigenvectors matrix and its inverse as

$$\boldsymbol{P}(t) = \begin{bmatrix} \alpha_{PO}(t)(\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} & -1 \\ \alpha_{OP}(t)(\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} & 1 \end{bmatrix}, \tag{4.13a}$$

$$\boldsymbol{P}^{-1}(t) = \begin{bmatrix} 1 & 1 \\ -\alpha_{OP}(t)(\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} & \alpha_{PO}(t)(\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} \end{bmatrix}. \tag{4.13b}$$

So the new eigenvectors and their adjoint vectors are

$$\vec{v}_1(t) = \begin{bmatrix} \alpha_{PO}(t)(\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} \\ \alpha_{OP}(t)(\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} \end{bmatrix}, \tag{4.14a}$$

$$\vec{v}_2(t) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \tag{4.14b}$$

$$\vec{w}_1(t) = \begin{bmatrix} 1 & 1 \end{bmatrix}, \tag{4.14c}$$

$$\vec{w}_2(t) = (\alpha_{PO}(t)+\alpha_{OP}(t))^{-1} \begin{bmatrix} -\alpha_{OP}(t) & \alpha_{PO}(t) \end{bmatrix}, \tag{4.14d}$$

$$\lambda_1 = 0, \tag{4.14e}$$

$$\lambda_2(t) = -(\alpha_{PO}(t)+\alpha_{OP}(t)). \tag{4.14f}$$

We have now found the eigenvalues and eigenvectors and scaled them to satisfy the condition of dynamical orthogonality. In the following subsection we will describe the reduction of the states O and P.

4.2.3 Reduction of States O and P to One State N

We reformulate the general equations (3.61) and (3.63) specifically for our system. This leads to a system with fewer dynamical variables than in the original system. The original system contains the variables $\vec{x}$ and the reduced system variables $\vec{a}$ as follows

$$\vec{x}^T = [O, P, Q, R, S, T, U, V, W], \tag{4.15a}$$

$$\vec{a}^T = [N, Q, R, S, T, U, V, W]. \tag{4.15b}$$

So, the dynamical variable N embraces both old variables O and P in the original system. This means, that the new state N can be explicitly written as a superposition of two modes, one of which decays with time. For the reduction we use an assumption, that the decay is fast, hence the corresponding direction of the dynamical variable can be neglected. A simple example is considered in Section 3.1.1.

The remaining variables $(Q, \dots, W)$ remain in the reduced model, although their dynamics slightly change, as a result of the reduction.

Using a new state variable $N(t) = a_0(t)$ we get

$$\begin{aligned} \frac{\mathrm{d}N}{\mathrm{d}t} =& N(t)\vec{w}_1(t)\boldsymbol{A}_1(t)\vec{v}_1(t) + \vec{w}_1(t)\vec{H}(t), && \text{(4.16a)} \\ O(t) =& \frac{\alpha_{PO}(t)}{\alpha_{PO}(t)+\alpha_{OP}(t)} N(t), && \text{(4.16b)} \\ P(t) =& \frac{\alpha_{OP}(t)}{\alpha_{PO}(t)+\alpha_{OP}(t)} N(t), && \text{(4.16c)} \end{aligned}$$

where the new state includes the two old states as $N(t) = O(t) + P(t)$.

Having obtained normalised eigenvalues and eigenvectors we can expand both the non-homogeneous term and the coefficient of $N(t)$ in (4.16a) to

$$\begin{aligned} \vec{w}_1(t)\vec{H}(t) =& (\alpha_{UO}(t) + \alpha_{UP}(t))U(t) + \alpha_{QP}(t)Q(t), && \text{(4.17a)} \\ \vec{w}_1(t)\boldsymbol{A}_1(t)\vec{v}_1(t) =& \frac{\alpha_{OU}(t)\alpha_{PO}(t) + \alpha_{PQ}(t)\alpha_{OP}(t) + \alpha_{PU}(t)\alpha_{OP}(t)}{\alpha_{PO}(t)+\alpha_{OP}(t)}. && \text{(4.17b)} \end{aligned}$$

The differential equation for the new variable is

$$\frac{\mathrm{d}N}{\mathrm{d}t} = (\alpha_{UP} + \alpha_{UO})U + \alpha_{QP}Q - \left(\frac{\alpha_{OU}\alpha_{PO} + \alpha_{PU}\alpha_{OP}}{\alpha_{PO}+\alpha_{OP}} + \frac{\alpha_{PQ}\alpha_{OP}}{\alpha_{PO}+\alpha_{OP}}\right) N. \quad \text{(4.18)}$$

In the system (4.1) equations (4.1c) and (4.1g) are dependent on states $O(t)$ and $P(t)$ which can be reformulated using (4.16b) and (4.16c) as

$$\begin{aligned} \frac{\mathrm{d}Q}{\mathrm{d}t} =& \alpha_{RQ}R + \alpha_{TQ}T + \frac{\alpha_{PQ}\alpha_{OP}}{\alpha_{PO}+\alpha_{OP}} N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, && \text{(4.19a)} \\ \frac{\mathrm{d}U}{\mathrm{d}t} =& \alpha_{TU}T + \alpha_{VU}V + \frac{\alpha_{OU}\alpha_{PO} + \alpha_{PU}\alpha_{OP}}{\alpha_{PO}+\alpha_{OP}} N - (\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV})U. && \text{(4.19b)} \end{aligned}$$

We define new transition rates

$$\begin{aligned} \alpha_{UN} =& \alpha_{UP} + \alpha_{UO}, && \text{(4.20a)} \\ \alpha_{QN} =& \alpha_{QP}, && \text{(4.20b)} \\ \alpha_{NU} =& \frac{\alpha_{OU}\alpha_{PO} + \alpha_{PU}\alpha_{OP}}{\alpha_{PO}+\alpha_{OP}}, && \text{(4.20c)} \\ \alpha_{NQ} =& \frac{\alpha_{PQ}\alpha_{OP}}{\alpha_{PO}+\alpha_{OP}}, && \text{(4.20d)} \end{aligned}$$

which are substituted into equations (4.18) and (4.19) to give

$$\begin{aligned} \frac{\mathrm{d}N}{\mathrm{d}t} =& \alpha_{UN}U + \alpha_{QN}Q - (\alpha_{NU} + \alpha_{NQ})\, N, && \text{(4.21a)} \\ \frac{\mathrm{d}Q}{\mathrm{d}t} =& \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{NQ}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, && \text{(4.21b)} \end{aligned}$$

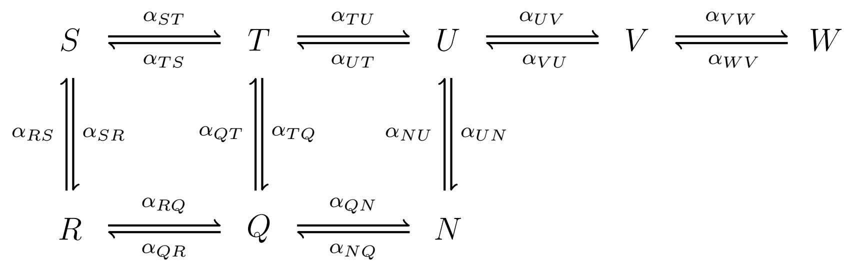


Figure 4.7: Diagram of the I_{Na} Markov chain with OP-reduction by state N. Compare with the original chain in Figure 4.1.1(b).

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{VU}V + \alpha_{NU}N - (\alpha_{UT} + \alpha_{UN} + \alpha_{UV})U. \tag{4.21c}$$

Equation (4.21a) substitutes both equations (4.1a) and (4.1b). Equations (4.21b) and (4.21c) replace (4.1c) and (4.1g) respectively. The reduced model is described by the following system of equations

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \alpha_{UN}U + \alpha_{QN}Q - (\alpha_{NU} + \alpha_{NQ})N, \tag{4.22a}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{NQ}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QN})Q, \tag{4.22b}$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{4.22c}$$
$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, \tag{4.22d}$$
$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, \tag{4.22e}$$
$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{NU}N + \alpha_{VU}V - (\alpha_{UT} + \alpha_{UN} + \alpha_{UV})U, \tag{4.22f}$$
$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{4.22g}$$
$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.22h}$$

The diagram of the model is shown in Figure 4.7.

Figure 4.8 shows the first 1 ms of simulated states occupancies of Markov chain I_{Na} driven by the membrane voltage, which is obtained from simulation of Clancy and Rudy (2002) model [2]. The reduced model shows the same qualitative behaviour as the corresponding embeddings. The largest difference between the original and reduced model is observed in the occupancy of the reduced states O and P. Although, the state U is directly connected with both of these states, only a minimal deviation is observed. This is because the transition rates of both O and P to U are slow. Moreover, the discrepancy from O is compensated by the opposite discrepancy caused by P.

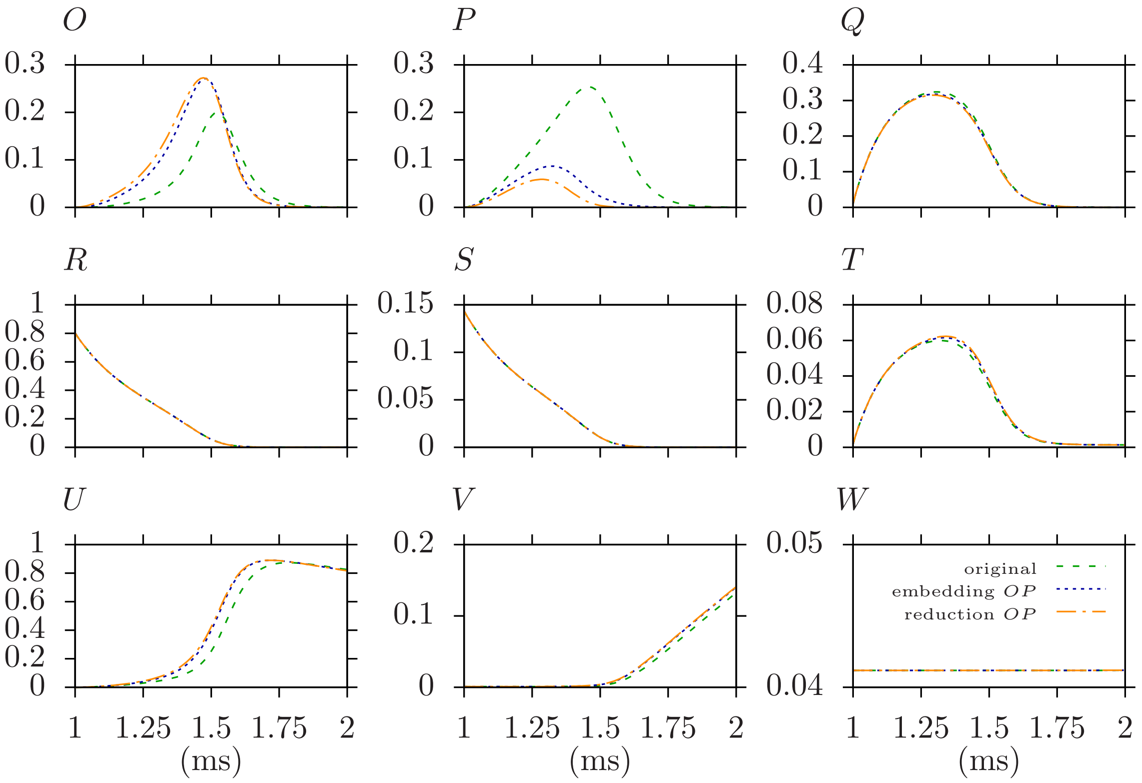


Figure 4.8: Comparison of state occupancy of $I_{\rm Na}$ in original model as described by equation (4.1) (green lines), and with OP-embedding (4.8), and where $\varepsilon = 0.1$ (blue lines) and OP-reduction (orange lines) system (4.2). The figure shows the detail of the first 1 ms after the initiation of action potential. The legend to the lines on all panels is shown in panel W. The horizontal axis denotes time (in milliseconds) shown in a linear scale. The vertical axis denotes the state occupancy, i.e. the probability that the channel resides a particular state. The $I_{\rm Na}$ models were driven by a simulated recording of action potential initiated at $t_0 = 1$.

4.2.4 Solution of OP-Reduction with a First-Order Term

The general equation for Markov chain model reduction (3.121) was developed in Section 3.1.5. By substituting the variables found for our system in equation (4.14) we can include the first-order term. The variables found in equation (4.14) are found for the non-trivial part of the system. These variables are sufficient to find the leading-order approximation. However, besides the leading-order we also need to include the first-order term. In general the first-order term for the other differential equations in the OP-reduction is

$$C_i^{\mathcal{O}(\varepsilon)} = -\left(\vec{w}_i \vec{A}_1^1(\sigma) v_{21}(\vec{a}) + \vec{w}_i \vec{A}_1^2(\sigma) v_{22}(\vec{a})\right) \frac{L_2(\sigma, t)}{\lambda_2(\sigma)}, \tag{4.23}$$

where the first and the second columns of the matrix are

$$\tilde{\boldsymbol{A}}_1 = \begin{bmatrix} -\alpha_{OU} & 0 & 0 & 0 & 0 & 0 & \alpha_{OU} & 0 & 0 \\ 0 & -(\alpha_{PU} + \alpha_{PQ}) & \alpha_{PQ} & 0 & 0 & 0 & \alpha_{PU} & 0 & 0 \end{bmatrix}^T. \tag{4.24}$$

As usual, the dynamical variables depend on the time t. We have introduced a new dynamical variable σ during the autonomisation (as described in section 3.1.5). The so-called "time-like" variable σ is linked with time, however it is now one of the coordinates (dynamical variables) on the slow manifold. Due to the autonomisation of the system, the parameters do not depend on time t directly, but are functions of time-like variable σ. In the following text, we omit the explicit dependence, i.e. notation for transition rates α implies $\alpha(\sigma)$ and for state variable e.g. N implies $N(t)$.

Then we find the first-order terms. The variables which contain a non-zero first-order are only for the states N, U and Q. Those read as

$$C_O^{\mathcal{O}(\varepsilon)} = \frac{L_2}{\alpha_{PO} + \alpha_{OP}}(\alpha_{OU} - \alpha_{PU} - \alpha_{PQ}), \tag{4.25a}$$

$$C_U^{\mathcal{O}(\varepsilon)} = \frac{L_2}{\alpha_{PO} + \alpha_{OP}}(\alpha_{PU} - \alpha_{OU}), \tag{4.25b}$$

$$C_Q^{\mathcal{O}(\varepsilon)} = \frac{L_2}{\alpha_{PO} + \alpha_{OP}}\alpha_{PQ}, \tag{4.25c}$$

where

$$\begin{aligned} L_2 =& \frac{\alpha_{OP}\alpha_{PO}(\alpha_{OU} - \alpha_{PU} - \alpha_{PQ})}{(\alpha_{PO} + \alpha_{OP})^2}N + \\ &+ \frac{\frac{\mathrm{d}\alpha_{PO}}{\mathrm{d}\sigma}\alpha_{OP} - \alpha_{PO}\frac{\mathrm{d}\alpha_{OP}}{\mathrm{d}\sigma}}{(\alpha_{PO} + \alpha_{OP})^2}N + \frac{\alpha_{PO}\alpha_{QP}}{(\alpha_{PO} + \alpha_{OP})}Q + \\ &+ \frac{\alpha_{PO}\alpha_{UP} - \alpha_{OP}\alpha_{UO}}{(\alpha_{PO} + \alpha_{OP})}U. \end{aligned} \tag{4.26}$$

We calculate $\frac{\mathrm{d}\alpha_{OP}}{\mathrm{d}\sigma}$ and $\frac{\mathrm{d}\alpha_{PO}}{\mathrm{d}\sigma}$ by the chain rule derivative as

$$\frac{\mathrm{d}\alpha_{OP}}{\mathrm{d}\sigma} = \frac{\partial\alpha_{OP}}{\partial \mathrm{V_m}}\frac{\mathrm{dV_m}}{\mathrm{d}\sigma}, \tag{4.27a}$$

$$\frac{\mathrm{d}\alpha_{PO}}{\mathrm{d}\sigma} = \frac{\partial\alpha_{PO}}{\partial \mathrm{V_m}}\frac{\mathrm{dV_m}}{\mathrm{d}\sigma}, \tag{4.27b}$$

where we use $\mathrm{dV_m}/\mathrm{d}\sigma$ which is assumed to be identical to $\mathrm{dV_m}/\mathrm{d}t$ as computed by the cellular model. According to the new notation (4.2c), the transition rates are $\alpha_{PO} = \alpha_{13}$ and $\alpha_{OP} = \beta_{13}$. Using the definition of α_{13} (2.48c) and β_{13} (2.48h) we find

$$\begin{aligned} \frac{\partial\alpha_{PO}}{\partial \mathrm{V_m}} =& \frac{\partial\alpha_{13}}{\partial \mathrm{V_m}} = \frac{\partial}{\partial \mathrm{V_m}}\left(\frac{3.802}{0.1027e^{-\mathrm{V_m}/12.0} + 0.25e^{-\mathrm{V_m}/150}}\right) = \\ =& \frac{3.802\left(0.1027/12.0e^{-\mathrm{V_m}/12.0} + 0.25/150e^{-\mathrm{V_m}/150}\right)}{\left(0.1027e^{-\mathrm{V_m}/12.0} + 0.25e^{-\mathrm{V_m}/150}\right)^2}, \end{aligned} \tag{4.28}$$

$$\frac{\partial\alpha_{OP}}{\partial \mathrm{V_m}} = \frac{\partial\beta_{13}}{\partial \mathrm{V_m}} = \frac{\partial}{\partial \mathrm{V_m}}\left(0.22e^{-(\mathrm{V_m}-10)/20.3}\right) = -\frac{2.2}{203}e^{-(\mathrm{V_m}-10)/20.3}. \tag{4.29}$$

The first-order terms are substituted into reduced model found in equations (4.22). The complete reduced model including the first-order terms is as follows (4.25)

$$\begin{aligned}
\frac{\mathrm{d}N}{\mathrm{d}t} =& \alpha_{UN}U + \alpha_{QN}Q - (\alpha_{NU} + \alpha_{NQ})N + \\
& + \varepsilon \frac{L_2}{\alpha_{PO} + \alpha_{OP}}(\alpha_{OU} - \alpha_{PU} - \alpha_{PQ}), && (4.30a)\\
\frac{\mathrm{d}Q}{\mathrm{d}t} =& \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{NQ}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QN})Q \\
& + \varepsilon \frac{L_2}{\alpha_{PO} + \alpha_{OP}}\alpha_{PQ}, && (4.30b)\\
\frac{\mathrm{d}R}{\mathrm{d}t} =& \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, && (4.30c)\\
\frac{\mathrm{d}S}{\mathrm{d}t} =& \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, && (4.30d)\\
\frac{\mathrm{d}T}{\mathrm{d}t} =& \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, && (4.30e)\\
\frac{\mathrm{d}U}{\mathrm{d}t} =& \alpha_{TU}T + \alpha_{NU}N + \alpha_{VU}V - (\alpha_{UT} + \alpha_{UN} + \alpha_{UV})U \\
& + \varepsilon \frac{L_2}{\alpha_{PO} + \alpha_{OP}}(\alpha_{PU} - \alpha_{OU}), && (4.30f)\\
\frac{\mathrm{d}V}{\mathrm{d}t} =& \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, && (4.30g)\\
\frac{\mathrm{d}W}{\mathrm{d}t} =& \alpha_{VW}V - \alpha_{WV}W, && (4.30h)\\
\frac{\mathrm{d}\sigma}{\mathrm{d}t} =& 1. && (4.30i)
\end{aligned}$$

The new variable N can be transformed to the original variables O and P. The transformation is found using the equation (3.65). In Section 4.2.3 we have found the equations (4.16b) and (4.16c) which correspond to the transformation onto the stable manifold (i.e. leading-order term $\vec{U}$ in (3.65)). This transformation can be made more accurate using a correction term corresponding to the coordinates orthogonal to the stable manifold $\vec{v}$. This term is a first order as $\mathcal{O}(\varepsilon)$ and is obtained using the equation (3.99). For our system we get

$$\vec{x} = N\vec{v}_1(\sigma) + \varepsilon\vec{v}_2(\vec{a}(t))b_2(t). \tag{4.31}$$

We substitute $b_2(t)$, that was identified in (3.92), by its equivalent definition using the (3.117) as $b_2 = L_2/\lambda_2$, where L_2 corresponds to (4.26), to get a corrected set of coordinates as

$$O = \frac{\alpha_{PO}}{\alpha_{PO} + \alpha_{OP}}N - \varepsilon\frac{L_2}{\alpha_{PO} + \alpha_{OP}}, \tag{4.32a}$$

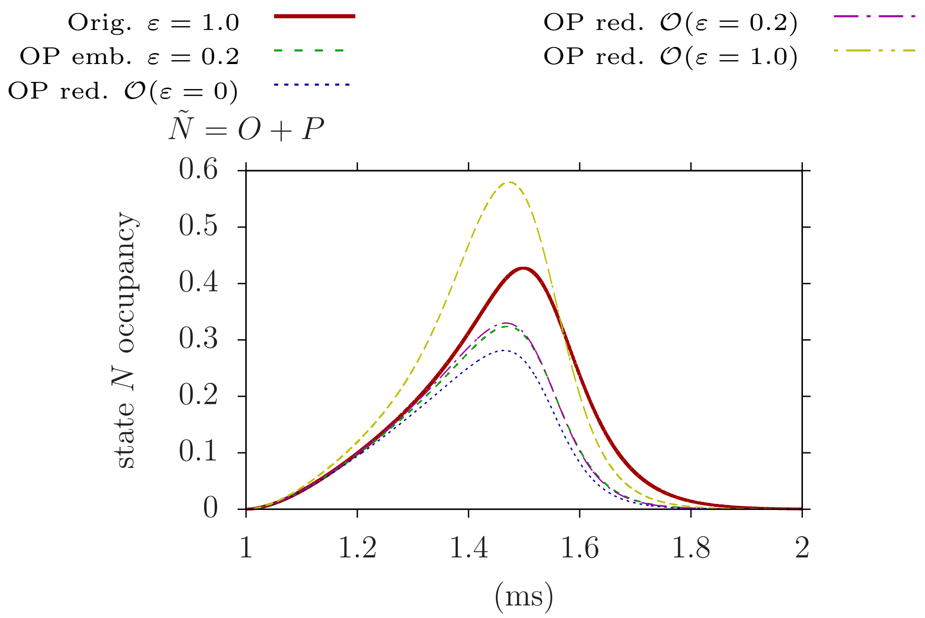


Figure 4.9: Occupancy of state N for the OP-reduction (OP red.) and OP-embedding (OP emb.) in leading-order and first-order of ε. The original model corresponds to embedded $\varepsilon = 1$ (red line). The OP-embedded model is shown also at values of $\varepsilon = 0.2$ (green line). The OP-reduced model with a first-order term dependent on ε is shown in a leading order $\varepsilon = 0$ (blue line), $\varepsilon = 0.2$ (magenta line) and $\varepsilon = 1$ (yellow-green line).

$$P = \frac{\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}} N + \varepsilon \frac{L_2}{\alpha_{PO} + \alpha_{OP}}. \tag{4.32b}$$

Figure 4.9 and Figure 4.11 show the simulation results of the I_{Na} model. The model was "driven" by membrane voltage $\mathrm{V_m}(t)$ from whole cell simulations of Clancy's model recorded every 0.01 ms. A linear interpolation was used to obtain corresponding value of membrane voltage at intermediate points in time. The size of the time step of the simulations was set to $\Delta t = 0.001$ ms. The figures shows the detail of the first 1 ms after the initiation of the action potential.

The simulation results of reduced and OP-embedded models are presented for corresponding variables. In case of the variable N, which is not present in the OP-embedded model, a formula $N = O + P$ was used. On the other hand, the variables for states O and P corresponding to the new state N in the reduced model were reconstructed from the variable N according to (4.32).

Figure 4.9 shows the simulation results for the a new state N. The original model corresponds to the OP-embedded model for $\varepsilon = 1$. The OP-reduced model was developed with a first-order correction term, which is dependent on ε. The leading-order and first-order approximation are identical for $\varepsilon = 0$. One can see that the results of the OP-embedded model closely approximates the results of OP-reduced model as $\varepsilon \to 0$ (see $\varepsilon = 0.2$ in the figure), which confirms that the derived formulas for a reduced model in the leading order are reasonable.

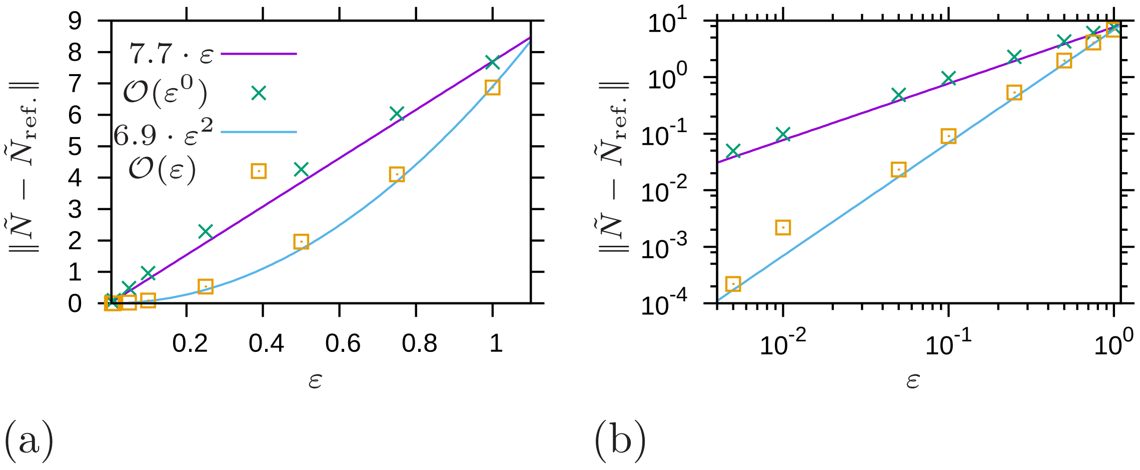


Figure 4.10: Order of approximation in ε for the leading-order reduced model (green crosses), first-order reduced model (yellow squares). The magenta line represents a function of $7.7 \cdot \varepsilon$ to show first-order error in ε, cyan line represents a function of $6.9 \cdot \varepsilon^2$ to show second-order dependence on ε. Panel (a) shows the results on a linear scale, panel (b) shows the same data on double logarithmic scale.

The results for the OP-reduced and OP-embedded model for of $\varepsilon = 0.2$ improve the approximation towards the results of the original model, as expected. However, as we increase to the value of $\varepsilon = 1$ in the reduced model, we observe much larger discrepancy with the original model towards the other direction.

Figure 4.10 and Table 4.2.4 depict how the first-order term in the OP-reduced model affects the accuracy of the results. Both the figure and the table show the same set of data. The simulations were done for the time step $\Delta t = 10^{-4}$ ms in all cases. The norm was computed according to the formula

$$\|N - N_{\text{ref.}}\| = \sqrt{\sum_{i=0}^{i_{\text{max.}}} (N(t_i, \varepsilon) - N_{\text{ref.}}(t_i, \varepsilon))^2} \tag{4.33}$$

where $i_{\text{max.}}$ corresponds to $t_{\text{max.}} = 2$ ms of time evolution. The $N_{\text{ref.}} = O + P$ represents the simulation using the OP-embedded model at values of ε as shown on the vertical axis.

The results are shown for the leading-order model $N(t, 0)$, i.e. fixed $\varepsilon = 0$ (green crosses) (4.30) which is equivalent to (4.22), and first-order model, i.e. variable ε (yellow squares) in (4.30), for ε as shown on vertical axis. The results are fitted to theoretical linear (magenta line) and quadratic functions (cyan line) to show the first-order and second-order error convergence. The formulas for the leading-order do not take into account terms of $\mathcal{O}(\varepsilon)$. The formulas in the first-order do not take into account terms above $\mathcal{O}(\varepsilon^2)$. This means that the formulas are zeroth-order and first-order accurate. As can be seen, the results confirm the expected error of convergence for each of those models.

Table 4.1: Norms $\|N - N_{\text{ref.}}\|$ as dependence on parameter ε for leading-order $\mathcal{O}(\varepsilon^0)$ and first-order $\mathcal{O}(\varepsilon)$ approximations.

ε	$\mathcal{O}(\varepsilon^0)$	$\mathcal{O}(\varepsilon^1)$
0.005	0.0495959	0.000219967
0.01	0.0976841	0.00219661
0.05	0.487114	0.0232822
0.1	0.959138	0.0902871
0.25	2.28563	0.53558
0.5	4.26603	1.96515
0.75	6.04319	4.10341
1.0	7.66945	6.86856

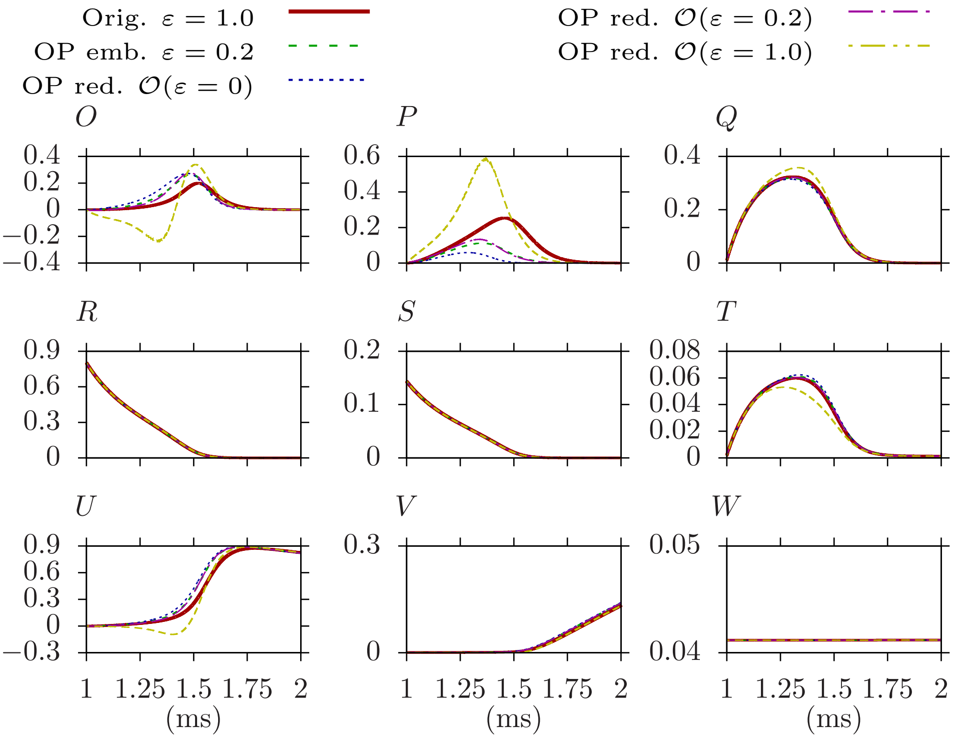


Figure 4.11: Occupancy of states in the I_{Na} model for the OP-reduction and OP-embedding in leading-order and first-order of ε. The legend and line colours are identical as in Figure 4.9. The states O and P in the reduced model that contains N variable instead were reconstructed using (4.32).

Figure 4.11 shows the time evolution states occupancy for all states and the simulations set-up was identical as for the data shown in Figure 4.9. The states O and P in the reduced model were reconstructed from the state N using equations (4.32) which include the first order correction term. The proximity of the OP-embedded model and the reconstruction of the states O and P suggests that the formulas for the reconstruction are reasonable.

The reconstruction including the first-order correction shows that in the state O the occupancy goes below zero. As the asymptotic series were built without reference to the physical restrictions, i.e. bounding the values of probabilities in the interval between (and including) 0 and 1, this result is not entirely unexpectable. However, we should emphasize, that the violation of those important physical constraints renders this model unusable for our purposes. Hence, we did not approach the derivation of higher-order formulas.

4.3 Leading-Order STU-Reduction in I_{Na} Markov Chain

4.3.1 Embedding of STU States

Figure 4.5 shows that the embedding of transition rates between the states S, T and U provides a good approximation for the open state occupancy O in the original system. In this section we aim to reduce the states S, T, and U by replacing them with a new state M using the theory from Section 3.1.3.

First, we define the embedded system of states S, T, and U analogously to Section 4.2. This means the transition rates between the states $T \leftrightarrow S$ and $S \leftrightarrow U$ in system of equations (4.1) should be multiplied by $1/\varepsilon$ (for $\varepsilon \to 0$). This yields the embedded system

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{UO}U - (\alpha_{OP} + \alpha_{OU})O, \tag{4.34a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{UP}U + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PU} + \alpha_{PO})P, \tag{4.34b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{4.34c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{4.34d}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{TS}T + \alpha_{RS}R - \left(\frac{1}{\varepsilon}\alpha_{ST} + \alpha_{SR}\right)S, \tag{4.34e}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \frac{1}{\varepsilon}\alpha_{ST}S + \frac{1}{\varepsilon}\alpha_{UT}U - \left(\alpha_{TQ} + \frac{1}{\varepsilon}\alpha_{TS} + \frac{1}{\varepsilon}\alpha_{TU}\right)T, \tag{4.34f}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{TU}T + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - \left(\frac{1}{\varepsilon}\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV}\right)U, \tag{4.34g}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{4.34h}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.34i}$$

For convenience we will only write the entries corresponding to states S, T and U. We put together the terms which contain the coefficient $1/\varepsilon$ as

$$\boldsymbol{A}_0 = \begin{bmatrix} -\alpha_{ST} & \alpha_{TS} & 0 \\ \alpha_{ST} & -(\alpha_{TS}+\alpha_{TU}) & \alpha_{UT} \\ 0 & \alpha_{TU} & -\alpha_{UT} \end{bmatrix} \tag{4.35}$$

and and the rest as

$$\boldsymbol{A}_1 = \begin{bmatrix} -\alpha_{SR} & 0 & 0 \\ 0 & -\alpha_{TQ} & 0 \\ 0 & 0 & -(\alpha_{UP}+\alpha_{UO}+\alpha_{UV}) \end{bmatrix}. \tag{4.36}$$

The non-homogeneous term and dynamical variables vector for this system are

$$\vec{x} = \begin{bmatrix} S \\ T \\ U \end{bmatrix}, \qquad \vec{H} = \begin{bmatrix} \alpha_{RS}R \\ \alpha_{QT}Q \\ \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O \end{bmatrix}. \tag{4.37}$$

Simlarly to the OP-reduction, we ignore the differential equations without transition rates embedding, as their solution is trivial.

4.3.2 Choice of Eigenvectors

First, we need to find eigenvalues and eigenvectors of matrix in (4.35). The roots of its characteristic equation, i.e. $\det(\boldsymbol{A}_0 - \lambda\mathbf{I})$ where $\mathbf{I}$ is the identity matrix, are found through

$$-\lambda^3 - \lambda^2(\alpha_{UT}+\alpha_{ST}+\alpha_{TU}+\alpha_{TS}) - \lambda(\alpha_{ST}\alpha_{UT}+\alpha_{TS}\alpha_{UT}+\alpha_{TU}\alpha_{ST}) = 0. \tag{4.38}$$

As expected one of the roots of the characteristic matrix is 0, which corresponds to the zero eigenvalue $\lambda_1 = 0$. We find that the eigenvector corresponding to this eigenvalue is

$$\vec{u}_1 = \begin{bmatrix} \alpha_{UT}\alpha_{TS} \\ \alpha_{UT}\alpha_{ST} \\ \alpha_{TU}\alpha_{ST} \end{bmatrix}. \tag{4.39}$$

The adjoint vector $\vec{\omega}_1$ of the eigenvector $\vec{u}_1$ is

$$\vec{\omega}_1 =(\alpha_{UT}\alpha_{TS}+\alpha_{UT}\alpha_{ST}+\alpha_{ST}\alpha_{TU})^{-1}\begin{bmatrix}1\\1\\1\end{bmatrix}. \tag{4.40}$$

The scaling factor (3.46) to satisfy the condition of dynamical orthogonality is

$$s_1=(\alpha_{UT}\alpha_{TS}+\alpha_{UT}\alpha_{ST}+\alpha_{ST}\alpha_{TU})^{-1}, \tag{4.41}$$

which gives us the required eigenvectors

$$\vec{v}_1=(\alpha_{UT}\alpha_{TS}+\alpha_{UT}\alpha_{ST}+\alpha_{ST}\alpha_{TU})^{-1}\begin{bmatrix}\alpha_{UT}\alpha_{TS}\\ \alpha_{UT}\alpha_{ST}\\ \alpha_{TU}\alpha_{ST}\end{bmatrix}, \qquad \vec{w}_1=\begin{bmatrix}1\\1\\1\end{bmatrix}. \tag{4.42}$$

The remaining eigenvalues are not required at this stage as long as they satisfy the requirements for the solution to be bounded, i.e. $\mathrm{Re}\{\lambda_{2,3}\}\leq 0$. To find eigenvalues $\lambda_{2,3}$ we find the remaining roots of the characteristic polynomial. We redefine (4.38) as $\lambda^2+\beta\lambda+\gamma=0$, where

$$\beta =\alpha_{UT}+\alpha_{ST}+\alpha_{TU}+\alpha_{TS}, \tag{4.43a}$$

$$\gamma =\alpha_{ST}\alpha_{UT}+\alpha_{TS}\alpha_{UT}+\alpha_{TU}\alpha_{ST}. \tag{4.43b}$$

Because the transition rates are always positive, the discriminant must be positive as well, as is shown in the following formula

$$\Delta=\beta^2-4\gamma=[(\alpha_{UT}-\alpha_{TS})+(\alpha_{TU}-\alpha_{ST})]^2+4\alpha_{TU}\alpha_{TS}\geq 0, \tag{4.44}$$

then $\lambda\in\mathbb{R}$. The eigenvalues can be found as

$$\lambda_{2,3}=\frac{-\beta\pm\sqrt{\Delta}}{2}. \tag{4.45}$$

For the solution to be bounded the eigenvalues have to be negative. We get the requirement $\sqrt{\beta^2-4\gamma}<\beta$ from (4.45). This is satisfied because the transition rates are always positive, and therefore $\beta,\gamma>0$.

4.3.3 Reduction of States S, T and U to One State M

We reduce the states S, T and U according to (3.61), where the eigenvalues are given by equation (4.45). The new state variable $a_0=M=S+T+U$ satisfies the following differential equation

$$\begin{aligned}\frac{\mathrm{d}M}{\mathrm{d}t} =&\alpha_{RS}R + \alpha_{QT}Q + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O- \\ &- \frac{\alpha_{SR}\alpha_{UT}\alpha_{TS} + \alpha_{TQ}\alpha_{UT}\alpha_{ST} + (\alpha_{UP} + \alpha_{OU} + \alpha_{UV})\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M. \end{aligned} \tag{4.46}$$

The relation between the new and old states occupancy is found from equation (3.63) as

$$S = \frac{\alpha_{UT}\alpha_{TS}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M, \tag{4.47a}$$

$$T = \frac{\alpha_{UT}\alpha_{ST}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M, \tag{4.47b}$$

$$U = \frac{\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M. \tag{4.47c}$$

We define new transition rates

$$\alpha_{MS} = \frac{\alpha_{UT}\alpha_{TS}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48a}$$

$$\alpha_{MT} = \frac{\alpha_{UT}\alpha_{ST}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48b}$$

$$\alpha_{MU} = \frac{\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48c}$$

$$\alpha_{RM} = \alpha_{RS}, \tag{4.48d}$$

$$\alpha_{QM} = \alpha_{QT}, \tag{4.48e}$$

$$\alpha_{PM} = \alpha_{PU}, \tag{4.48f}$$

$$\alpha_{VM} = \alpha_{VU}, \tag{4.48g}$$

$$\alpha_{OM} = \alpha_{OU}, \tag{4.48h}$$

$$\alpha_{MO} = \frac{\alpha_{UO}\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48i}$$

$$\alpha_{MP} = \frac{\alpha_{UP}\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48j}$$

$$\alpha_{MQ} = \frac{\alpha_{TQ}\alpha_{UT}\alpha_{ST}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48k}$$

$$\alpha_{MR} = \frac{\alpha_{SR}\alpha_{UT}\alpha_{TS}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48l}$$

$$\alpha_{MV} = \frac{\alpha_{UV}\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{4.48m}$$

which allow to reformulate the system of differential equations (4.34) as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{MO}M - (\alpha_{OP} + \alpha_{OM})O, \tag{4.49a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{MP}M + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PM} + \alpha_{PO})P, \tag{4.49b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{MQ}M + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QM} + \alpha_{QP})Q, \tag{4.49c}$$

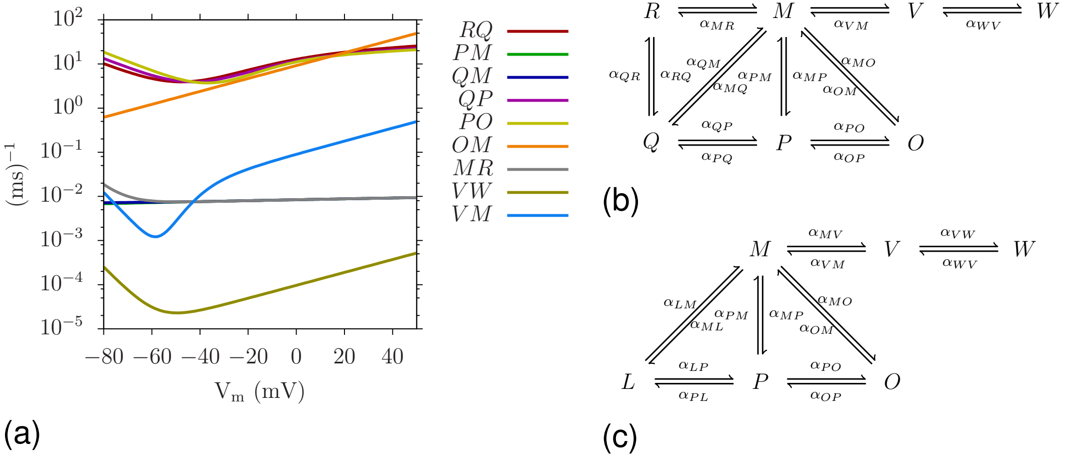


Figure 4.12: (a) Sum of the transition rates between two states in the STU-reduced model. The legend on the right of the panel denotes the corresponding states. (b) Diagram of the STU-reduction model as described by equation (4.49). (c) Diagram of RQ-reduction in the STU-reduced model as described by equation (4.57). The diagram of the original chain can be found in Figure 4.1.1(b).

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{MR}M + \alpha_{QR}Q - (\alpha_{RM} + \alpha_{RQ})R, \tag{4.49d}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{MV}M + \alpha_{WV}W - (\alpha_{VM} + \alpha_{VW})V, \tag{4.49e}$$

$$\begin{aligned}\frac{\mathrm{d}M}{\mathrm{d}t} =& \alpha_{OM}O + \alpha_{PM}P + \alpha_{QM}Q + \alpha_{RM}R + \alpha_{VM}V - \\ & - (\alpha_{MO} + \alpha_{MP} + \alpha_{MQ} + \alpha_{MR} + \alpha_{MV})M,\end{aligned} \tag{4.49f}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.49g}$$

Figure 4.12(b) shows the diagram of the structure of the STU-reduced Markov chain model. Figure 4.14 shows the detail of the first millisecond of the simulated traces of Markov chain I_{Na} channel driven by the recorded action potential from simulation in the original cellular model. The figure shows that the STU-reduction affects only the states S, T and U, which were reduced to one state. All the remaining states overlap. The state O in the reduced model is a good approximation of the original model.

4.4 Leading-Order RQ-Reduction in the STU-Reduced I_{Na} Model

4.4.1 Embedding of RQ States

The STU-reduced model shows a good approximation of the open state O which is needed to compute the conductance of the I_{Na} channel.

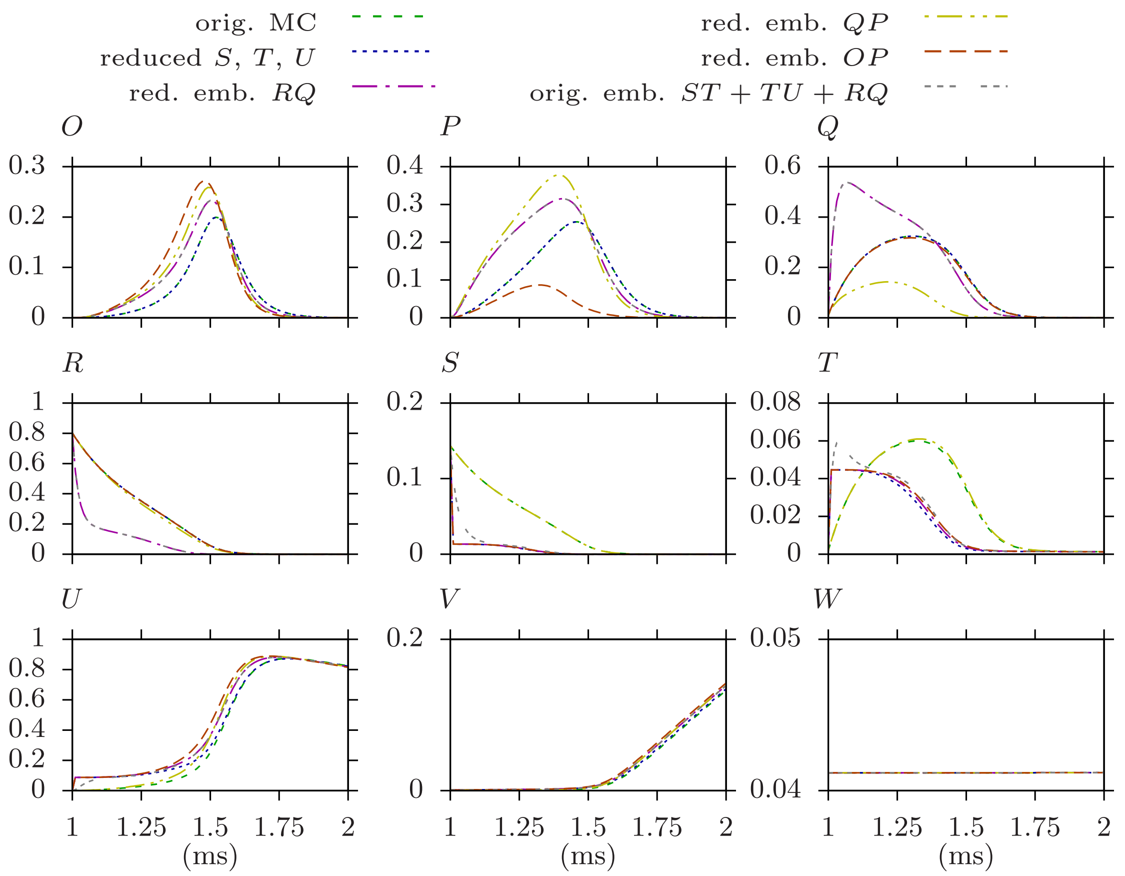


Figure 4.13: Comparison of state occupancy in STU-reduced I_{Na} Markov chain model with further embeddings. Green lines show original model described by equation (4.1), blue lines show STU-reduced model as described by (4.49), magenta lines show RQ-embedding states, yellow lines show QP-embedding, brown lines show OP-embedding. All the embeddings are done on the STU-reduced model with $\varepsilon = 0.1$. Gray lines show embedding of original model with transition rates between states ST, TU and RQ with $\varepsilon = 0.1$. The figure shows the detail of the first 1 ms after the initiation of action potential.

Figure 4.13 shows the detail of the first millisecond of the states occupancy driven by the action potential. The embedding RQ of the reduced model and embeddings ST, TU, RQ in original model affect the same transition rates, but the state occupancy is different in some states (R, S, T, U). The simulated traces of open states O in the STU-reduced model and the RQ-embedding of the STU-reduced model overlap with each other. This suggests that the RQ-reduction would provide a good approximation of the model. We define the RQ-embedding of the system (4.49) as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{MO}M - (\alpha_{OP} + \alpha_{OM})O, \tag{4.50a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{MP}M + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PM} + \alpha_{PO})P, \tag{4.50b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{RQ}R + \alpha_{MQ}M + \alpha_{PQ}P - \left(\frac{1}{\varepsilon}\alpha_{QR} + \alpha_{QM} + \alpha_{QP}\right)Q, \tag{4.50c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{MR}M + \frac{1}{\varepsilon}\alpha_{QR}Q - \left(\alpha_{RM} + \frac{1}{\varepsilon}\alpha_{RQ}\right)R, \tag{4.50d}$$

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \alpha_{OM}O + \alpha_{PM}P + \alpha_{QM}Q + \alpha_{RM}R + \alpha_{VM}V - \\ - (\alpha_{MO} + \alpha_{MP} + \alpha_{MQ} + \alpha_{MR} + \alpha_{MV})M, \tag{4.50e}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{MV}M + \alpha_{WV}W - (\alpha_{VM} + \alpha_{VW})V, \tag{4.50f}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.50g}$$

We put together the terms that contain the parameter ε and write the transition rates matrices and non-homogeneous terms for the states

$$\vec{x} = \begin{bmatrix} R \\ Q \end{bmatrix} \tag{4.51}$$

that we aim to reduce to a new state L. For the reduction we compute

$$\boldsymbol{A}_0 = \begin{bmatrix} -\alpha_{RQ} & \alpha_{QR} \\ \alpha_{RQ} & -\alpha_{QR} \end{bmatrix}, \tag{4.52a}$$

$$\boldsymbol{A}_1 = \begin{bmatrix} -\alpha_{RM} & 0 \\ 0 & -(\alpha_{QM} + \alpha_{QP}) \end{bmatrix}, \tag{4.52b}$$

$$\vec{H} = \begin{bmatrix} \alpha_{MR}M \\ \alpha_{MQ}M + \alpha_{PQ}P \end{bmatrix}, \tag{4.52c}$$

which will be used to find the RQ-reduction in the STU-reduced model.

4.4.2 Choice of Eigenvectors

The scaled eigenvector for the zero eigenvalue and its corresponding adjoint vector are found to be

$$\vec{v}_1 = (\alpha_{QR} + \alpha_{RQ})^{-1} \begin{bmatrix} \alpha_{QR} \\ \alpha_{RQ} \end{bmatrix}, \qquad \vec{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{4.53}$$

4.4.3 Reduction of states R and Q by One State L

We substitute the states R and Q of the STU-reduced system (4.49) by a single state $L = R + Q$. The relation between old and new dynamical variables is given by (3.63) as

$$R = \frac{\alpha_{RQ}}{\alpha_{QR} + \alpha_{RQ}}L, \tag{4.54a}$$

$$Q = \frac{\alpha_{QR}}{\alpha_{QR} + \alpha_{RQ}}L, \tag{4.54b}$$

which yields the following system

$$\frac{\mathrm{d}L}{\mathrm{d}t} =(\alpha_{MR}+\alpha_{MQ})M+\alpha_{PQ}P-\left(\frac{\alpha_{QP}\alpha_{RQ}}{\alpha_{QR}+\alpha_{RQ}}+\frac{\alpha_{QM}\alpha_{RQ}+\alpha_{RM}\alpha_{QR}}{\alpha_{QR}+\alpha_{RQ}}\right)L, \tag{4.55a}$$

$$\frac{\mathrm{d}M}{\mathrm{d}t} =\alpha_{OM}O+\alpha_{PM}P+\frac{\alpha_{QM}\alpha_{RQ}+\alpha_{RM}\alpha_{QR}}{\alpha_{QR}+\alpha_{RQ}}L+\alpha_{VM}V- \\ -(\alpha_{MO}+\alpha_{MP}+(\alpha_{MQ}+\alpha_{MR})+\alpha_{MV})M, \tag{4.55b}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} =\frac{\alpha_{QP}\alpha_{RQ}}{\alpha_{QR}+\alpha_{RQ}}L+\alpha_{MP}M+\alpha_{OP}O-(\alpha_{PQ}+\alpha_{PM}+\alpha_{PO})P. \tag{4.55c}$$

We define new transition rates as

$$\alpha_{LP} =\frac{\alpha_{QP}\alpha_{RQ}}{\alpha_{QR}+\alpha_{RQ}}, \tag{4.56a}$$

$$\alpha_{ML} =\alpha_{MQ}+\alpha_{MR}, \tag{4.56b}$$

$$\alpha_{PL} =\alpha_{PQ}, \tag{4.56c}$$

$$\alpha_{LM} =\frac{\alpha_{QM}\alpha_{RQ}+\alpha_{RM}\alpha_{QR}}{\alpha_{QR}+\alpha_{RQ}}. \tag{4.56d}$$

The system can then be written as

$$\frac{\mathrm{d}L}{\mathrm{d}t} =\alpha_{ML}M+\alpha_{PL}P-(\alpha_{LP}+\alpha_{LM})L, \tag{4.57a}$$

$$\frac{\mathrm{d}M}{\mathrm{d}t} =\alpha_{OM}O+\alpha_{PM}P+\alpha_{LM}L+\alpha_{VM}V-(\alpha_{MO}+\alpha_{MP}+\alpha_{ML}+\alpha_{MV})M, \tag{4.57b}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} =\alpha_{PO}P+\alpha_{MO}M-(\alpha_{OP}+\alpha_{OM})O, \tag{4.57c}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} =\alpha_{LP}L+\alpha_{MP}M+\alpha_{OP}O-(\alpha_{PL}+\alpha_{PM}+\alpha_{PO})P, \tag{4.57d}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} =\alpha_{MV}M+\alpha_{WV}W-(\alpha_{VM}+\alpha_{VW})V, \tag{4.57e}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} =\alpha_{VW}V-\alpha_{WV}W. \tag{4.57f}$$

The diagram of the RQ-STU-reduced Markov chain model is shown in Figure 4.12(c). Figure 4.14 shows the detail of the first millisecond of the simulations of the Markov chain I_{Na} channel driven by recorded action potential onset obtained from simulation with original Markov chain model from [2]. As pointed out previously, the STU-reduction is a very close approximation in state O. The RQ-STU-reduction shows larger deviation from the original model, however the simulated traces are closer than in the OP-reduction.

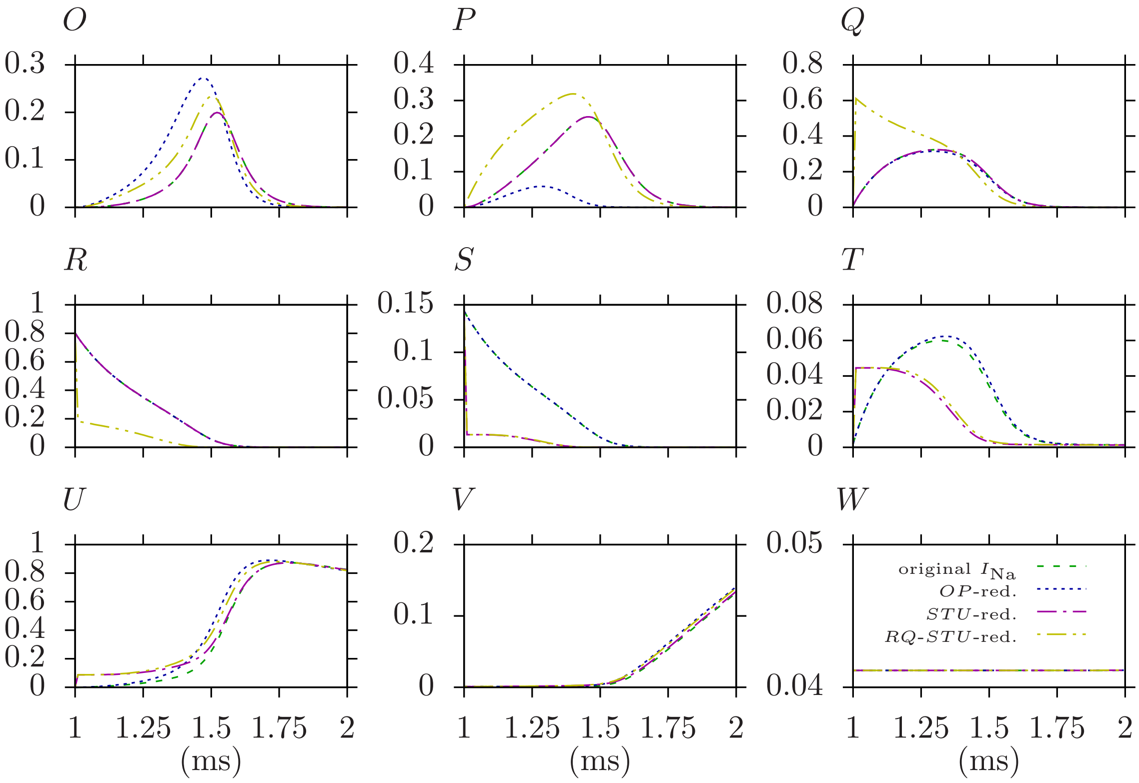


Figure 4.14: Comparison of states occupancy of I_{Na} in driven by recorded action potential. The figure shows the detail of the first 1 ms after the initiation of action potential. Green lines show the original model equation (4.1), blue lines show the OP-reduced model (4.22), magenta lines the STU-reduced model (4.49), yellow lines show RQ-STU-reduced model (4.57).

4.5 Testing of the Reduced Models within a Cell Model

4.5.1 Choice of the Cellular Model

The I_{Na} Markov chain model was reproduced from the Clancy and Rudy (2002), who used it for a simulation study within an environment of a whole cell [2]. The authors did not publish the code along with the article, however Colleen Clancy from University of California, Davis, has kindly provided her version of the source code on request. The complete definition of the cellular model is given in Appendix A.

The embeddings and development of the code was done using the Markov chain model of the I_{Na} channel extracted from the cellular model. The model was driven by an action potential which we recorded from the simulations with the cellular model.

The authors' model was not an autonomous model due to a time dependent stimulation current. We autonomised the model by removing the stimulation. Instead, the action potential is initialised by setting an appropriate initial value of membrane voltage to a value above -50 mV.

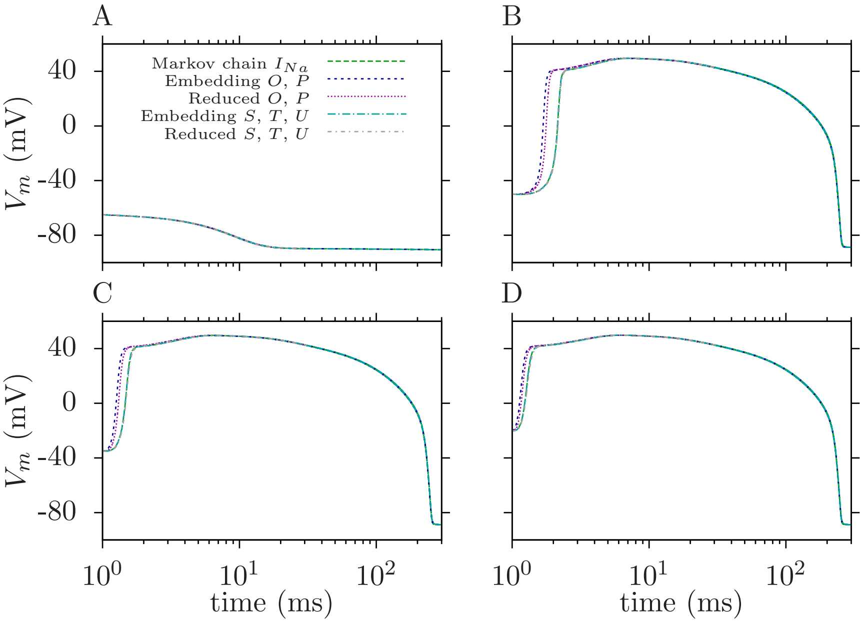


Figure 4.15: Dependence of the action potential ($\mathrm{V_m}$) on the initial conditions of a ventricular cell [2] in the original Markov chain model of I_{Na} (green), OP-embedding (blue), OP-reduction (magenta), STU-embedding (cyan), and STU-reduction (grey) of the Markov chain. The green, cyan, and grey lines overlap. A: $\mathrm{V_{m0}} = -65$ mV, B: $\mathrm{V_{m0}} = -50$ mV, C: $\mathrm{V_{m0}} = -35$ mV, D: $\mathrm{V_{m0}} = -20$ mV. The simulation was initiated at $t_0 = 1$ ms. The horizontal axis has a logarithmic scale.

The action potential was simulated using the original Markov chain formulation of the I_{Na} channel, the Markov chain model with transition rates embedding of α_{OP} and α_{PO}, and the reduced Markov chain model. The initial conditions of I_{Na} were obtained from simulations of 5 pulses of action potential with a cycle length of 1 s using the original Markov chain model. The same initial conditions were used in the embedded and reduced models. The initial state of the new variable in the reduced model was set as $N = O + P$ in the OP-reduction and to $M = R + S + T$ in the RST-reduction.

The forward Euler integration method of the original and the reduced models used the time step size $\Delta t = 0.01$ ms. The embedded model used a time step lower by factor of ten, i.e. $\Delta t = 0.001$ ms. This value was chosen because some of the processes were sped up by factor of ten due to the $\varepsilon = 0.1$.

The authors code uses adaptative time step size of value $\Delta t = 0.005$ ms during the action potential onset and $\Delta t = 0.01$ ms otherwise. This approach allows saving of computational time when the change in membrane voltage is slow. The adaptative time step can be only applied in single cell simulations. In spatial models of cardiac tissue the action potential happens at different moments of time, therefore the time step has to be of the minimal value at all times.

Figure 4.15 shows the time evolution of the membrane voltage at four different initial conditions for voltage $\mathrm{V_{m0}} = \{-65, -50, -35, -20\}$ mV. Panel A shows the

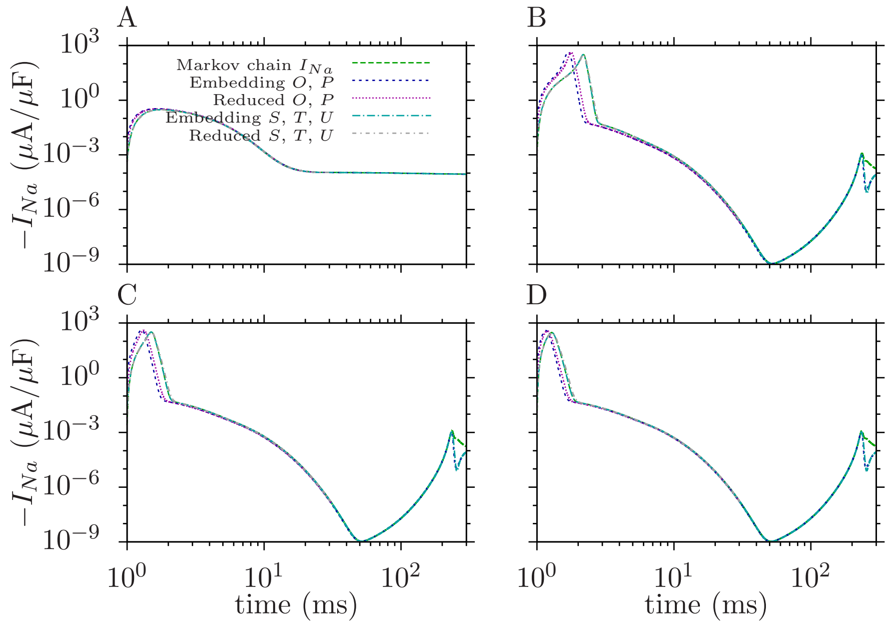


Figure 4.16: Dependence of I_{Na} on the initial conditions in a ventricular cell [2] of I_{Na}, in the original Markov chain model of I_{Na} (green), OP-embedding (blue), OP-reduction (magenta), STU-embedding (cyan), and STU-reduction (grey) of the Markov chain. The green, cyan, and grey lines overlap. A: $\mathrm{V_{m0}} = -65$ mV, B: $\mathrm{V_{m0}} = -50$ mV, C: $\mathrm{V_{m0}} = -35$ mV, D: $\mathrm{V_{m0}} = -20$ mV. The simulation was initiated at $t_0 = 1$ ms. The horizontal axis has a logarithmic scale.

sub-threshold potential. The potential in this case is insufficient to trigger the excitation. Panels B, C and D show simulation where the initial voltage was set above the threshold which causes excitation and the influx of the sodium current into the cell. The higher is the initial value of membrane voltage, the faster is the onset of action potential.

Figure 4.16 shows the the time evolution of the I_{Na} current corresponding to the action potential. The onset of the I_{Na} current is earlier in the OP-embedded and OP-reduced model with respect to the original model. The most remarkable difference is observed in panel B which correspond to the initiation by the membrane voltage of $\mathrm{V_m} = -50$ mV, where the difference is about 0.1 ms. The deviation in long term behaviour of the action potential is insignificant and cannot be perceived in the figure.

4.5.2 Stiffness of the Model

We measure the stiffness of the model by the maximum time step size Δt_{max} which provides a stable solution using the forward Euler solver for the isolated I_{Na} model driven by a recorded action potential. The stiffness of a model can be measured in terms of the maximal time step Δt_{max} it allows.

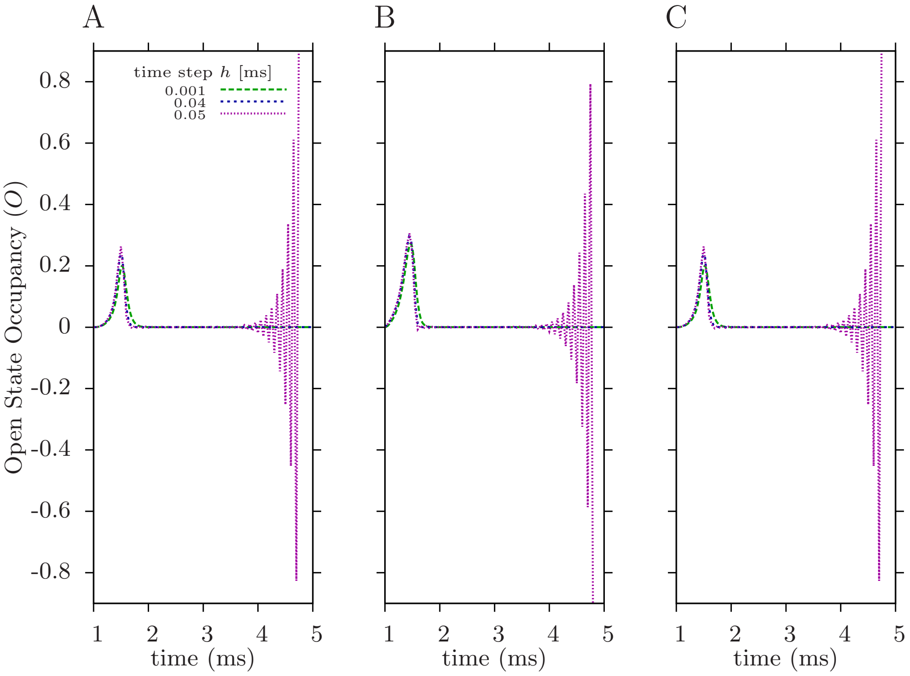


Figure 4.17: Stability of the solution for I_{Na} Markov chain model at different values of the time step. A: original 9-states model; B: reduced (O, P) 8-states ; C: reduced (S, T, U) 7-states I_{Na} Markov chain model. Green line shows a solution with small time step $\Delta t = 0.001$ ms; blue line shows the largest time step which provides stable solution – $\Delta t = 0.04$ ms; magenta line show unstable solution with the time step $\Delta t = 0.05$ ms. The action potential was initiated at $t_0 = 1$ ms. The lines for $\Delta t = 0.001$ (green) and $\Delta t = 0.04$ (blue) overlap.

The simulation step $\Delta t = 0.04$ ms provides stable results. The traces are slightly deviated from the solution at the step size $\Delta t = 0.001$ in both original and reduced I_{Na} model. All three models lose stability around $\Delta t_{max} \approx 0.044$ ms.

When the threshold of stability Δt_{max} is reached, the numerical solution starts to oscillate around the true solution. In a cellular model, this error propagates to other components of the model causing large inaccuracies. Thus, the solution becomes unrealistic and often cause an overflow of the numerical precision in the floating point variables and the simulation fails.

Figure 4.17 shows the detail of the simulated traces during the first 4 ms of the simulation for three reduced models. The characteristics of the instabilities observed are very similar in all of those cases.

4.6 Conclusions

In this chapter we tested the reduction of the dimensionality of the I_{Na} Markov chain model. Initially, we sped up selected transition rates between the states by using a transition rates embedding. This allowed us to identify fast processes whose elimination should provide a good approximation to the original model. We selected a few embedded models and proceeded to apply the reduction according to the procedures described in Chapter 3.

We performed several combinations of dimensionality reductions. Simulated traces using all the selected reductions seem identical after an initial transient. First, we did the reduction of the states O and P. The leading-order approximation of the states O and P are substituted by a new state N. Using the OP-reduction we have to calculate the occupancy of the state O because it contributes to the I_{Na} current. The occupancy of the state O is a voltage dependent proportion of the state N computed from an algebraic formula. Because the state O contributes to the formation of the action potential, a good approximation for O is crucial for an accurate solution of the action potential. To improve the accuracy in the OP-reduction, we included a first-order correction term, in addition to the leading-order. Second, we did the reduction of states S, T and U. The leading-order approximation for the states S, T and U substitutes those states by a new state M. In this case deviation from the solution for the state O of the reduced model from the original model is small. Therefore, the course of the I_{Na} current and time evolution of action potential remains unaffected. Hence, in the STU-reduced model we further reduced the combination of states R and Q.

Although derived reduced models gave reasonably accurate results, they failed to achieve the main objective, which was to address the issue of numerical instability of explicit solvers. The instabilities in the reduced models were still present at roughly the same values of time steps as in the original model. So, similarly to the original model, the numerical integration step was restricted to a relatively small time step size, hence a large number of algebraic operations had to be carried out. As a result the computational cost was still as high as when using original model (because of roughly the same number of algebraic operations). Therefore, the methods were unsatisfactory in providing any practical benefit for the simulations of I_{Na} Markov chain model.

Our hypothesis was that the reduced models were less stiff than the original due to the elimination of the fast transition rates. This would allow the solver to have a larger time step size without the loss of stability, which would lead to a reduced computational cost. However, as suggested by the test simulations with embedded models we performed in Section 4.1.2, the elimination of all fast processes would lead to inaccurate simulations of the open state O as compared to the original model. So, in this system we can or have a good accuracy or improve the stability on the cost of very inacurate results. .

Finally, we comment on the type of models where the dimensionality reduction method might help to address instability issues. The restriction on the maximal time step size in the I_{Na} model was due to fast processes adjacent (in the topology of the Markov chain) to the open state O (which contributes to computation of the ionic current). It might be that in some other Markov chain models the fast processes lie between closed states, which are associated with the computation of the ionic current only indirectly, through their transitions to the open state. In such Markov chains the elimination of fast processes might affect the open state occupancy only slightly. This could provide a good approximation and permit the use of larger time steps. However, none of the Markov chain models we analysed satisfy this requirement.

Chapter 5

Exponential Solvers for Markov Chain Models

This chapter describes the development and application of exponential integration methods. A hybrid method is based on operator splitting and integration for a fixed time step is done analogicaly with “frozen” transition rates. This can be achieved due to the specific coupling between the states of the system. Another method is the so-called matrix Rush-Larsen method. It is an extension of the Rush-Larsen method to Markov chain models. Both methods are applied to the I_{Na} model by Clancy and Rudy (2002) in order to address the instability of explicit solvers and to allow larger time steps.

The following section applies these methods to the RyR and $I_{\mathrm{Ca}(L)}$ model by Faber *et al.* (2007), which is known to be particularly stiff. The exponential integration methods for this model require further adaptation because of the dependence of the Markov chains on multiple variables.

Finally, we perform a theoretical analysis of the numerical truncation error which inevitably happens due to numerical integration and operator splitting methods.

5.1 Application to I_{Na} Model

5.1.1 Operator Splitting for I_{Na} Model

In this section we continue our work with the I_{Na} channel developed by Clancy and Rudy (2002) [2]. This model was described in Subsection 2.4.1. In this section we use the notation corresponding to the reformulation introduced in Subsection 4.1.1, which is shown in Figure 4.1.1(b).

We recall that the Markov chain is described by a system of equations in a form

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}(t)\vec{u}(t), \tag{5.1}$$

where the entries of $\vec{u}$ represent the probability that the channel occupies a particular state (also called states occupancy) and the $\boldsymbol{A}(t)$ is a transition matrix. In case of the I_{Na} model the transition matrix $\boldsymbol{A}(\mathrm{V_m}(t))$ depends on the membrane voltage.

In the previous chapter we have introduced embeddings of pairs of transition rates between two states, and different combinations of embeddings of those pairs. This approach leads to a reduction according to perturbation theory.

Here we introduce an approach where even a single transition rate can be embedded, i.e. multiplied by $1/\varepsilon$. The choice of the transition rates is made depending on their speed, i.e. the probability of the transition from one state to another per unit of time. We use Figure 4.1.1(a) which shows the transition rates at the range of physiological voltages as observed during the action potential between -85 mV and +40 mV.

The numerical instabilities are due to the parts of the model which cause fast variation in the solution. Such elements in the Markov chain model are the fast transition rates. So we aim to treat the fast transition rates in a different way than the rest of the model. We divide the fast transition rates into two groups according to their speed at high and low values of voltages: the first group contains transition rates which are fast at high membrane voltages and slow at low membrane voltages, the second group contains fast transition rates at low membrane voltages and slow at high membrane voltages. This split can be formulated as

$$\boldsymbol{A} = \boldsymbol{A}_0 + \boldsymbol{A}_1 + \boldsymbol{A}_2 \tag{5.2}$$

where the first matrix contains fast transition rates at high voltage as

$$\boldsymbol{A}_0 = \begin{bmatrix} -\alpha_{OU} & \alpha_{PO} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\alpha_{PO} & \alpha_{QP} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\alpha_{QP} & \alpha_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\alpha_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\alpha_{ST} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha_{ST} & -\alpha_{TU} & 0 & 0 & 0 \\ \alpha_{OU} & 0 & 0 & 0 & 0 & \alpha_{TU} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{5.3}$$

the second matrix contains fast transition rates at low voltage as

$$\boldsymbol{A}_1 = \begin{bmatrix} -\alpha_{OP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha_{OP} & -\alpha_{PQ} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_{PQ} & -\alpha_{QR} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_{QR} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha_{TS} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\alpha_{TS} & \alpha_{UT} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\alpha_{UT} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{5.4}$$

and the third matrix contains uniformly slow transition rates at both high and low voltages as

$$\boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \alpha_{UO} & 0 & 0 \\ 0 & -\alpha_{PU} & 0 & 0 & 0 & 0 & \alpha_{UP} & 0 & 0 \\ 0 & 0 & -\alpha_{QT} & 0 & 0 & \alpha_{TQ} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\alpha_{RS} & \alpha_{SR} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{RS} & -\alpha_{SR} & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_{QT} & 0 & 0 & -\alpha_{TQ} & 0 & 0 & 0 \\ 0 & \alpha_{PU} & 0 & 0 & 0 & 0 & A_{2,U} & \alpha_{VU} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \alpha_{UV} & -(\alpha_{VU}+\alpha_{VW}) & \alpha_{WV} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_{VW} & -\alpha_{WV} \end{bmatrix} \tag{5.5}$$

where $A_{2,U} = -(\alpha_{UP} + \alpha_{UO} + \alpha_{UV})$.

Depending on the membrane voltage one of the fast subsystems $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$ is sped up, i.e. multiplied by $1/\varepsilon$. At the high value of membrane voltage the transition matrix $\boldsymbol{A}_0$ is fast and so it becomes the leading order term as

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \frac{1}{\varepsilon}\boldsymbol{A}_0\vec{u} + (\boldsymbol{A}_1 + \boldsymbol{A}_2)\vec{u}, \tag{5.6}$$

while at the low values of the voltage the $\boldsymbol{A}_0$ is slow and so it is considered as a higher-order term, while the transition matrix $\boldsymbol{A}_1$ is fast and is therefore leading order as in the equation

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}_0 u + \frac{1}{\varepsilon}\boldsymbol{A}_1\vec{u} + \boldsymbol{A}_2\vec{u}. \tag{5.7}$$

The matrix $\boldsymbol{A}_2$ is always slow and therefore is a higher-order term.

This section was dedicated for the analysis and division of the ODEs describing I_{Na} model. In the following subsection we suggest a hybrid method to solve this system.

5.1.2 Hybrid Method for I_{Na} model

Solution for Leading Order System at High Voltage

Substituting the $\boldsymbol{A}_0$ from (5.3) to (5.6) the leading order system at high voltages reads as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P - \alpha_{OU}O, \tag{5.8a}$$
$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q - \alpha_{PO}P, \tag{5.8b}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R - \alpha_{QP}Q, \tag{5.8c}$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = -\alpha_{RQ}R, \tag{5.8d}$$
$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\alpha_{ST}S, \tag{5.8e}$$
$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{ST}S - \alpha_{TU}T, \tag{5.8f}$$
$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{OU}O, \tag{5.8g}$$
$$\frac{\mathrm{d}V}{\mathrm{d}t} = 0, \tag{5.8h}$$
$$\frac{\mathrm{d}W}{\mathrm{d}t} = 0. \tag{5.8i}$$

We use an assumption that the transition rates do not change much during one time step and can be “frozen”, i.e. considered constant. Then, due to the specific coupling of those equations, we can obtain an analytic solution as follows. First, we notice that the equations for states R, S, V and W are decoupled and can be solved directly. This solution is then substituted into equations (5.8f) and (5.8c) which are now in a closed form so we obtain the solution of T and Q. The solution for Q is substituted into (5.8b) and the solution of T into (5.8g) and we solve P and U. Finally, we solve the expression for O after we have substituted the solution of P. So, we get the solution as

$$O_{n+1/3} = O_n\mu_{OU} + P_nK_{PO} + Q_nK_{QO} + R_nK_{RO} \tag{5.9a}$$
$$P_{n+1/3} = P_n\mu_{PO} + Q_nK_{QP} + R_nK_{RP} \tag{5.9b}$$
$$Q_{n+1/3} = Q_n\mu_{QP} + R_nK_{RQ} \tag{5.9c}$$
$$R_{n+1/3} = R_n\mu_{RQ} \tag{5.9d}$$

$$S_{n+1/3} = S_n \mu_{ST} \tag{5.9e}$$

$$T_{n+1/3} = T_n \mu_{TU} + S_n K_{ST} \tag{5.9f}$$

$$U_{n+1/3} = U_n + T_n[1-\mu_{TU}] + S_n K_{SU} + O_n[1-\mu_{OU}] + P_n K_{PU} + Q_n K_{QU} + R_n K_{RU} \tag{5.9g}$$

where $\mu_{jk} = \exp(-\alpha_{jk}\Delta t)$, and

$$K_{PO} = \frac{\alpha_{PO}(\mu_{PO}-\mu_{OU})}{\alpha_{OU}-\alpha_{PO}} \tag{5.10a}$$

$$K_{QO} = \frac{\alpha_{PO}\alpha_{QP}(\mu_{QP}-\mu_{OU})}{(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{QP})} - \frac{\alpha_{PO}\alpha_{QP}(\mu_{PO}-\mu_{OU})}{(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{PO})} \tag{5.10b}$$

$$\begin{aligned} K_{RO} = &- \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{QP}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{QP})} + \\ &+ \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{PO}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{PO})} + \\ &+ \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{RQ}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})(\alpha_{OU}-\alpha_{RQ})} - \\ &- \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{PO}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})(\alpha_{OU}-\alpha_{PO})} \end{aligned} \tag{5.10c}$$

$$K_{QP} = \frac{\alpha_{QP}(\mu_{QP}-\mu_{PO})}{\alpha_{PO}-\alpha_{QP}} \tag{5.10d}$$

$$K_{RP} = -\frac{\alpha_{QP}\alpha_{RQ}(\mu_{QP}-\mu_{PO})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})} + \frac{\alpha_{QP}\alpha_{RQ}(\mu_{RQ}-\mu_{PO})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})} \tag{5.10e}$$

$$K_{RQ} = -\frac{\alpha_{RQ}(\mu_{QP}-\mu_{RQ})}{\alpha_{QP}-\alpha_{RQ}} \tag{5.10f}$$

$$K_{ST} = -\frac{\alpha_{ST}(\mu_{TU}-\mu_{ST})}{\alpha_{TU}-\alpha_{ST}} \tag{5.10g}$$

$$K_{SU} = 1 + \frac{\alpha_{ST}\mu_{TU}-\alpha_{TU}\mu_{ST}}{\alpha_{TU}-\alpha_{ST}} \tag{5.10h}$$

$$K_{PU} = 1 - \frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}} \tag{5.10i}$$

$$\begin{aligned} K_{QU} = &\frac{\alpha_{PO}}{\alpha_{PO}-\alpha_{QP}}\left(1-\frac{\alpha_{OU}\mu_{QP}-\alpha_{QP}\mu_{OU}}{\alpha_{OU}-\alpha_{QP}}\right) - \\ &- \frac{\alpha_{QP}}{\alpha_{PO}-\alpha_{QP}}\left(1-\frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}}\right) \end{aligned} \tag{5.10j}$$

$$\begin{aligned} K_{RU} = &- \frac{\alpha_{PO}\alpha_{RQ}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})}\left(1-\frac{\alpha_{OU}\mu_{QP}-\alpha_{QP}\mu_{OU}}{\alpha_{OU}-\alpha_{QP}}\right) + \\ &+ \frac{\alpha_{QP}\alpha_{RQ}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})}\left(1-\frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}}\right) + \\ &+ \frac{\alpha_{PO}\alpha_{QP}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})}\left(1-\frac{\alpha_{OU}\mu_{RQ}-\alpha_{RQ}\mu_{OU}}{\alpha_{OU}-\alpha_{RQ}}\right) - \\ &- \frac{\alpha_{QP}\alpha_{RQ}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})}\left(1-\frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}}\right) \end{aligned} \tag{5.10k}$$

that can be written as

$$\vec{u}_{n+1/3} = \boldsymbol{T}_1(t, \Delta t)\vec{u}_n, \tag{5.11}$$

where the operator matrix is

$$\boldsymbol{T}_1(t,\Delta t) = \begin{bmatrix} \mu_{OU} & K_{PO} & K_{QO} & K_{RO} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_{PO} & K_{QP} & -K_{RP} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_{QP} & -K_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_{ST} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -K_{ST} & \mu_{TU} & 0 & 0 & 0 \\ [1-\mu_{OU}] & K_{PU} & K_{QU} & K_{RU} & K_{SU} & [1-\mu_{TU}] & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.12}$$

Solution for Leading Order System at Low Voltage

Substituting the $\boldsymbol{A}_1$ from (5.4) to (5.7) the leading order system at low voltages reads as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = -\alpha_{OP}O, \tag{5.13a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{OP}O - \alpha_{PQ}P, \tag{5.13b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{PQ}P - \alpha_{QR}Q, \tag{5.13c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{QR}Q, \tag{5.13d}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T, \tag{5.13e}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{UT}U - \alpha_{TS}T, \tag{5.13f}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = -\alpha_{UT}U, \tag{5.13g}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = 0, \tag{5.13h}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = 0. \tag{5.13i}$$

Again, using the specific coupling of the equations we can solve this system of equations. First, we get solution for O, U, V and W which are already in the closed form. Then, we can substitute the solution for U into (5.13f) and the solution for O into (5.13b) which gets in the closed form, so we obtain solutions for T and P. Substituting T also (5.13e) and P into (5.13c) we can solve S and Q. Finally, we substitute Q into (5.13d) to get R. The result can be written as

$$O_{n+2/3} = O_{n+1/3}\mu_{OP} \tag{5.14a}$$
$$P_{n+2/3} = O_{n+1/3}L_{OP} + P_{n+1/3}\mu_{PQ} \tag{5.14b}$$
$$Q_{n+2/3} = O_{n+1/3}L_{OQ} + P_{n+1/3}L_{PQ} + Q_{n+1/3}\mu_{QR} \tag{5.14c}$$
$$R_{n+2/3} = O_{n+1/3}L_{OR} + P_{n+1/3}L_{PR} + Q_{n+1/3}[1-\mu_{QR}] + R_{n+1/3} \tag{5.14d}$$
$$S_{n+2/3} = U_{n+1/3}L_{US} + T_{n+1/3}[1-\mu_{TS}] + S_{n+1/3} \tag{5.14e}$$
$$T_{n+2/3} = U_{n+1/3}L_{UT} + T_{n+1/3}\mu_{TS} \tag{5.14f}$$
$$U_{n+2/3} = U_{n+1/3}\mu_{UT} \tag{5.14g}$$
$$V_{n+2/3} = V_{n+1/3} \tag{5.14h}$$
$$W_{n+2/3} = W_{n+1/3} \tag{5.14i}$$

where

$$L_{OP} = \frac{\alpha_{OP}(\mu_{OP}-\mu_{PQ})}{\alpha_{PQ}-\alpha_{OP}} \tag{5.15a}$$
$$L_{OQ} = \frac{\alpha_{PQ}\alpha_{OP}(\mu_{OP}-\mu_{QR})}{(\alpha_{PQ}-\alpha_{OP})(\alpha_{QR}-\alpha_{OP})} - \frac{\alpha_{PQ}\alpha_{OP}(\mu_{PQ}-\mu_{QR})}{(\alpha_{PQ}-\alpha_{OP})(\alpha_{QR}-\alpha_{PQ})} \tag{5.15b}$$
$$L_{PQ} = \frac{\alpha_{PQ}(\mu_{PQ}-\mu_{QR})}{\alpha_{QR}-\alpha_{PQ}} \tag{5.15c}$$
$$L_{OR} = 1 + \frac{\alpha_{PQ}(\alpha_{OP}\mu_{QR}-\alpha_{QR}\mu_{OP})}{(\alpha_{PQ}-\alpha_{OP})(\alpha_{QR}-\alpha_{OP})} - \frac{\alpha_{OP}(\alpha_{PQ}\mu_{QR}-\alpha_{QR}\mu_{PQ})}{(\alpha_{PQ}-\alpha_{OP})(\alpha_{QR}-\alpha_{PQ})} \tag{5.15d}$$
$$L_{PR} = 1 + \frac{\alpha_{PQ}\mu_{QR}-\alpha_{QR}\mu_{PQ}}{\alpha_{QR}-\alpha_{PQ}} \tag{5.15e}$$
$$L_{US} = 1 + \frac{\alpha_{UT}\mu_{TS}-\alpha_{TS}\mu_{UT}}{\alpha_{TS}-\alpha_{UT}} \tag{5.15f}$$
$$L_{UT} = \frac{\alpha_{UT}(\mu_{UT}-\mu_{TS})}{\alpha_{TS}-\alpha_{UT}} \tag{5.15g}$$

that can be written as

$$\vec{u}_{n+2/3} = \boldsymbol{T}_2(t,\Delta t)\vec{u}_{n+1/3}, \tag{5.16}$$

where the operator matrix is

$$\boldsymbol{T}_2(t,\Delta t)=\begin{bmatrix} \mu_{OP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ L_{OP} & \mu_{PQ} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ L_{OQ} & L_{PQ} & \mu_{QR} & 0 & 0 & 0 & 0 & 0 & 0\\ L_{OR} & L_{PR} & [1-\mu_{QR}] & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & [1-\mu_{TS}] & L_{US} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & L_{UT} & \mu_{TS} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & \mu_{UT} & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.17}$$

Solution for Uniformly Slow System

So far, we have solved the leading order term at high voltage with $\boldsymbol{A}_0$ equation (5.7) and at low voltage $\boldsymbol{A}_1$ from equation (5.7). The complement to these systems remains to be solved. The complementary system contains uniformly slow transition rates at both high and low voltages, and has the form

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t}=\boldsymbol{A}_2\vec{u}, \tag{5.18}$$

The complementary first-order term contains only uniformly slow transition rates (both at high and low voltages). Substituting (5.5) into (5.18) the complementary system reads as

$$\frac{\mathrm{d}O}{\mathrm{d}t}=\alpha_{UO}U, \tag{5.19a}$$
$$\frac{\mathrm{d}P}{\mathrm{d}t}=\alpha_{UP}U-\alpha_{PU}P, \tag{5.19b}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t}=\alpha_{TQ}T-\alpha_{QT}Q, \tag{5.19c}$$
$$\frac{\mathrm{d}R}{\mathrm{d}t}=\alpha_{SR}S-\alpha_{RS}R, \tag{5.19d}$$
$$\frac{\mathrm{d}S}{\mathrm{d}t}=\alpha_{RS}R-\alpha_{SR}S, \tag{5.19e}$$
$$\frac{\mathrm{d}T}{\mathrm{d}t}=\alpha_{QT}Q-\alpha_{TQ}T, \tag{5.19f}$$
$$\frac{\mathrm{d}U}{\mathrm{d}t}=\alpha_{PU}P+\alpha_{VU}V-(\alpha_{UP}+\alpha_{UO}+\alpha_{UV})U, \tag{5.19g}$$
$$\frac{\mathrm{d}V}{\mathrm{d}t}=\alpha_{UV}U+\alpha_{WV}W-(\alpha_{VU}+\alpha_{VW})V, \tag{5.19h}$$
$$\frac{\mathrm{d}W}{\mathrm{d}t}=\alpha_{VW}V-\alpha_{WV}W. \tag{5.19i}$$

Because this system contains only slow transition rates, it is less stiff and the maximal value of the time step size for a stable solution is high. So, instead of finding exponential routine, which is not straightforward by an analytic solution of the system, we approximate the solution using the forward Euler method as

$$O_{n+1} = O_{n+2/3} + \alpha_{UO} U_{n+2/3} \Delta t, \tag{5.20a}$$

$$P_{n+1} = P_{n+2/3} + \left(\alpha_{UP} U_{n+2/3} - \alpha_{PU} P_{n+2/3}\right) \Delta t, \tag{5.20b}$$

$$Q_{n+1} = Q_{n+2/3} + \left(\alpha_{TQ} T_{n+2/3} - \alpha_{QT} Q_{n+2/3}\right) \Delta t, \tag{5.20c}$$

$$R_{n+1} = R_{n+2/3} + \left(\alpha_{SR} S_{n+2/3} - \alpha_{RS} R_{n+2/3}\right) \Delta t, \tag{5.20d}$$

$$S_{n+1} = S_{n+2/3} + \left(\alpha_{RS} R_{n+2/3} - \alpha_{SR} S_{n+2/3}\right) \Delta t, \tag{5.20e}$$

$$T_{n+1} = T_{n+2/3} + \left(\alpha_{QT} Q_{n+2/3} - \alpha_{TQ} T_{n+2/3}\right) \Delta t, \tag{5.20f}$$

$$U_{n+1} = U_{n+2/3} + \left(\alpha_{PU} P_{n+2/3} + \alpha_{VU} V_{n+2/3} - (\alpha_{UP} + \alpha_{UO} + \alpha_{UV}) U_{n+2/3}\right) \Delta t, \tag{5.20g}$$

$$V_{n+1} = V_{n+2/3} + \left(\alpha_{UV} U_{n+2/3} + \alpha_{WV} W_{n+2/3} - (\alpha_{VU} + \alpha_{VW}) V_{n+2/3}\right) \Delta t, \tag{5.20h}$$

$$W_{n+1} = W_{n+2/3} + \left(\alpha_{VW} V_{n+2/3} - \alpha_{WV} W_{n+2/3}\right) \Delta t, \tag{5.20i}$$

that can be also written as

$$\vec{u}_{n+1} = \vec{u}_{n+2/3} + \Delta t \boldsymbol{A}_2(t_n) \vec{u}_{n+2/3}. \tag{5.21}$$

Summary of Hybrid Method

Then the equations (5.11), (5.16), and (5.21) constitute the computational algorithm. This algorithm can be reformulated as

$$\vec{u}_{n+2/3} = \boldsymbol{T}(t, \Delta t) \vec{u}_n, \tag{5.22a}$$

$$\vec{u}_{n+1} = \vec{u}_{n+2/3} + \Delta t \boldsymbol{A}_2(t_n) \vec{u}_{n+2/3}, \tag{5.22b}$$

where $\boldsymbol{T}(t, \Delta t) = \boldsymbol{T}_1(t, \Delta t) \boldsymbol{T}_2(t, \Delta t)$ that combines both (5.12) and (5.17). In order to speed up the computation, the algorithm uses tabulation of the matrix $\boldsymbol{T}(t, \Delta t)$, which contains several exponential functions that are computationally demanding. The details about the tabulation are given in Subsection 6.2.1.

5.1.3 Matrix Rush-Larsen for I_{Na} Model

The Rush-Larsen method is a numerical method for gate models. The details of the method were described in Subsection 3.2.3 by equation (3.131). The basic idea is that the transition rates are assumed to be constant for the duration of one time step. Such an equation is then solved analytically. The analytical solution is then used in an iterative integration scheme, where the transition rates are updated for the new values at the beginning of each time step. The Rush-Larsen method is not directly applicable to Markov chain models. Here we generalise the idea for a Markov chain models.

For the purposes of the I_{Na} model it can be written as

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}(t_n)\Delta t\right)\vec{u}(t_n), . \tag{5.23}$$

For practical reasons we can write it in the form

$$\vec{u}_{n+1} = \boldsymbol{T}(t_n)\vec{u}(t_n) = \boldsymbol{V}(t_n)\exp\left(\boldsymbol{\Lambda}(t_n)\Delta t\right)\boldsymbol{W}^{\boldsymbol{T}}(t_n)\vec{u}(t_n), \tag{5.24}$$

in which we use eigenvalue decomposition $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{W}^{\boldsymbol{T}}$ where the right eigenvectors are concatenated in the columns of matrix $\boldsymbol{V}$, and left eigenvectors are concatenated in the columns of matrix $\boldsymbol{W}$. The order of the eigenvectors corresponds to the same order in which are sorted the eigenvalues in the eigenvalue matrix $\boldsymbol{\Lambda}$.

The motivation to introduce the eigenvalue decomposition is because the computation of the exponential of diagonal eigenvalue matrix is straightforward as

$$\exp\left(\begin{bmatrix} \lambda_1\Delta t & 0 & \dots & 0 \\ 0 & \lambda_2\Delta t & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_N\Delta t \end{bmatrix}\right) = \begin{bmatrix} \exp(\lambda_1\Delta t) & 0 & \dots & 0 \\ 0 & \exp(\lambda_2\Delta t) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \exp(\lambda_N\Delta t) \end{bmatrix}. \tag{5.25}$$

That assumes that the transition rates matrix is diagonalisable.

5.1.4 Simulation in a Cellular Model

The two previous subsections described two algorithms for the computation of the I_{Na} model. In this subsection we apply the algorithm within a cellular model. For this purpose we have implemented the hybrid algorithm (5.22) and the Matrix Rush-Larsen (5.24) into a single cell model by Clancy and Rudy (2002) [2]. The details of the cellular model can be found in Appendix A.

Figure 5.1 shows the results of the simulations. In the simulation with the forward Euler method the system becomes unstable at values of the time step above $\Delta t = 0.05$, so the results are only shown for time step $\Delta t = 0.01$ and 0.04 ms. The results for the matrix Rush-Larsen and hybrid methods are shown for time steps $\Delta t = 0.01, 0.04$, and 0.10 ms. Those methods are unconditionally stable. This is because all the eigenvalues of the system are non-positive and real, which means that the time evolution of the solution will approach its steady-state.

A generic property of any Markov chain is a state conservation law, which implies that the sum of the occupancies of all states of a Markov chain is equal

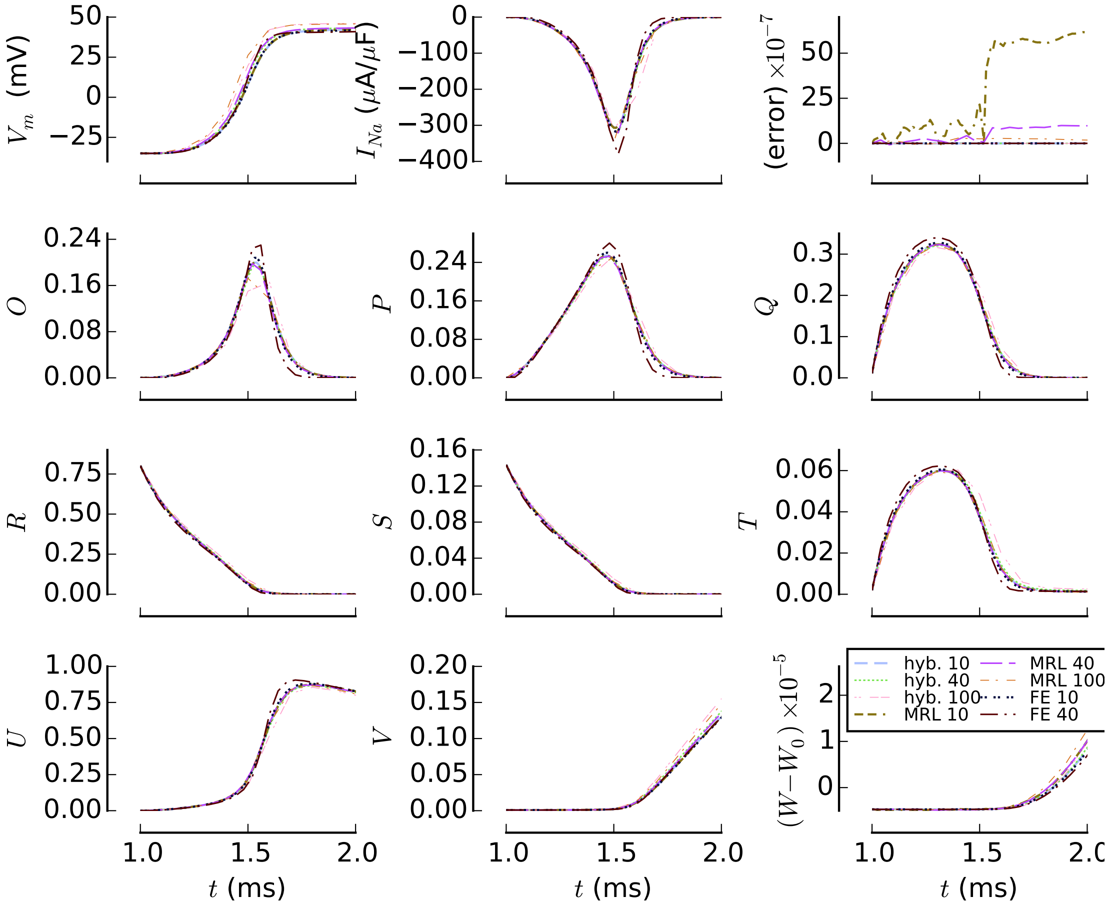


Figure 5.1: Action potential ($\mathrm{V_m}$), I_{Na} current, error from conservation law, and I_{Na} Markov chain states occupancy (zoom to first 1 ms). The algorithm combines the forward Euler solution of the slow subsystem with the analytical solution of the fast subsystem at low and high voltage (hyb); matrix Rush-Larsen (hyb); and forward Euler solution (FE). The solution was obtained at time step $\Delta t = 0.01, 0.04$ and 0.10 ms (except for FE, which is unstable at 0.01).

to 1. We use the deviation from the state conservation law as an indicator of the accuracy of the computation.

Another consideration is the accuracy of the methods. As is discussed in more detail in Section 5.3, the accuracy is inversely proportional to the time step size. So, when we further increase the time step we obtain significant inaccuracies, e.g. at $\Delta t = 2$ ms there is 30 mV overshoot in the action potential as compared with the value at $\Delta t = 0.01$ ms. Increasing the time step even further, we observe unphysical values of concentration at about $\Delta t = 2$ ms in the hybrid method. The MRL method allows the time step to be increased up to 7.5 ms. At this time step the model becomes unstable due to other components than Markov chains.

The figure shows that about 90% of the channels occupy the states R and S before the initiation of the action potential. After the initiation, the channels transit rapidly towards the open state O, and within about 0.7 μs almost all channels

Table 5.1: Computational time [s] of 100 pulses with cycle length of 1000 ms. The first column specifies the method used for computation as forward Euler ("FE"), matrix Rush-Larsen ("MRL"), hybrid operator spliting ("hybrid"). The method with tabulation of transition rates or transtion rates matrices are denoted by "(tab.)". The columns denoted by "I_{Na}" show accumulated computational time for simulation of I_{Na} Markov chain only, while columns denoted by "Total" show time spend by the simulation of the whole cell (with output disabled). The simulation for $\Delta t = 0.10$ ms in FE method cannot be done due to numerical instability.

	$\Delta t = 0.01$		$\Delta t = 0.04$		$\Delta t = 0.10$	
I_{Na} Model	I_{Na}	Total	I_{Na}	Total	I_{Na}	Total
FE	5.44	24.01	1.29	6.02		
FE (tab.)	2.74	21.38	0.69	5.36		
MRL (tab.)	4.79	24.36	1.23	6.23	0.54	2.45
hybrid	9.51	28.13	2.22	7.04	0.79	2.83
hybrid (tab.)	2.89	21.71	0.77	5.49	0.37	2.21

reside in the state U. Then the channels slowly transit to the state V where they stay until the resting potential is recovered (not shown).

The results for the time step $\Delta t = 0.01$ ms are consistent in all panels. The FE is still stable at $\Delta t = 0.40$ ms, but compared to the MRL and hybrid solution we observe a larger peak and faster decay in the open state occupancy. This suggests that the exponential integrators can be used to improve accuracy of the solution even in situations when the instability is not a concern.

Table 5.1 shows the elapsed CPU time during the simulations in the cell model. The simulations were performed for 100 pulses of CL 1000 ms. To obtain only the results of the computations, the program did not generate any output. The C language source code was compiled using the GNU compiler collection (version 4.7.2) and the simulations were performed on Intel Core i5-3470 CPU with clock frequency 3.20 GHz under the GNU/Linux operating system. Five consecutive runs were performed and the minimum necessary time is shown in the table. Each action potential was initiated by simulating an external potassium current raising the membrane voltage to $\mathrm{V_m} = -35$ mV. The time step of the simulations was $\Delta t = 0.01, 0.04$ for the forward Euler method, because the model becomes unstable for time steps above 0.05 ms. The hybrid and MRL method were also used in simulations with a time step of $\Delta t = 0.10$ ms.

The MRL method was simulated using tabulated eigenvalues and eigenvectors as computed for the transition rates matrices using the mathematical software Sage [20]. The hybrid method was simulated with and without tabulation. The table of pre-calculated values was created for values of voltage ranging from -100 to 70 mV with grid step of 0.01 mV.

Comparing the computational cost of the methods with tabulations, the forward Euler is the most efficient. However, the most important benefit of hybrid and MRL methods is the possibility of using larger time steps without the loss of stability.

5.2 Application to RyR and $I_{\mathrm{Ca}(L)}$ Models

5.2.1 Cellular model

The cardiac myocyte model published by Faber *et al.* (2007) [1] is an example of a numerically stiff model. That means that the numerical integration using explicit solvers requires very small step size to avoid instability.

We obtained the `C` source code of the Faber model from the Rudy Laboratory website [21]. In the model all of the gate ion channels are implemented using the Rush-Larsen method, which always yields a stable solution. The Markov chain models are used for Ryanodine receptor (RyR) and $I_{\mathrm{Ca}(L)}$. The models use the forward Euler scheme, which is also employed for the integration of ionic concentrations ($[\mathrm{Ca}^{2+}]_i$, $[\mathrm{Na}^{+}]_i$, $[\mathrm{K}^{+}]_i$) and the membrane voltage $\mathrm{V_m}$. The suggested time step in the source code is $\Delta t = 1\ \mu\mathrm{s}$ and increasing the time step above $\Delta t = 6\ \mu\mathrm{s}$ results in instabilities causing significant deviation from the true solution, and above $\Delta t = 9\ \mu\mathrm{s}$ the simulation fails in the overflow in the double floating point values.

In this section we describe an application of hybrid method combining Matrix Rush-Larsen and forward Euler to the RyR and $I_{\mathrm{Ca}(L)}$ channels. This achieves a stable solution of such stiff models, and allows larger time steps and substantially reduces computational cost. In our case, we achieved to increase the time step, while obtaining a stable solution from 6 $\mu\mathrm{s}$ to 180 $\mu\mathrm{s}$. This leads to savings of computational cost of about 96%.

5.2.2 RyR Markov Chain Model

The RyR model of Faber *et al.* (2007) was described in the subsection 2.4.3. The diagram of the Markov chain model of RyR is shown in Figure 2.14. The Figure 5.2 shows transition rates, i.e. the probability of the transitions per millisecond, given that the channel resides in a specific state.

Transition rates between the states in the top row (horizontal transition rates in Figure 2.14) are identical in both rows, i.e. a transition rate in the top row corresponds to the transition rate at the same location in the bottom row. The transition rates β are given constants not depending on any other variables. The transition rates α depend on calcium concentration $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. The values of those transition rates in the physiological conditions, when $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$ is between 10^{-4} and $0.06\ \mathrm{mM}$, range from 10^{-2} to about $10^{2}\ \mathrm{ms}^{-1}$.

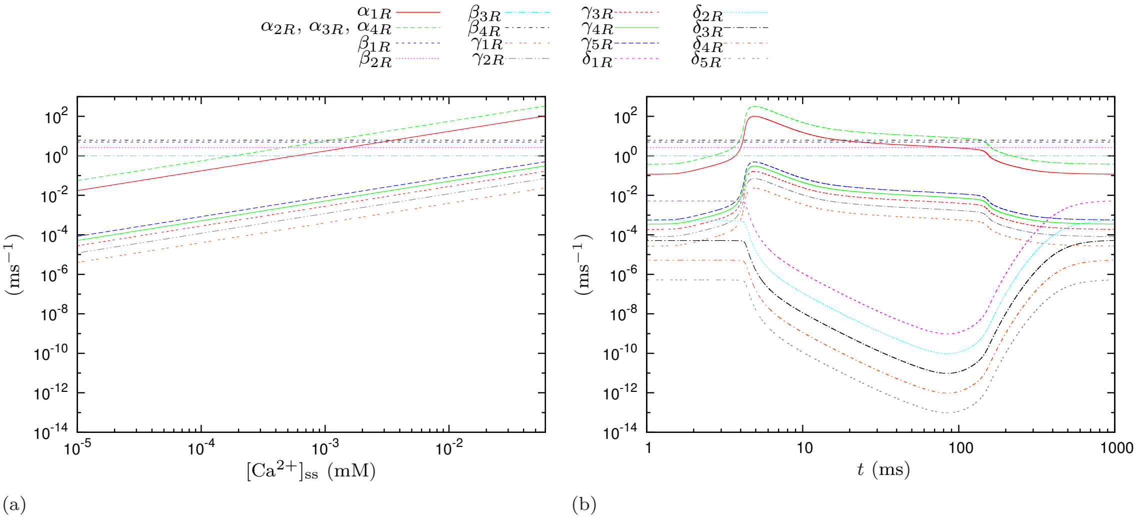


Figure 5.2: Transition rates in the Markov chain model of RyR (diagram in Figure 2.14). Panel (a) shows transition rates as function of $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. Panel (b) shows the transition rates during the action potential. The plots are in logarithmic scale.

The transitions between the top and the bottom row on the diagram (vertical transition rates in Figure 2.14) are uniformly slow. Their maximum value only reaches 1 ms^{-1}. The transition rates γ (transitions from the top to the bottom on the diagram) depend only on the subspace calcium concentration $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$, while the transitions δ (transitions in the opposite direction) are functions of two calcium concentrations. The first is the calcium concentration in the subspace $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$, like in the other transition rates, and second is the calcium concentration in the junctional space of sarcoplasmic reticulum (JSR) $[\mathrm{Ca}^{2+}]_{\mathrm{JSR}}$. Because of the dependence on two variables, these transition rates have been shown only in panel (b) in Figure 5.2.

To study the numerical properties of the cellular model, we performed a number of simulations with different time step sizes. Figure 5.3 shows the result of the simulations. The observed numerical instability first occurred in state O_1 at $\Delta t = 6.7\ \mu\mathrm{s}$. We also start observing instabilities in the calcium concentration in the subspace $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$ at time step size $\Delta t = 7.0\ \mu\mathrm{s}$, which is because the state O_1 is directly connected with the calcium release from the sarcoplasmic reticulum.

The membrane voltage $\mathrm{V_m}$ and other dynamical variables do not present instability until the time-step size reaches values of $\Delta t > 9.0016\ \mu\mathrm{s}$. Below this value, the amplitude of the instability oscillations in the calcium concentration grow larger as we increase the time step. Then, the calcium concentration drifts to values close to zero, however, it still remains in the physiologically plausible range. As we increase the time step even further, the calcium concentration reaches physically impossible negative values, and this error propagates to other variables in the model, which causes the simulation to fail (not shown).

Comparing the form of the speed of the transition rates under the action potential Figure 5.2b with Figure 5.3 we observe a clear correspondence between

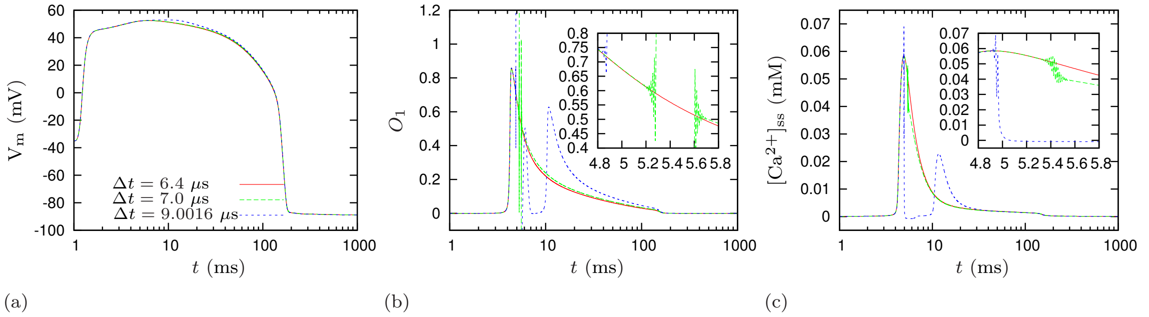


Figure 5.3: Numerical instability caused by RyR model using forward Euler integration.

the time point when the instability occurs, and the time when the fastest transition rates ($\alpha_{2R} = \alpha_{3R} = \alpha_{4R}$) reach their maximum. This confirms that the instability in the forward Euler solver occurs due to the fast transition rates.

To address the instability we suggest employing the matrix Rush-Larsen solver as described in Subsection 5.3.3, together with tabulating the coefficients of the exponential operator matrix (see Subsection 6.2.1 for details about tabulation). However, the implementation of the algorithm for RyR channel is more challenging, because of the dependence of the transition rates on more than one dynamical variable. Tabulation on a multivariable grid is possible, but more demanding in terms of memory and computational resources. Hence, we employ the idea of operator splitting.

The RyR model can be split into subsystems according to the speeds of the transition rates. The first subsystem contains α and β, which are fast. The second subsystem contains δ and γ, which are uniformly slow. Then the subsystem with the slow transition rates can be readily solved using forward Euler solver. After the splitting, the system can be written in the form

$$\boldsymbol{M}([\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \mathrm{CSQN}) = \left(\begin{array}{c|c} \boldsymbol{F} & 0 \\ \hline 0 & \boldsymbol{F} \end{array}\right) + \boldsymbol{G} \tag{5.26}$$

where the first matrix is split into two identical matrices $\boldsymbol{F}$, due to the particular topology of the RyR where the transition rates in both rows of the model are identical. The $\boldsymbol{F} = \boldsymbol{F}([\mathrm{Ca}^{2+}]_{\mathrm{ss}}(t))$ represents the “horizontal” transition rates in the diagram as

$$\boldsymbol{F} = \begin{bmatrix} -\beta_{1R} & \alpha_{1R} & 0 & 0 & 0 \\ \beta_{1R} & -(\alpha_{1R} + \beta_{2R}) & \alpha_{2R} & 0 & 0 \\ 0 & \beta_{2R} & -(\alpha_{2R} + \beta_{3R}) & \alpha_{3R} & 0 \\ 0 & 0 & \beta_{3R} & -(\alpha_{3R} + \beta_{4R}) & \alpha_{4R} \\ 0 & 0 & 0 & \beta_{4R} & -\alpha_{4R} \end{bmatrix}, \tag{5.27}$$

and $\|\boldsymbol{G}(\dots)\| \lesssim 1\,\mathrm{ms}^{-1}$ represents the "vertical" transition rates as

$$\boldsymbol{G} = \begin{bmatrix} -\gamma_{1R} & 0 & 0 & 0 & 0 & \delta_{1R} & 0 & 0 & 0 & 0 \\ 0 & -\gamma_{2R} & 0 & 0 & 0 & 0 & \delta_{2R} & 0 & 0 & 0 \\ 0 & 0 & -\gamma_{3R} & 0 & 0 & 0 & 0 & \delta_{3R} & 0 & 0 \\ 0 & 0 & 0 & -\gamma_{4R} & 0 & 0 & 0 & 0 & \delta_{4R} & 0 \\ 0 & 0 & 0 & 0 & -\gamma_{5R} & 0 & 0 & 0 & 0 & \delta_{5R} \\ \gamma_{1R} & 0 & 0 & 0 & 0 & -\delta_{1R} & 0 & 0 & 0 & 0 \\ 0 & \gamma_{2R} & 0 & 0 & 0 & 0 & -\delta_{2R} & 0 & 0 & 0 \\ 0 & 0 & \gamma_{3R} & 0 & 0 & 0 & 0 & -\delta_{3R} & 0 & 0 \\ 0 & 0 & 0 & \gamma_{4R} & 0 & 0 & 0 & 0 & -\delta_{4R} & 0 \\ 0 & 0 & 0 & 0 & \gamma_{5R} & 0 & 0 & 0 & 0 & -\delta_{5R} \end{bmatrix}. \tag{5.28}$$

Hence, for this Markov chain we use a mix of Matrix Rush-Larsen and forward Euler according to Lie style operator splitting (as will be described in Section 5.3.4) as

$$\vec{v}_{n+1/2} = \exp\left(\Delta t\, \boldsymbol{F}(t_n)\right)\, \vec{v}_n, \tag{5.29a}$$

$$\vec{w}_{n+1/2} = \exp\left(\Delta t\, \boldsymbol{F}(t_n)\right)\, \vec{w}_n, \tag{5.29b}$$

$$\vec{u}_{n+1} = \vec{u}_{n+1/2} + \Delta t\, \boldsymbol{G}(t_n)\, \vec{u}_{n+1/2}, \tag{5.29c}$$

where

$$\vec{v} = [I_1, I_2, I_3, I_4, I_5]^T$$

contains the states of the top row in Figure 2.14, and

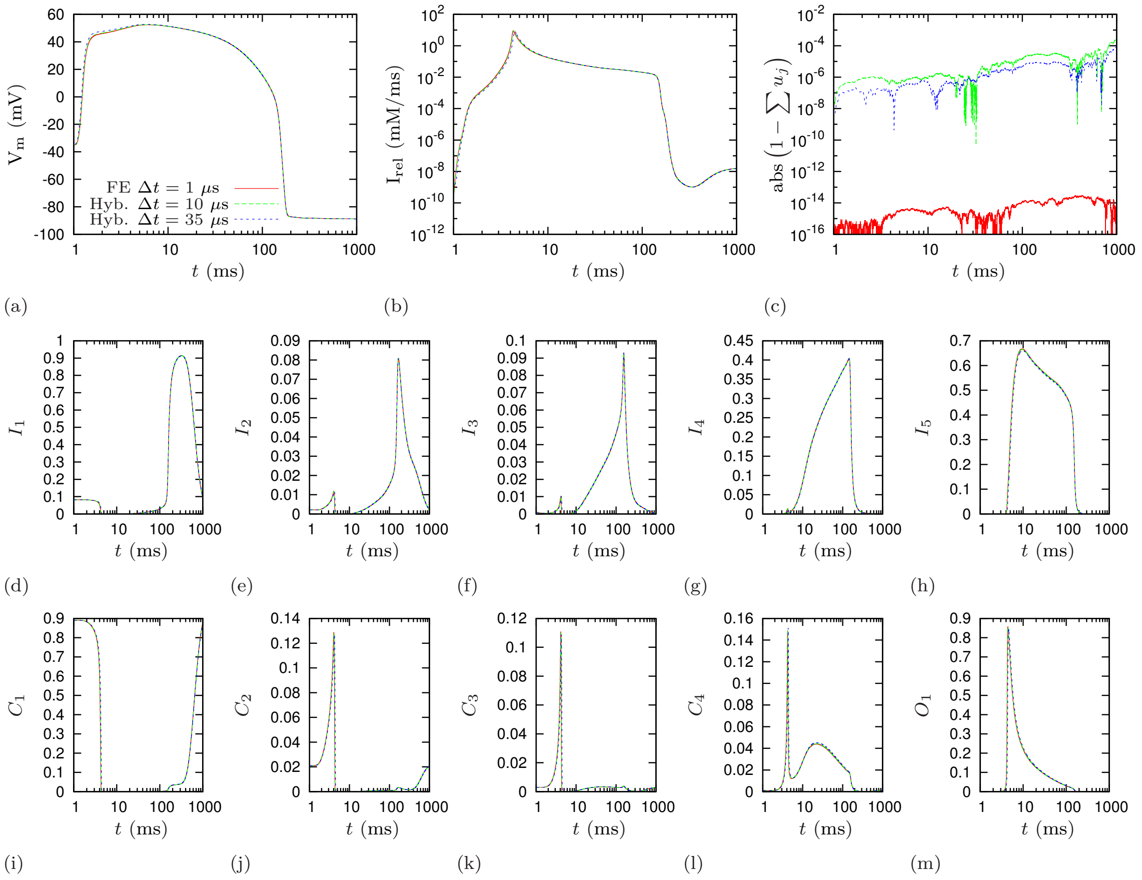
$$\vec{w} = [C_1, C_2, C_3, C_4, O_1]^T$$

contains the states of the bottom row in the figure. Then the state vector including all the states can be written as

$$\vec{u} = \begin{bmatrix} \vec{v} \\ \vec{w} \end{bmatrix}. \tag{5.30}$$

Before we start with the computation, we create a look-up table of eigenvalues and eigenvectors of $\boldsymbol{F}$ for a range of values of control variable $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. The transition rates in $\boldsymbol{G}$ are not tabulated as the tabulation does not offer a major speed-up for the forward Euler method which will be used for the integration of this slow subsystem.

The tabulation variable for the $\boldsymbol{F}$ remains $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. Since $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$ ranges over five orders of magnitude, the tabulation for a regular grid of sufficient precision would be computationally and memory expensive. Therefore, we consider the



(a) (b) (c)

(d) (e) (f) (g) (h)

(i) (j) (k) (l) (m)

Figure 5.4: Comparison of integrators (forward Euler and hybrid) for RyR Markov chain model: (a) action potential; (b) I_{rel}; (c) deviation from states conservation law; (d-m) states occupancy in RyR MC model.

rates as functions of $\mathcal{T} \equiv \ln([\mathrm{Ca}^{+2}]_{\mathrm{ss}})$ and use a regular grid in $\mathcal{T}$ (as described in Subsection 6.2.1).

Using the suggested hybrid method, the instability observed previously at the time step values $\Delta t > 6.7$ $\mu\mathrm{s}$ has disappeared (not shown). The instability reappears at time steps larger than $\Delta t > 37$ $\mu\mathrm{s}$ due to the $I_{\mathrm{Ca}(L)}$ Markov chain model. In order to eliminate this instability, we have implemented an exponential solver for $I_{\mathrm{Ca}(L)}$, as described in the subsequent subsection.

5.2.3 $I_{\mathrm{Ca}(L)}$ Markov Chain Model

The RyR Markov chain was the primary factor limiting the step size. The Faber *et al.* (2006) model contains another stiff Markov chain model of membrane calcium current $I_{\mathrm{Ca}(L)}$. Having applied the hybrid method for the RyR channel as described in the previous section, the stiffness of the $I_{\mathrm{Ca}(L)}$ model is the next cause of instability, which is observed as we increase the time step to higher values.

The detailed description of the $I_{\mathrm{Ca}(L)}$ model was given in Section 2.4.2. The diagram of the model is shown in Figure 2.13. The $I_{\mathrm{Ca}(L)}$ model operates in two modes: the Ca-mode (bottom layer of the diagram) and the $\mathrm{V_m}$-mode (top layer of

new state	old state
G	$C_0 + C_{0\mathrm{Ca}}$
H	$C_1 + C_{1\mathrm{Ca}}$
I	$C_2 + C_{2\mathrm{Ca}}$
L	$C_3 + C_{3\mathrm{Ca}}$
$O_{\mathrm{Ca}(L)}$	$O + I_{\mathrm{Ca}}$
M	$I_{Vs} + I_{Vs\mathrm{Ca}}$
N	$I_{Vf} + I_{Vf\mathrm{Ca}}$

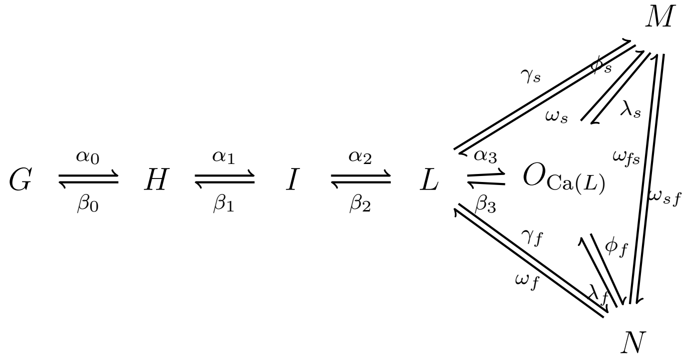


Figure 5.5: New terminology of simplified $I_{\mathrm{Ca}(L)}$ model. Correspondence between the old and new states (left), simplified diagram of the Markov chain (right).

the diagram), where the transition rate θ determines the probability of transition to $\mathrm{V_m}$-mode and transition rate δ back to Ca-mode. The conductive state O is present only in the $\mathrm{V_m}$-mode, and all the states in the Ca-mode are non-conductive.

The transitions between the modes are calcium dependent, while the transitions within the modes are voltage dependent. The states and transition rates within both modes correspond to each other, i.e. given that the location within the mode is the same, the probability of transitions to the neighbouring states is identical in both modes. This allows factoring out the Ca^{2+} dependent inactivation as a gate model. This gate model is then multiplied with an open probability of a simplified Markov chain. The simplified Markov chain has a form of a single layer of any of the modes. The states occupancies in the simplified model correspond to the sum of the corresponding state occupancies in both modes. The diagram of the simplified Markov chain is shown in Figure 5.5 (right hand side). The table on the left in the figure shows the correspondence between the new notation and the states of the authors' model. The idea of factoring out the calcium dependent transition is also used in the authors' code [21], however the equations in the paper are presented in the long form with 14 rather than 7 states [1].

This factorisation procedure gives the following system of ODEs

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \delta(1-q) - \theta q, \tag{5.31a}$$

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{D}(\mathrm{V_m}(t))\vec{u} \tag{5.31b}$$

where the gate q controlls the calcium dependent switching between the modes The Markov chain states are

$$\vec{u} = [G, H, I, L, O_{\mathrm{Ca}(L)}, N, M]^T, \tag{5.32}$$

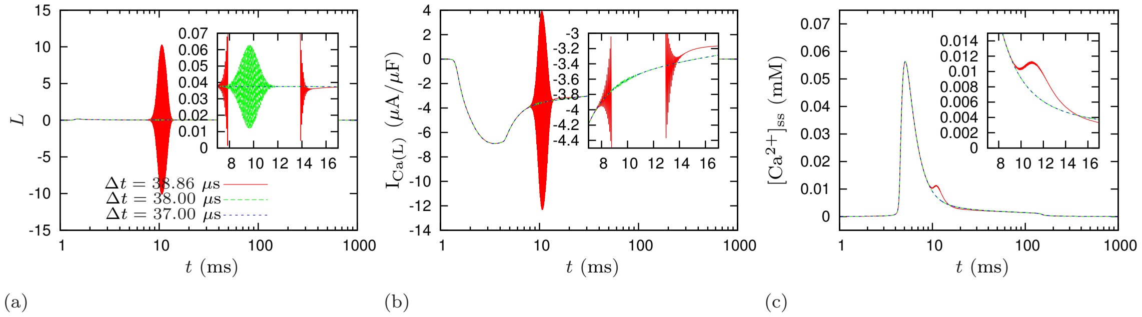


Figure 5.6: Numerical instability caused by $I_{\mathrm{Ca}(L)}$ model using forward Euler integration and hybrid method for RyR channel.

and $\boldsymbol{D}$ is the transition matrix of the simplified Markov chain. The transition matrix reads as

$$\boldsymbol{D} = \begin{bmatrix} -\alpha_0 & \beta_0 & 0 & 0 & 0 & 0 & 0 \\ \alpha_0 & -(\alpha_1+\beta_0) & \beta_1 & 0 & 0 & 0 & 0 \\ 0 & \alpha_1 & -(\beta_1+\alpha_2) & \beta_2 & 0 & 0 & 0 \\ 0 & 0 & \alpha_2 & D_L & \beta_3 & \omega_f & \omega_s \\ 0 & 0 & 0 & \alpha_3 & D_O & \lambda_f & \lambda_s \\ 0 & 0 & 0 & \gamma_f & \phi_f & D_N & \omega_{sf} \\ 0 & 0 & 0 & \gamma_s & \phi_s & \omega_{fs} & D_M \end{bmatrix}, \tag{5.33}$$

where the diagonal entries are

$$D_L = -\,(\gamma_f + \gamma_s + \alpha_3 + \beta_2), \tag{5.34a}$$

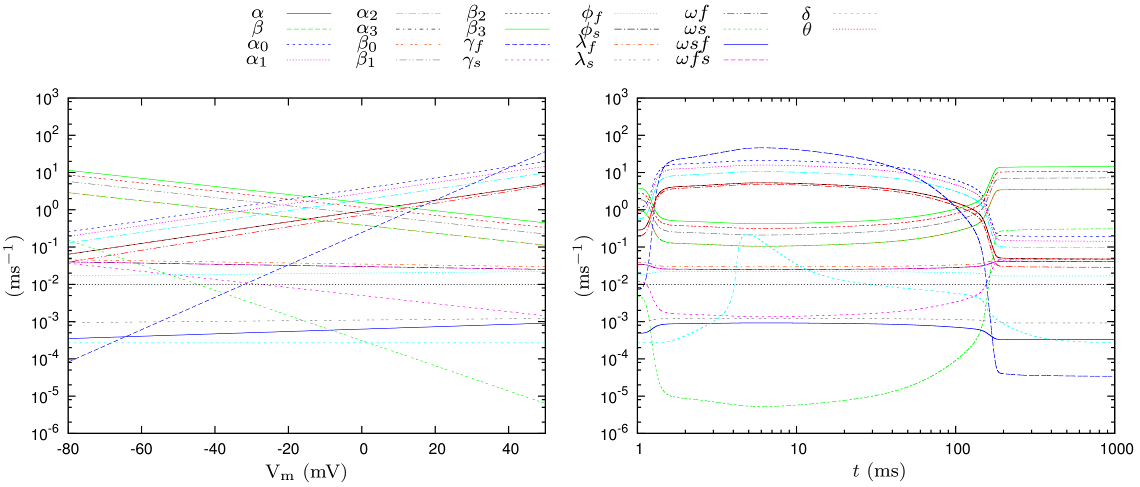
$$D_O = -\,(\phi_f + \phi_s + \beta_3), \tag{5.34b}$$

$$D_N = -\,(\omega_f + \lambda_f + \omega_{fs}), \tag{5.34c}$$

$$D_M = -\,(\omega_s + \lambda_s + \omega_{sf}). \tag{5.34d}$$

The transition matrix is dependent only on voltage, which simplifies the tabulation of its eigenvectors and eigenvalues. For the purposes of tabulation we use values of $\mathrm{V_m}$ ranging from -100 mV to 70 mV with a time-step of 0.01 mV.

Figure 5.6 shows the instability caused by the $I_{\mathrm{Ca}(L)}$ model using the forward Euler solver. The largest time step that allows a stable solution is $\Delta t = 37\ \mu$s. The state L is the first one to become unstable, which happens at the value of time step of $\Delta t = 38\ \mu$s. This instability can hardly be observed in other variables because of only a minor influence of the state L on the $I_{\mathrm{Ca}(L)}$ current. However, we observe a wide oscillations around the exact solution at time steps $\Delta t = 38.86\ \mu$s. These oscillations propagate to $I_{\mathrm{Ca}(L)}$ and cause a drift in $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. Using even higher time steps leads to a propagation of fatal numerical errors causing the simulation to fail.



(a) (b)

Figure 5.7: Transition rates in $I_{\mathrm{Ca}(L)}$ model. (a) as functions of membrane voltage ($\mathrm{V_m}$); (b) during action potential.

There is no clear separation between the speed of the transition rates of the $I_{\mathrm{Ca}(L)}$ model (as shown in Figure 5.7). Similarly to the RyR model, the instability occurs at the same point in time as the fastest transition rate (γ_f) reaches its maximum.

To address the instability we use Rush-Larsen method for the gate model and matrix Rush-Larsen for the Markov chain as

$$q_{n+1} = \frac{\delta}{\delta+\theta} - \left(\frac{\delta}{\delta+\theta} - q_n\right)\exp\left(-(\delta+\theta)\Delta t\right), \tag{5.35a}$$

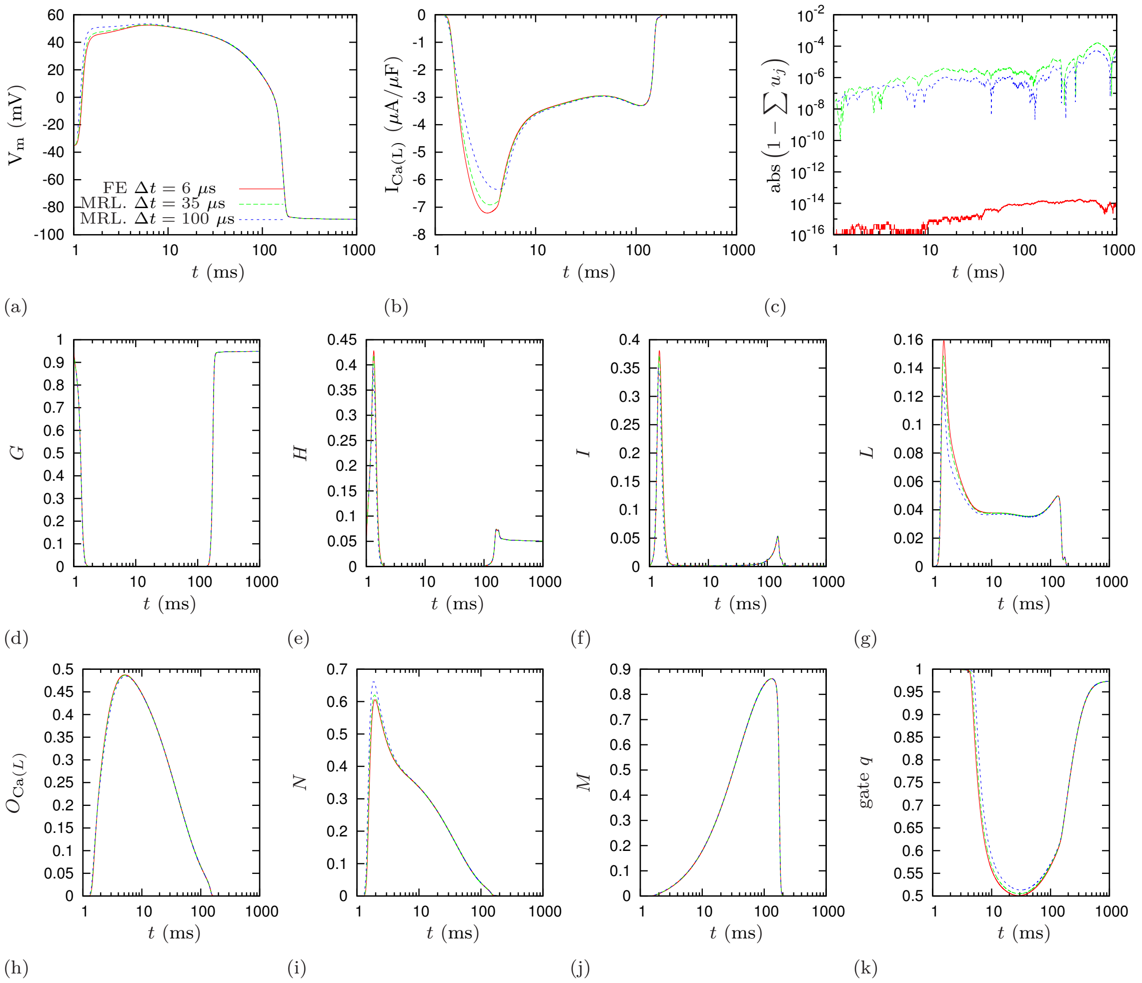
$$\vec{u}_{n+1} = \exp\left(\boldsymbol{D}\Delta t\right)\vec{u}_n. \tag{5.35b}$$

The open probability is obtained as a product of the open state $O_{\mathrm{Ca}(L)}$ and the gate q as $P_{\mathrm{open}} = O_{\mathrm{Ca}(L)}q$.

Figure 5.8 shows results of the integration of the whole cell model using the suggested methods for RyR and $I_{\mathrm{Ca}(L)}$ channels. The states of the model visually overlap except in panels (g), (i), and (k).

5.2.4 Conclusions for RyR and $I_{\mathrm{Ca}(L)}$ Case Study

The MRL method for $I_{\mathrm{Ca}(L)}$ combined with hybrid method for RyR allow larger time steps (up to $190\ \mu\mathrm{s}$). The causes of the instability at larger time step sizes are the intracellular sodium $[\mathrm{Na}^+]_i$ and potassium $[\mathrm{K}^+]_i$ concentrations that are calculated using the forward Euler method (Figure 5.9). Because those variables are coupled with other dynamical variables of the cellular model (ion currents) and are non-linear, we cannot directly apply the MRL methods.



(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

Figure 5.8: Comparison of integrators for $I_{\mathrm{Ca}(L)}$ Markov chain model: (a) action potential; (b) $I_{\mathrm{Ca}(L)}$; (c) deviation from states conservation law; (d-j) states occupancy in $I_{\mathrm{Ca}(L)}$ MC model; (k) the gating variable q controlling transition between Mode-Ca and Mode-$\mathrm{V_m}$.

The computational cost is shown in Table 5.2. The simulation was performed for 100 beats with a cycle length of 1000 ms. During this computation the output of the variables was omitted.

The main benefit of applying the suggested methods is the possibility of increasing the time step size. This leads to a reduction of the computational cost without running the risk that the solution becomes unstable. Our results show it is possible to increase the time step from $\Delta t = 6\ \mu\mathrm{s}$, which was the limit for the stable solution in the forward Euler method, by using matrix Rush-Larsen methods for the RyR and $I_{\mathrm{Ca}(L)}$ models. This permits an increase in the time step size up to $\Delta t = 180\ \mu\mathrm{s}$ without the loss of stability.

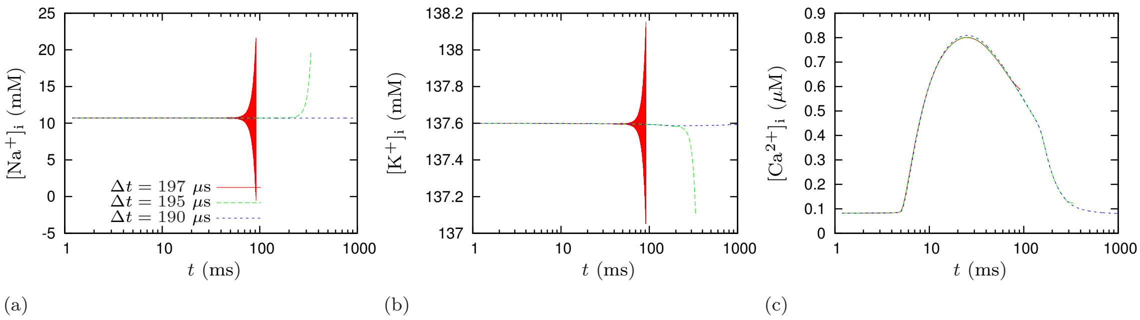


Figure 5.9: Numerical instability caused by computation of intracellular ionic concentrations of (a) sodium $[\mathrm{Na}^+]_i$; (b) potassium $[\mathrm{K}^+]_i$; and (c) calcium $[\mathrm{Ca}^{2+}]_i$

Table 5.2: Computational cost in forward Euler (FE), hybrid (hyb.) and matrix Rush-Larsen (MRL) methods during 100 beats with cycle length 1000 ms.

Δt [μs]		1	6	15	35	100	180
RyR	FE	22.44	3.90				
	hyb.	38.80	6.50	2.61	1.14	0.38	0.23
$I_{\mathrm{Ca}(L)}$	FE	68.17	11.33	4.51	1.90		
	MRL	80.35	13.39	5.42	2.24	0.72	0.47
total	FE	323.02	54.08				
	hyb.	337.64	57.21	22.69	9.83		
	MRL	354.86	59.14	23.60	10.13	3.56	2.00

5.3 Accuracy of Numerical Methods for Markov Chain

5.3.1 Order of Approximation

The solution of a Markov chain system (5.1) is approximated using fixed-point iterations [22] as

$$\vec{u}^{(\ell+1)}(t_{n+1}) = \vec{u}(t_n) + \int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau)\vec{u}^{(\ell)}(\tau)\mathrm{d}\tau. \tag{5.36}$$

which converges to the exact solution as $\ell \to \infty$. For the zeroth iteration ($\ell = 0$) we use the state of the system at the time point t_n as $\vec{u}^{(1)}(t_{n+1}) = \vec{u}(t_n)$ which gives the solution of the consequent iterations as

$$\vec{u}^{(2)}(t_{n+1}) = \vec{u}(t_n) + \left(\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau)\mathrm{d}\tau\right)\vec{u}(t_n) = \left(\mathbf{I} + \int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau)\mathrm{d}\tau\right)\vec{u}(t_n) \tag{5.37a}$$

$$\vec{u}^{(3)}(t_{n+1}) = \left(\mathbf{I} + \int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau)\mathrm{d}\tau + \int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau_1)\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau_2)\mathrm{d}\tau_2\mathrm{d}\tau_1\right)\vec{u}(t_n) \tag{5.37b}$$

and for the iteration #$(\ell+1)$ we can follow the scheme

$$\vec{u}^{(\ell+1)}(t_{n+1}) = \Bigg(\mathbf{I} + \int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau)\mathrm{d}\tau + \int_{t_n}^{t_n+\Delta t}\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau_1)\boldsymbol{A}(\tau_2)\mathrm{d}\tau_2\mathrm{d}\tau_1 + \\ + \ldots + \int_{t_n}^{t_n+\Delta t}\cdots\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau_1)\boldsymbol{A}(\tau_2)\cdots\boldsymbol{A}(\tau_\ell)\mathrm{d}\tau_\ell\ldots\mathrm{d}\tau_2\mathrm{d}\tau_1\Bigg)\vec{u}(t_n). \tag{5.38}$$

The precision increases with larger ℓ such that each consecutive iteration gives one order of better approximation. The accuracy of the solution for ℓ iteration is then $\mathcal{O}(\Delta t^\ell)$.

To find the solution accurate up to the order $\mathcal{O}(\Delta t^2)$ we use the formula (5.37b). First we use Taylor expansion to rewrite the matrix $\boldsymbol{A}(\tau)$ around a point $\tau = t_n$ as

$$\boldsymbol{A}(\tau) = \boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\tau + \mathcal{O}(\tau^2), \tag{5.39}$$

where $\boldsymbol{A}_n = \boldsymbol{A}(t_n)$ and $\mathrm{d}\boldsymbol{A}_n/\mathrm{d}t$ is derivative of $\boldsymbol{A}(t)$ evaluated at $t = tSt_n$.

To find the result we need to integrate

$$\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau)\mathrm{d}\tau = \int_{t_n}^{t_n+\Delta t} \left(\boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\tau + \mathcal{O}(\tau^2)\right)\mathrm{d}\tau = \\ = \boldsymbol{A}_n\Delta t + \frac{1}{2}\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t^2 + \mathcal{O}(\Delta t^3) \tag{5.40}$$

and

$$\int_{t_n}^{t_n+\Delta t}\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(\tau_1)\boldsymbol{A}(\tau_2)\mathrm{d}\tau_2\mathrm{d}\tau_1 = \\ = \int_{t_n}^{t_n+\Delta t}\int_{t_n}^{t_n+\Delta t} \left(\boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\tau_1\right)\left(\boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\tau_2\right)\mathrm{d}\tau_2\mathrm{d}\tau_1 + \mathcal{O}(\Delta t^4) = \\ = \int_{t_n}^{t_n+\Delta t}\int_{t_n}^{t_n+\Delta t} \left[\boldsymbol{A}_n^2 + \boldsymbol{A}_n\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\tau_2 + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\boldsymbol{A}_n\tau_1 + \left(\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\right)^2\tau_2\tau_1\right]\mathrm{d}\tau_2\mathrm{d}\tau_1 + \\ + \mathcal{O}(\Delta t^4) = \frac{1}{2}\boldsymbol{A}_n^2\Delta t^2 + \mathcal{O}(\Delta t^3).$$

Then equation (5.37b) rewrites as

$$\vec{u}^{(3)}(t_{n+1}) = \left[\mathbf{I} + \boldsymbol{A}_n\Delta t + \frac{1}{2}\left(\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t} + \boldsymbol{A}_n^2\right)\Delta t^2\right]\vec{u}(t_n) + \mathcal{O}(\Delta t^3), \tag{5.41}$$

which is used for the estimation of the the order of accuracy of forward Euler and MRL methods. The developed methods are compared to the solution (5.41), as for the purposes of assessing the accuracy of the methods, this equation provide a sufficient number of terms.

5.3.2 Forward Euler method

The solution of the system (5.1) according to the forward Euler method is

$$u_{\mathrm{FE},n+1} = (\mathbf{I} + \boldsymbol{A}_n \Delta t) u(t_n). \tag{5.42}$$

Comparing this value with (5.41) we obtain the local truncation error of forward Euler as

$$E_{\mathrm{FE}} = \|u(t_{n+1}) - u_{\mathrm{FE},n+1}\| = \frac{1}{2}\Delta t^2 \left(\left\|\boldsymbol{A}_n^2\right\| + \left|\frac{\mathrm{dV_m}}{\mathrm{d}t}\right| \left\|\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{dV_m}}\right\| \right) + \mathcal{O}(\Delta t^3) \tag{5.43}$$

where $\|\cdot\|$ denotes the norm of a matrix.

5.3.3 Matrix Rush-Larsen

In order to find the truncation error of the Matrix Rush-Larsen method, we expand (5.23) using Taylor expansion as

$$u_{\mathrm{MRL},n+1} = \left[\mathbf{I} + \boldsymbol{A}_n \Delta t + \frac{1}{2}\boldsymbol{A}_n^2 \Delta t^2\right] u(t_n) + \mathcal{O}(\Delta t^3). \tag{5.44}$$

Comparing this result with (5.41) we find the truncation error

$$E_{\mathrm{MRL}} = \|u(t_{n+1}) - u_{\mathrm{MRL},n+1}\| = \frac{1}{2}\left\|\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\right\| \Delta t^2 + \mathcal{O}(\Delta t^3). \tag{5.45}$$

If $\boldsymbol{A}_n = \boldsymbol{A}(\mathrm{V_m}(t_n))$, the chain rule for the derivation of $\mathrm{dV_m}/\mathrm{d}t$ allows us to rewrite this expression in the form

$$E_{\mathrm{MRL}} = \frac{1}{2}\Delta t^2 \left(\left\|\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{dV_m}}\right\| \left|\frac{\mathrm{dV_m}}{\mathrm{d}t}\right| \right) + \mathcal{O}(\Delta t^3). \tag{5.46}$$

So, the difference in error of forward Euler from the MRL method is

$$E_{\mathrm{FE}} - E_{\mathrm{MRL}} = \frac{1}{2}\Delta t^2 \left\|\boldsymbol{A}_n^2\right\| + \mathcal{O}(\Delta t^3). \tag{5.47}$$

5.3.4 Operator Splitting for Linear Systems

The hybrid methods use the operator splitting technique for the transition matrix of the Markov chain. The simplest method of operator splitting is called Lie splitting. In this method the transition matrix is split as

$$\boldsymbol{A} = \boldsymbol{A}_1 + \boldsymbol{A}_2 + \ldots + \boldsymbol{A}_k, \tag{5.48}$$

where we use the assumption that the $\boldsymbol{A}$ can be approximated by a constant for the duration of the time step.

Then the system (5.1) becomes

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}\vec{u} = (\boldsymbol{A}_1 + \boldsymbol{A}_2 + \ldots + \boldsymbol{A}_k)\,\vec{u}. \tag{5.49}$$

From the solution of this system we derive the following iterative scheme

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t + \boldsymbol{A}_2\Delta t + \ldots + \boldsymbol{A}_k\Delta t\right)\vec{u}_n. \tag{5.50}$$

However, sometimes it is convenient to use an approximation to the solution in substeps as

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right)\ldots\exp\left(\boldsymbol{A}_k\Delta t\right)\vec{u}_n, \tag{5.51}$$

which rewrites as

$$\vec{u}_{n+1/k} = \exp\left(\boldsymbol{A}_1\Delta t\right)\vec{u}_n, \tag{5.52a}$$

$$\vec{u}_{n+2/k} = \exp\left(\boldsymbol{A}_2\Delta t\right)\vec{u}_{n+1/k}, \tag{5.52b}$$

$$\vdots$$

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_k\Delta t\right)\vec{u}_{n+(k-1)/k}. \tag{5.52c}$$

This approximate method called Lie splitting allows us to substitute specific steps by an alternative methods, if the solution obtained using equation (5.50) is too costly. Hence, we call the approximate method in (5.51) the hybrid method.

5.3.5 Truncation Error of Lie Splitting

Double Splitting

Here we analyse the truncation error for double splitting in Lie splitting, i.e. when $k = 2$ in (5.48). This section is inspired by [23]. The exponential in the exact solution in this case is

$$\begin{aligned}\exp\Big((\boldsymbol{A}_1 + \boldsymbol{A}_2)\Delta t\Big) &= \mathbf{I} + \Delta t\,(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}\Big(\boldsymbol{A}_1^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_2^2\Big) + \\ &+ \frac{\Delta t^3}{6}(\boldsymbol{A}_1^3 + \boldsymbol{A}_1^2\boldsymbol{A}_2 + \boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_1\boldsymbol{A}_2^2 + \boldsymbol{A}_2\boldsymbol{A}_1^2 + \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2\boldsymbol{A}_1 + \boldsymbol{A}_2^3) + \\ &+ \mathcal{O}(\Delta t^4).\end{aligned} \tag{5.53}$$

The operator splitting method rewrites as

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right)\vec{u}_n. \tag{5.54}$$

Each exponential expands according to

$$\exp(\boldsymbol{A}_j\Delta t) = 1 + \Delta t\boldsymbol{A}_j + \frac{\Delta t^2}{2}\boldsymbol{A}_j^2 + \mathcal{O}(\Delta t^3), \tag{5.55}$$

and we multiply two exponentials for $j = 0, 1$ to get

$$\begin{aligned}\left(\mathbf{I} + \Delta t\boldsymbol{A}_1 + \frac{\Delta t^2}{2}\boldsymbol{A}_1^2\right)\left(\mathbf{I} + \Delta t\boldsymbol{A}_2 + \frac{\Delta t^2}{2}\boldsymbol{A}_2^2\right) + \mathcal{O}(\Delta t^3) =\\ =\mathbf{I} + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}(\boldsymbol{A}_1^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2) + \mathcal{O}(\Delta t^3)\end{aligned} \tag{5.56}$$

the local truncation error is found after subtracting the (5.56) from (5.53), which by orders of Δt gives

$$\mathcal{O}(\Delta t^0) :\mathbf{I} - \mathbf{I} = \mathbf{0}, \tag{5.57a}$$

$$\mathcal{O}(\Delta t^1) :(\boldsymbol{A}_1 + \boldsymbol{A}_2) - (\boldsymbol{A}_1 + \boldsymbol{A}_2) = 0, \tag{5.57b}$$

$$\begin{aligned}\mathcal{O}(\Delta t^2) :&\frac{1}{2}\left((\boldsymbol{A}_1^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_2^2) - (\boldsymbol{A}_1^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2)\right) =\\ &= \frac{1}{2}(\boldsymbol{A}_2\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_2).\end{aligned} \tag{5.57c}$$

So, that the error of double splitting is

$$E_{\mathrm{OS},2} = \frac{1}{2}\Delta t^2 \left\|[\boldsymbol{A}_2, \boldsymbol{A}_1]\right\| + \mathcal{O}(\Delta t^3) \tag{5.58}$$

where we define the commutator $[\boldsymbol{A}_2, \boldsymbol{A}_1] = \boldsymbol{A}_2\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_2$, which will become more useful in the case of triple splitting. We notice, that the commutator is small as long as the terms of either of the two matrices are small.

Triple Splitting

Here we analyse the truncation error for triple splitting in Lie splitting, i.e. when $k = 3$ in (5.48). The exponential in the exact solution in this case is

$$\begin{aligned}&\exp\left((\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3)\Delta t\right) = \mathbf{I} + \Delta t\,(\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) +\\ &+ \frac{\Delta t^2}{2}\left(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2 + \boldsymbol{A}_3^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_1\boldsymbol{A}_3 + \boldsymbol{A}_3\boldsymbol{A}_1 + \boldsymbol{A}_2\boldsymbol{A}_3 + \boldsymbol{A}_3\boldsymbol{A}_2\right)\\ &+ \mathcal{O}(\Delta t^3).\end{aligned} \tag{5.59}$$

The operator splitting method rewrites as

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right)\exp\left(\boldsymbol{A}_3\Delta t\right)\vec{u}_n \tag{5.60}$$

we multiply all three terms expanded as (5.55) and using (5.56) to get

$$\left(\mathbf{I} + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}(\boldsymbol{A}_1^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2)\right)\left(\mathbf{I} + \Delta t\boldsymbol{A}_3 + \frac{\Delta t^2}{2}\boldsymbol{A}_3^2\right) + \mathcal{O}(\Delta t^3) =$$
$$= \mathbf{I} + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) + \Delta t^2(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2 + \boldsymbol{A}_3^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + 2\boldsymbol{A}_1\boldsymbol{A}_3 + 2\boldsymbol{A}_2\boldsymbol{A}_3) \tag{5.61}$$

And the local truncation error is then

$$\mathcal{O}(\Delta t^0) : \mathbf{I} - \mathbf{I} = \mathbf{0} \tag{5.62}$$
$$\mathcal{O}(\Delta t^1) : (\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) - (\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) = 0 \tag{5.63}$$
$$\begin{aligned}\mathcal{O}(\Delta t^2) : &\frac{1}{2}\Big[(\boldsymbol{A}_1^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_1\boldsymbol{A}_3 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_2^2 + \boldsymbol{A}_2\boldsymbol{A}_3 + \boldsymbol{A}_1\boldsymbol{A}_3 + \boldsymbol{A}_2\boldsymbol{A}_3 + \boldsymbol{A}_3^2) \\ &\quad -(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2 + \boldsymbol{A}_3^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + 2\boldsymbol{A}_1\boldsymbol{A}_3 + 2\boldsymbol{A}_2\boldsymbol{A}_3)\Big] = \\ &\quad = \frac{1}{2}[(\boldsymbol{A}_2\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_2) + (\boldsymbol{A}_3\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_3) + (\boldsymbol{A}_3\boldsymbol{A}_2 - \boldsymbol{A}_2\boldsymbol{A}_3)].\end{aligned} \tag{5.64}$$

So, we find the error of triple splitting as

$$E_{\mathrm{OS},3} = \frac{1}{2}\Delta t^2\left(\|[\boldsymbol{A}_2, \boldsymbol{A}_1] + [\boldsymbol{A}_3, \boldsymbol{A}_1] + [\boldsymbol{A}_3, \boldsymbol{A}_2]\|\right) + \mathcal{O}(\Delta t^3). \tag{5.65}$$

5.4 Conclusions

This chapter described the application of exponential integration to Markov chain models. Exponential integrators are able to increase the accuracy of the solution and prevent numerical instabilities which appear in the forward Euler model at relatively small step sizes due to the stiffness inherent to Markov chain models. The solution using exponential solvers then allows larger time steps without the loss of stability, leaving the accuracy considerations to be the only factor limiting the time step size.

We have also estimated the order of accuracy of the forward Euler and MRL methods, which showed first-order accuracy of both methods. The comparison of the error estimate revealed that MRL is more accurate then the forward Euler. However, the main advantage of the MRL method is the possibility of increasing the time step size without the instability issues. This allows to further improve the accuracy by developing higher order accuracy methods, for example an extension of Perego-Veneziani (2009) [24], and Sundnes *et al.* (2009) [25] methods to Markov chains, or conversion of higher-order methods (e.g. Runge-Kutta) into an exponential scheme.

The implementation of the MRL as presented in this chapter remains a largely manual procedure which requires a number of steps to be taken care of. To simplify this process, we have implemented the methods into the cardiac simulation package $\mathrm{BeatBox}$ as described in the following chapter.

Chapter 6

Exponential Solvers for Markov Chain Models in BeatBox

In the previous chapter we have developed exponential integration methods, which address the instability issues and provide more efficient solution for Markov chain ion channel. In order to avoid separate implementation of the solver for each ion channel, we have implemented the exponential integration methods into a cardiac simulation package BeatBox. This chapter describes the usage of those methods as well as the format of specific C-functions.

BeatBox is aimed for simulations of the heart. Mathematically we can describe the heart by means of equations characterising the reactions in individual cells, combined with the characteristics of the diffusion through the cellular tissue. Hence, this system is called reaction-diffusion system. Section 6.1 gives a formal definition of the reaction system and its division into subsystems according to the form of ordinary differential equations (ODEs) to the extent necessary to understand the description of cellular models in BeatBox. Section 6.2 describes numerical methods used for solving of the cellular modules in BeatBox.

Section 6.3 gives a brief overview of the usage of the BeatBox package. A detailed description can be found in the BeatBox documentation [26] and in the McFarlane's (2010) thesis [27]. Section 6.4 describes the implementation of the reaction system into BeatBox in two different ways – as so-called `rhs` (right-hand side) modules and `ionic` modules. Section 6.5 describes the procedures within the `euler` and the `rushlarsen` devices used for the solution of `rhs` and `ionic` modules respectively. Section 6.6 specifies the format of structures in the `ionic` modules as used by so-called `rushlarsen` "device" by the matrix Rush-Larsen method (a definition of "device" can be found on page 153). Section 6.7 shows some of the results of the simulations using the extended version of BeatBox.

The path in the filenames in this section is given relative to the BeatBox repository.

In order to be able to describe the implementation we need to describe the pre-existing state of BeatBox. This text provides the minimal context sufficient to explain the new contributions. The modifications of the old code and newly created parts of BeatBox package are marked with a $\boldsymbol{*}$ sign in the text of this chapter.

6.1 Definition of a Reaction System

The cardiac cells are described by a systems of ODEs in a form

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \vec{f}(\vec{u}), \tag{6.1}$$

where $\vec{f}(\vec{u})$ is a function that describes the kinetics of the system. The vector $\vec{u}$ contains the dynamical variables of the system.

The dynamical variables can be divided according to their role and form as

$$\vec{u} = \left[\vec{v}, \vec{y}, \vec{z}\right],$$

where $\vec{v}$ represents Hodgkin-Huxley type gating variables of ion channels, $\vec{y}$ represents non-gating “other” variables such as membrane voltage $\mathrm{V_m}$, ionic concentrations etc., and $\vec{z}$ represents Markov chain variables describing the ion channels.

We divide the Hodgkin-Huxley type gating variables $\vec{v}$ into two groups according to the dynamical variable which the gates depend on, as

$$\vec{v} = \left[\vec{w}, \vec{x}\right],$$

where $\vec{w}$ represent gates dependent on the membrane voltage $\mathrm{V_m}$, and $\vec{x}$ represent gates dependent on “other” variables $\vec{y}$ except voltage.

If the cellular model has more than one Markov chain, the variable $\vec{z}$ is composed of the individual Markov chain variables as

$$\vec{z} = \left[\vec{z}^1, \vec{z}^2, \ldots, \vec{z}^{N_m}\right],$$

where each $\vec{z}^k$ corresponds to one ion channel.

The ODEs of the whole reaction system (6.1) is reformulated in terms of specific type of equations for each of the groups as

$$\frac{\mathrm{d}w^i}{\mathrm{d}t} = \alpha^i(\mathrm{V_m})(1 - w^i) - \beta^i(\mathrm{V_m})w^i, \tag{6.2a}$$

$$\frac{\mathrm{d}x^j}{\mathrm{d}t} = \alpha^j(\vec{y})(1 - x^j) - \beta^j(\vec{y})x^j, \tag{6.2b}$$

$$\frac{\mathrm{d}\vec{y}}{\mathrm{d}t} = \vec{\phi}(\vec{w}, \vec{x}, \vec{y}, \vec{z}), \tag{6.2c}$$

$$\frac{\mathrm{d}\vec{z^k}}{\mathrm{d}t} = \boldsymbol{A}^k(\vec{y})\vec{z^k}, \tag{6.2d}$$

where α and β represent opening and closing transition rates of Hodgkin-Huxley type gating variables respectively. We denote the gates dependent on voltage by a superscript $i = 1, 2, \ldots, N_t$ and gates dependent on other variables by a superscript $j = 1, 2, \ldots, N_n$. Matrices $\boldsymbol{A}^k(\vec{y})$ contain transition rates of the Markov chain $k = 1, 2, \ldots, N_m$. The function $\vec{\phi}$ represents the kinetics of the non-gating "other" dynamical variables.

The methods for solving each of the individual groups of the system (6.2) were described in different parts of the thesis. For the readers' convenience we present the formulas once more in the following section.

6.2 Solution of Reaction System

6.2.1 Tabulation

The cellular models involve functions in a form $\kappa(r(t))$ dependent on a dynamical variable r. An example of this type of functions are the transition rates of gating variables $\alpha(\mathrm{V_m}(t))$ and $\beta(\mathrm{V_m}(t))$. The value of these functions have to be found multiple times during the simulation. In many cases, these computations are done for the same, or a very similar value of the input variable r.

To avoid such computationally expensive calculations at each time step we use a process known as tabulation. This means that we prepare a look-up table of functional values of $\kappa(r)$ for a grid of the "control" variable r before starting the simulation. During the simulation we fetch an approximated functional value from a specific entry in the look-up table as will be described below.

Before discussing the tabulation in more detail, we choose a scaling function, which maps the control variable on the tabulation scale $\mathcal{T} = \psi(r)$, for example an identity function in the simplest case, or a more complicated logarithmic function $\psi = \ln$. The tabulation is done for a regular grid defined as $\mathcal{T}_j = \mathcal{T}_{\min} + j\Delta\mathcal{T}$ for table entries $j = 1, 2, \ldots, k$. Constant $\mathcal{T}_{\min}$ is a lower limit of tabulation. The tables are filled up with the corresponding values of functions $\kappa_j = \kappa(\mathcal{T}_j)$.

Assuming a sufficient detail of the grid and the continuity of the tabulated function, we approximate the functional value at time step number n by the value in a table entry with index $\xi(n)$ as $\kappa(t_n) \approx \kappa^{\xi(n)}$ where the index is found as

$$\xi(n) = \mathrm{round}\left(\frac{\mathcal{T}(t_n) - \mathcal{T}_{\min}}{\Delta\mathcal{T}}\right).$$

In the simplest case the scaling function is an identity function $r = \mathcal{T}$ and the grid of the control variable is linear. This is used in the case of voltage-dependent functions $\kappa(\mathrm{V_m}(t))$ where the control variable corresponds to the tabulation variable $\mathcal{T} = r = \mathrm{V_m}$.

If the control variable r is an ionic concentration it is sometimes more convenient to use a tabulation on a logarithmic grid. This means that the scaling function is a logarithm and the tabulation variable $\mathcal{T} = \ln(r)$.

Minimal and maximal limits of tabulation on the logarithmic scale are determined as $\mathcal{T}_{\min} = \ln(r_{\min})$ and $\mathcal{T}_{\max} = \ln(r_{\max})$. The appropriate table increment $\Delta\mathcal{T}$ given on the logarithmic grid is found from desired number k of entries in the tableas

$$\Delta\mathcal{T} = \frac{1}{k}\ln\left(\frac{r_{\max}}{r_{\min}}\right),$$

and the table index is obtained as

$$\xi(n) = \mathrm{round}\left(\frac{\ln(r(t_n)) - \ln(r_{\min})}{\Delta\mathcal{T}}\right) = \mathrm{round}\left[\frac{1}{\Delta\mathcal{T}}\ln\left(\frac{r(t_n)}{r_{\min}}\right)\right].$$

The entries of the transition matrix of a Markov chain are functions of dynamical variables. Normally, they depend on a single variable, in which case the tabulation described above is a straightforward task. However, some transition rate matrices depend on multiple variables, for example voltage and calcium concentration as in $\boldsymbol{A}^k(\mathrm{V_m}, [\mathrm{Ca}^{2+}]_i)$. The tabulation of such transition rates matrices as a whole is possible, but expensive on computational time and memory resources, which are required to fill multi-dimensional tables (in our example two dimensional).

An alternative approach is to use an operator splitting method and split the transition matrix into a sum of a number of “submatrices” depending on a single control variable. If, in our example, the transition matrix has a sum form as

$$\boldsymbol{A}^k(\mathrm{V_m}, [\mathrm{Ca}^{2+}]_i) = \boldsymbol{A}_1^k(\mathrm{V_m}) + \boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_i) \tag{6.3}$$

then the “submatrices” $\boldsymbol{A}_1^k(\mathrm{V_m})$ and $\boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_i)$ can be tabulated independently. This way we get the voltage-dependent “submatrices” as $\boldsymbol{A}_1^k(\mathrm{V_m}^{\xi(n)}) \approx \boldsymbol{A}_1^k(\mathrm{V_m}(t_n))$ on a linear grid of voltage $\mathrm{V_m}$, and $\boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_i^{\xi(n)}) \approx \boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_i(t_n))$ on a logarithmic grid of calcium concentration $[\mathrm{Ca}^{2+}]_i$.

The operator splitting method is not universal, because it is not guaranteed that the transition matrix can be split into a sum of “submatrices” dependent on a single control variable, for a general case, however, it could be done in the cases we have considered. The operator splitting technique for Markov chain model was described in Section 5.3.4.

6.2.2 Forward Euler Method

The simplest time stepping method is known as forward Euler. The forward Euler method is based on the time discretisation $\Delta t = t_{n+1} - t_n$. The forward Euler method for the system (6.1) reads as

$$\vec{u}_{n+1} = \vec{u}_n + \Delta t \vec{f}(\vec{u}_n) \tag{6.4}$$

where $\vec{u}_{n+1} \approx \vec{u}(t_{n+1})$ is a numerical solution at time points $t_n = t_0 + n\Delta t$ with a small time step Δt. The iterations start from a given initial state of the system $\vec{u}(t_0)$.

The forward Euler method can be applied to all of the groups of the divided system described by equations (6.2). The particular time-stepping scheme is obtained by substituting the $\vec{f}(\vec{u})$ in equation (6.2) with the right-hand side of the corresponding equations for each of the groups as

$$w^i_{n+1} = w^i_n + \Delta t \left[\alpha^i({\mathrm{V_m}}^{\xi(n)})(1 - w^i_n) - \beta^i({\mathrm{V_m}}^{\xi(n)}) w^i_n\right], \tag{6.5a}$$

$$x^j_{n+1} = x^j_n + \Delta t \left[\alpha^j(\vec{y}_n)(1 - x^j_n) - \beta^j(\vec{y}_n) x^j_n\right], \tag{6.5b}$$

$$\vec{y}_{n+1} = \vec{y}_n + \Delta t \left[\vec{\phi}(\vec{w}_n, \vec{x}_n, \vec{y}_n, \vec{z}_n)\right], \tag{6.5c}$$

$$\vec{z}^k_{n+1} = \vec{z}^k_n + \Delta t \left[\boldsymbol{A}^k(\vec{y}_n)\vec{z}^k_n\right]. \tag{6.5d}$$

The accuracy of the forward Euler method is of $\mathcal{O}(\Delta t)$. As the $\Delta t \to 0$ the solution converges to an exact solution. The cost of convergence is the computational time required for the solution, so a trade-off between the accuracy and the speed of the computations has to be found.

6.2.3 Exponential Integration for Hodgkin-Huxley Type Gates

Exponential integration methods for the Hodgkin-Huxley type gate model were proposed by Rush-Larsen [17]. Their scheme assumes that the transition rates α and β vary slowly and can be “frozen” during one time step, i.e. assumed to be a constant. This approximate system system can be solved analytically. The analytic solution of the equation was described in detail in Section 3.2.3, with the final result in equation (3.131).

This result can be used to deduce an iterative numerical method, here presented for gating variable v^i which corresponds to any of the Hodgkin-Huxley gates. Recall that $\vec{v} = [\vec{x}, \vec{w}]$, and the kinetics of the system are described by the equations (6.2b) and (6.2a) respectively. The exponential integration scheme then reads as

$$v^i_{n+1} = \frac{\alpha^i(\vec{y}_n)}{\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n)} - \left[\frac{\alpha^i(\vec{y}_n)}{\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n)} - v^i_n\right]\exp\Big[-(\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n))\Delta t\Big], \tag{6.6}$$

where $\alpha^i(\vec{y}_n)$ and $\beta^i(\vec{y}_n)$ are the values of the transition rates “frozen”, i.e. considered constant at their values at the beginning of the time step. We can include the exponentials into the definition of the coefficients as

$$a(\vec{y}_n, \Delta t) = \frac{\alpha^i(\vec{y}_n)}{\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n)}\Big(1-\exp\Big[-(\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n))\Delta t\Big]\Big), \tag{6.7a}$$

$$b(\vec{y}_n, \Delta t) = \exp\Big[-(\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n))\Delta t\Big]. \tag{6.7b}$$

The integration method for the two groups of gating variables (6.2b), (6.2a) is

$$w^i_{n+1} = a^i({\mathrm{V}_\mathrm{m}}^{\xi(n)}, \Delta t) - b^i({\mathrm{V}_\mathrm{m}}^{\xi(n)}, \Delta t)w^i_n, \tag{6.8a}$$

$$x^j_{n+1} = a^j(\vec{y}_n, \Delta t) - b^j(\vec{y}_n, \Delta t)x^i_n, \tag{6.8b}$$

where the a^i and b^i for gates w^i are tabulated, and a^j and b^j for gates x^j are computed on-the-fly, i.e. during the simulations.

6.2.4 Exponential Integration for Markov Chains

We have suggested and described an exponential integration scheme for Markov chains called matrix Rush-Larsen (MRL) in the section 5.3.3 (final equation (5.24)). Briefly, we assume that the transition matrix of the Markov chains can be “frozen” for the duration of one time step. This yields a linear system of ODEs with constant coefficients which can be solved analytically. The analytic solution can be used to develop an iterative integration scheme. This scheme for our system $\boldsymbol{A}^k(\vec{y}_n)$ as defined in equations (6.2d) reads as

$$\vec{z}^k_{n+1} = \exp\Big(\boldsymbol{A}^k(\vec{y}_n)\Delta t\Big)\,\vec{z}^k_n. \tag{6.9}$$

The exponential operator matrix can be transformed to

$$\boldsymbol{T}^k(\vec{y}_n, \Delta t) = \exp\Big(\boldsymbol{A}^k(\vec{y}_n)\Delta t\Big) = \boldsymbol{V}^k(\vec{y}_n)\exp\Big[\boldsymbol{\Lambda}^k(\vec{y}_n)\Delta t\Big]\,\boldsymbol{W}^{\boldsymbol{T}^k}(\vec{y}_n), \tag{6.10}$$

where $\boldsymbol{A}^k = \boldsymbol{V}^k\boldsymbol{\Lambda}^k\boldsymbol{W}^{\boldsymbol{T}^k}$ is an eigenvalue decomposition of the transition matrix. The $\boldsymbol{\Lambda}^k$ is a diagonal eigenvalue matrix (with the eigenvalues on the diagonal). The $\boldsymbol{V}^k$ is the right eigenvector matrix (the columns contain the corresponding right eigenvectors in the same order as the eigenvalue matrix) and $\boldsymbol{W}^k$ is the left eigenvector matrix (the columns contain corresponding left eigenvectors). The eigenvectors are scaled to give an identity matrix $\boldsymbol{I} = \boldsymbol{V}^k \boldsymbol{W^T}^k$.

For the transition rates matrices dependent on a multiple variables such as in the formula (6.3), the iterative scheme uses operator splitting method as

$$\vec{z}^{k}_{n+1/2} = \exp\left(\boldsymbol{A}_1^k ({\mathrm{V}_\mathrm{m}}^{\xi(n)}) \Delta t\right) \vec{z}^{k}_{n}, \tag{6.11a}$$

$$\vec{z}^{k}_{n+1} = \exp\left(\boldsymbol{A}_2^k ({[\mathrm{Ca}^{2+}]_i}^{\xi(n)}) \Delta t\right) \vec{z}^{k}_{n+1/2}. \tag{6.11b}$$

Before the simulation starts we create look-up tables for the eigenvalues and eigenvector matrices of $\boldsymbol{A}_1^k$ and $\boldsymbol{A}_2^k$. The tables are then used in the numerical scheme similarly to (6.10).

If the tabulation is unusable due to the complicated dependence on multiple variables, as discussed before, we can still do the eigenvalue decomposition on-the-fly. This might be beneficial when the system is stiff and the forward Euler method leads to numerical instability, because the possible increase in time step size might outweigh the cost of eigenvalue decomposition at each time step.

6.3 Running BeatBox Simulation

BeatBox is a simulation environment for biophysically and anatomically realistic simulations of reaction-diffusion systems such as cardiac tissue. BeatBox is implemented in C language and the source code is freely available [28]. The distribution of the package is allowed under the conditions of GNU GPLv2 license, which guarantee the freedom to share and adapt the software.

BeatBox combines numerical methods and models to simulate cardiac excitation in a single cell and as its' spatial propagation. The spatial models include regular 1D thread, 2D sheet, and 3D box or even anatomically detailed geometry provided from experimental data.

The design of BeatBox follows a modular paradigm which allows the construction of a simulation protocol according to specific requirements. The numerical methods for a specific purpose are implemented within modules called "devices". For instance a device can serve for a numerical integration of a reaction system, a computation of the diffusion within the tissue, or a data output of a particular dynamical variable into a text file.

The BeatBox simulation is launched from a command line interface by calling the BeatBox executable. The BeatBox executable requires a command line argument `<bbs-script>` which is a plain text `bbs` script file. The `bbs` script is a configuration script in a specific format. BeatBox reads the `bbs` script and constructs the simulation according to the commands and specific calls of the devices.

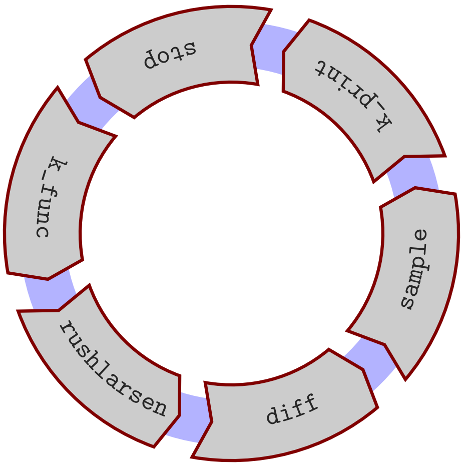


Figure 6.1: Conceptual "ring of devices". The ring is constructed from the instructions provided in the bbs script. Each revolution of the ring corresponds to one time step.

BeatBox processes the bbs script to set up a "ring of devices", i.e. a conceptual arrangement of the devices as shown in Figure 6.1. The devices in the ring are executed one by one in a loop. Each revolution of the ring corresponds to one time step in the simulation. The same device can be present in the ring several times. The criteria for the execution are set in the bbs script independently for each instance of a device.

The execution of a device can also be conditional to a particular spatial position of the cell within the tissue, and specific time of the simulation.

Section 6.5 specifies the parameters of the two most commonly used devices for the computation of reaction system – euler and rushlarsen. A scripting guide describing how the bbs script is processed and including details about generic device parameters can be found in the BeatBox documentation.

6.4 Definition of Reaction System in BeatBox

BeatBox offers a framework to define the reaction system of cellular models in two different ways – as rhs or ionic modules. Those modules are not to be confused with BeatBox devices because they do not implement numerical methods for the simulations. Instead they provide the description of the characteristics of the reaction system in C-functions. The input and output of the C-functions implemented within the rhs and ionic modules are understood by the devices intended for the solution of the reaction system.

Beside the C-functions describing the characteristics of the reaction system, the rhs and ionic modules implement a C-function for initialisation, which ensures that all data structures used during the integration are defined properly (for correct sizes of arrays etc.).

6.4.1 `rhs` Modules

The `rhs` (right-hand side) modules implement the right-hand side of the system described by equation (6.1), where the kinetics of all dynamical equations are within one C-function, which returns an increment of all the variables. So, the `rhs` modules do not provide any specification of the type of variables.

The `rhs` format is quite generic and straightforward to implement. However, the `rhs` format lacks information about the role of the variables. Therefore, it does not allow the possibility of exploring "hybrid" integration methods which treat the variables by a different scheme according to their role and a form. This limitation is addressed in `ionic` modules described in the following subsection.

For illustration a minimal working example of a `rhs` module for the Hodgkin-Huxley squid axon is listed in the Appendix section D.3.1.

6.4.2 `ionic` Modules

An alternative approach for defining the reaction system of cellular models is to divide the whole system into subsystems according to specific properties and a form of the equations describing the system.

The division which is used by `ionic` modules, corresponds to the grouping as described in (6.2), i.e. tabulated $\vec{w}$ and non-tabulated $\vec{x}$ gating variables, "other" variables $\vec{y}$, and Markov chains $\vec{z}$. To form an `ionic` module we need to implement a number of C-functions, whose formal arguments are understood by an appropriate integration device for `ionic` modules.

The non-gating "other" variables are implemented in a similar form to `rhs` modules as right-hand sides of the subsystem (6.2c) in one C-function. The same C-function also updates the values of the transition rates α^x and β^x of gates dependent on "other" variables except voltage (6.2b).

The voltage-dependent transition rates of the gating variables specified by equation (6.2a) are implemented in another C-function within the `ionic` module. This can also include other voltage-dependent functions within the cellular model. This C-function is used before the integration starts to tabulate the values of the voltage-dependent functions for a grid of membrane voltage as described in Section 6.2.1.

The `ionic` module does not have to implement formulas to obtain the time derivative of gating variables explicitly, instead they are implied by the standard form of the equations and known transition rates for a particular state of dynamical variables.

A minimal working example of the Hodgkin-Huxley squid axon model as `ionic` module is listed in the Appendix section D.3.2.

6.4.3 Extension of `ionic` Modules *

Previously, no specific C-functions to describe Markov chains were present in `ionic` modules. When the user wanted to employ Markov chains in the cell model, the formulas for the right-hand side of the time derivative had to be implemented within the C-function for “other” dynamical variables. Then the Markov chains were solved by explicit methods used within the integration device. However, explicit methods might suffer from numerical instabilities in stiff models.

We have implemented the exponential solver for Markov chains into BeatBox by modifying the format of `ionic` modules. Here we describe the details of the implementation.

The formal description of an ionic module contains tabulated $\vec{w}$ and non-tabulated $\vec{x}$ gating variables, “other” variables $\vec{y}$, and possibly a number of Markov chains $\vec{z}^k$. Each of these Markov chains in the `ionic` module (as specified by the equation (6.2d)) are split into a number of subchains according to the control variable on which the transition rates depend (as in operator splitting in equation (6.3)). The `ionic` module implements a separate C-function of each subchain to fill the entries of the corresponding transition sub-matrix. In most cases the Markov chain depends on a single control variable, and therefore such system contains only a single subchain.

A minimal working example of the Hodgkin-Huxley squid axon model with the ion channels converted into Markov chain description can be found in the Appendix Section D.3.3 .

6.5 Solution of Reaction System in BeatBox

The reaction system is defined by the `rhs` and the `ionic` modules. BeatBox can include a number of devices, to solve the modules by specific numerical methods. In this section we describe a `euler` device to solve `rhs` modules and a `rushlarsen` device to solve `ionic` modules. Both devices are described from the user perspective by describing the parameters of the devices which are called from the `bbs` script.

6.5.1 `euler` Device

Overview of the `euler` Device

The `euler` device is the simplest integration device for solving the `rhs` modules. Recall that the `rhs` modules implement formulas for the time derivative of all dynamical variables into one C-function which follows a certain format. This C-function is called during the time-stepping by the `euler` device. Another C-function is used for the initialisation of the device’s data structures with correct properties.

Table 6.1: Parameters to set up `euler` device in `bbs` script (adapted from [26] *rhs.h*).

type	name	description
`real`	`ht`	time step duration.
`str`	`ode`	name of cellular model in `rhs` format.
`int`	`rest`	number of steps to approximate initial conditions of the dynamical variables (resting state). When the value is set to zero (`0`), the default initial conditions from the module are used.
`codeblock`	`par`	model-dependent parameters of `rhs` module. The parameter is set by a codeblock in a form `par={<parameter>=<value> ...}` separated by blank spaces if more than one parameter is specified.

The simulation protocol is set up using `bbs` script. A summary of the relevant parameters that can be used in a call of the `euler` device is shown in Table 6.1.

Parameters of the `euler` Device

The parameter `ht` defines a duration of one time step used in calculations using the forward Euler. This corresponds to the Δt in the equation (6.4).

The parameter `ode` is a name for the `rhs` module that implements the cellular kinetics. This module is used for the simulation. The `ode` model has to follow the `rhs` format.

The parameter `rest` is used when the device determines the initial conditions of the dynamical variables. The device runs the simulation with the initial conditions specified in the module code for the number of steps specified in `rest`. This aims to approximate steady-state conditions.

Model dependent parameters of a `rhs` module are passed to the `euler` device through the `par` `bbs` script parameter. The assignment has to follow the format `{<parameter>=<value>}` (within curly brackets `{}` as shown). When multiple parameter are assigned they must be all specified within the same curly brackets and each assignment of them must be separated by a blank space (such as space, or newline).

Running `euler`

The `euler` device can be included in the "ring of devices" of BeatBox simulation. An example usage of the `euler` device can be found in the `bbs` script listed in Listing D.13. The parts relevant for the call of the `euler` device are:

Listing 6.1: Call of `euler` device with parameters within `bbs` script.

```
def int neqn 4; /* number of layers of state variables */
def real dt 0.01;           /* time step */
/* Reaction substep */
```

```
euler v0=0 v1=neqn-1 ht=dt ode=hh par={IV=@4;};
```

This bbs script contains several generic device parameters that specify the options for the execution of the device. More details on the generic device conditions can be found in the BeatBox documentation.

6.5.2 `rushlarsen` Device

Overview of the `rushlarsen` Device *

The `rushlarsen` device is an integration device for the `ionic` models. The. `ionic` models do not implement the formulas for the time derivative of all dynamical variables as `rhs`, instead they contain a number of C-functions for computation of voltage-dependent transition rates and functions, computation of time derivative of non-gating variables and a C-function for initialisation of the device's data structures with correct properties. The time derivatives are defined explicitly only for "other" variables, and in other cases they are implied from a standard form for the particular type of equations.

The formal parameters the C-functions implementing Markov chain ion channel models and their functionality are specified in this chapter including the changes for the implementation of exponential solver of Markov chains. The Markov chain is divided into subchains according to the control variable of transition rates matrices. The `ionic` model was extended by an individual C-function to fill up the transition matrix corresponding to a subchain of the Markov chain model.

In the `rushlarsen` device the C-functions calculating the transition rates matrices are used before the simulation starts. After the transition rates matrices are filled up, an exponential operator $\boldsymbol{T}(\vec{y}_j, \Delta t)$ is tabulated for a particular value of time step using the eigenvalue decomposition of the transition rates matrices. During the simulation a particular entry of the tabulated matrix is fetched depending on the value of a simulated control variable at the particular time step.

The simulation protocol is set up using `bbs` script. A summary of relevant parameters that can be used in a call of the `rushlarsen` device is shown in Table 6.2. The horizontal lines separate corresponding parameters to the `euler` device (as described in Table 6.1) from the specific parameters for the `rushlarsen` device. The second horizontal line separates new parameters used for the integration of the Markov chains.

Parameters of `rushlarsen` Analogous to `euler` Device

The following parameters of the `rushlarsen` device are analogous to the `euler` device.

The parameter `ht` defines the duration of one time step used in forward Euler and Rush-Larsen calculations.

The model of cellular kinetics is passed to the rushlarsen device through the string parameter ionic. The module has to follow the ionic format.

Module dependent coefficients in the model equations of the ionic model can be set using the bbs script parameter par. This assignment is done through a block of code in a form <parameter>=<value>. The <parameter> is a name of the coefficient specific to the particular ionic module (specified in ionic). If more than one parameter is specified all specification must be within the same curly brackets and each assignment must be separated by a blank space.

The rushlarsen device can approximate the resting state of the variables to be used as initial conditions. This is done by simulation for the number of steps specified in the rest parameter.

Table 6.2: Parameters to set up rushlarsen device in bbs script (adapted from [26] *rushlarsen.c*).

type	name	description
Analogous to euler		
real	ht	time step duration.
str	ionic	name of the cellular model in ionic format.
int	rest	number of steps to approximate initial conditions of the dynamical variables (resting state). When the value is set to zero (0), the default initial conditions from the module are used.
codeblock	par	model-dependent parameters of ionic module. The parameter is set by a codeblock in a form par={<parameter>=<value>} separated by blank spaces if more than one parameter is specified.
Specific for rushlarsen		
str	order	the order of execution of substeps: one of tog, tgo, totg, where the substeps are done consecutively: t stands for table look-up, o stands for “other” variables and computation non-tabulated gates, g stands for tabulated gating variables.
str	exp_ngate	defines the method of computation of non-tabulated gates: non-zero for exponential (Rush-Larsen); zero (0 default) for forward Euler method.
real	Vmin, Vmax	minimal and maximal value of voltage for tabulation. Default is -200, 200 respectivelly.
real	dV	voltage-step for the tabulation of gating transition rates. Default is 0.01. Value 0.0 disables tabulation.
Specific for rushlarsen*, related to Markov chains* *		
str	exp_mc	specifies the method of integration of Markov chains: mcfe forward Euler, tabmrl MRL with tabulation, ntabmrl MRL without tabulation (MRL only if subchain has a control variable, otherwise uses forward Euler)

Specific Parameters of rushlarsen

The following part describes bbs script parameters specific to the rushlarsen device i.e. parameters which do not have an analogy in the euler device.

The computation of the different groups of dynamical variables (i.e. tabulated and non-tabulated gates, and “other” variables), is done consecutively. The particular order in which the computations are performed can be controlled by the order parameter. The order can be set to the value of one of the following codes: tog, tgo, or totg. The order of letters in the codes corresponds to the order in which the particular group is computed. The letter t stands for finding the reference to a particular entry in a look up table of the tabulated voltage-dependent gates. The letter g stands for computing of the tabulated gates. The letter o stands for computing of the remaining variables i.e. non-gating variables followed by computing of non-tabulated gating variables and Markov chain variables.

For example, when the code totg is specified, the computation will be done in the following order:

1. tabulated gates, followed by the computation of
2. non-gating, non-tabulated and Markov chain variables, and then
3. tabulated gates again (now for updated values of dynamical variables), and finally the computation of
4. tabulated gating variables.

This is repeated for each time step during the whole simulation.

The parameter exp_ngate is used to choose the integration algorithm of the non-tabulated gating variables. The value 0 (default) stands for computation by forward Euler. If the value is non-zero, the non-tabulated gates are computed using the Rush-Larsen technique. In cases when the non-tabulated gates are not the source of numerical instability the forward Euler method is faster, because it avoids computationally expensive calculation of exponentials.

Three parameters are used to define the tabulation of the transition rates of the voltage-dependent gate variables. Specifically those parameters set up the lower and upper limits of the tabulation by Vmin (default -200) and Vmax (default 200) respectively, and the step in the voltage grid by dV (default 0.01).

If the value of dV=0.0, the tabulation is disabled and all transition rates are computed on-the-fly. If the value of membrane voltage exceeds the limits of tabulation, the transition rates are found on-the-fly, until the value of voltage is restored within the limits of the look-up table.

Specific Parameters of rushlarsen Related to Markov Chains*

The default method for the integration of Markov chains can be provided in a bbs script using a parameter exp_mc. The possible values of this parameter are mcfe for forward Euler, tabmrl for MRL with tabulation, and ntabmrl for MRL without tabulation. However, due to specific properties of a particular subchain, the method for the integration of the subchain is determined during the computation. In a standard situation, the subchain is integrated by the default method. If the subchain depends on multiple variables, then the tabulation cannot be used and the subchain is computed on-the-fly using forward Euler method.

Running rushlarsen

The rushlarsen device can be run during simulation in BeatBox by setting up a corresponding line in a bbs script. An example setup of rushlarsen device can be found in the bbs script listed in Listings D.19, and D.23 for a case with and without Markov chain models. The relevant parts are:

Listing 6.2: Call of rushlarsen device with parameters within bbs script.

```
def int neqn 4; /* number of layers of state variables */
def real dt 0.01;           /* time step */
/* Reaction substep */
rushlarsen v0=0 v1=neqn-1 ht=dt ionic=hh52 order=tog
  exp_mc=ntabmrl par={ht=dt};
```

6.6 Specification of ionic Modules

The ionic modules split the system of equation describing the kinetics of the reaction system according to their type and other characteristics as discussed in section 6.4.2.

This section specifies relevant structures and C-functions within the new format for ionic module. The new ionic format includes a separate definition of Markov chain models to allow their integration in the rushlarsen device using forward Euler and exponential methods. This section also specifies some of the features of the rushlarsen device which is used for the integration of ionic modules. The rushlarsen device performs the time integration of ionic cellular models as explained in the previous section. The section is divided into the description of the data structures and C-functions that operate on the data structures. The C-functions are implemented within the cellular modules.

A minimalist example of ionic code is shown in Appendix sections D.3.2 and D.3.3.

6.6.1 Data Structures

The implementation of data structures in BeatBox is hierarchical. A diagram in Figure 6.2 shows the data structure used by the rushlarsen device. The top level structure STR contains an element I which refers to an underlying structure ionic_str. The structure ionic_str points to channel_str called channel which contains Markov chains. Each Markov chain is divided to a number of subchains of

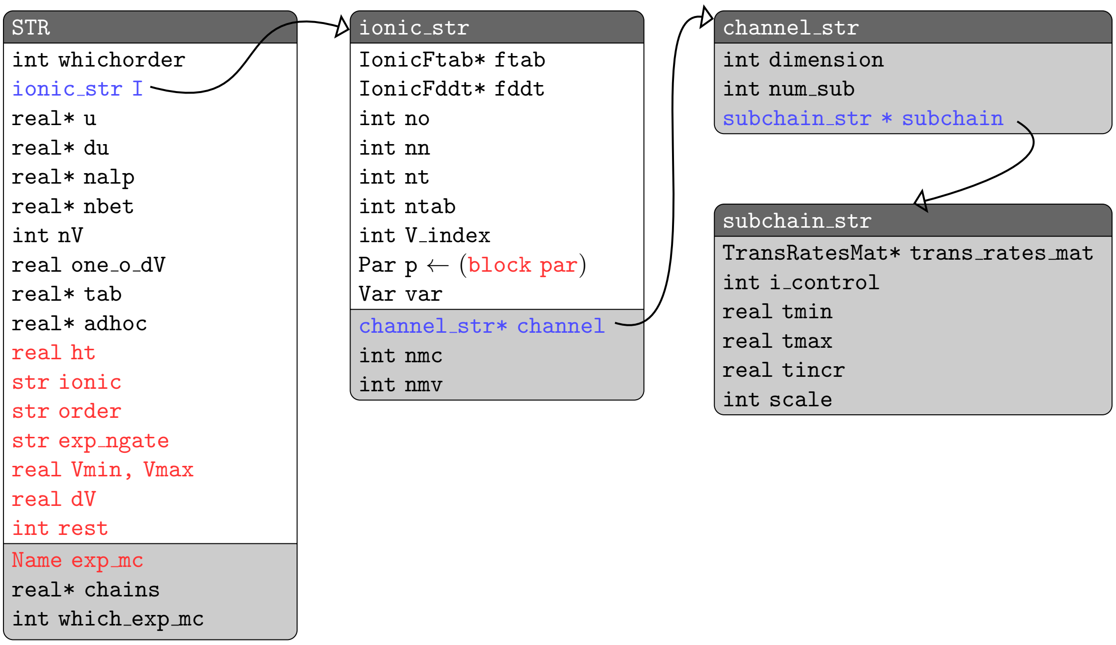


Figure 6.2: Data structures in `rushlarsen` device. Elements accepted from `bbs` script (red font); elements specifying underlying substructures (blue font, with arrow); new elements introduced for the implementation of Markov chain models (grey background).

type `subchain_str` to which the `channel_str` points through the element named `subchain`.

Some elements of the `rushlarsen` data structure `STR` can be provided through parameters of `bbs` script (in red on Figure 6.2). The name of the parameter of `bbs` script, which initialises an element of the structure, is normally identical to the variable name of the element (except `p` initialised by `par` parameter). Most of the parameters are not required in `bbs` script, and when they are missing the element adopts its default value.

Detailed information about the elements initialised from `bbs` script can be found in Table 6.2. The other elements of the data structures are used to act in internal tasks of the `rushlarsen` device and hold intermediate results, look up tables, or a translation of `bbs` script variables in a different type.

In this subsection we describe the hierarchical structure from bottom up.

Elements of `subchain_str` *

The lowest level of the data structure is the `subchain_str` which describes the subchains for the operator splitting of a Markov chain. Table 6.3 describes the elements of the structure.

The element `trans_rates_mat` is a pointer to a C-function `<ionic>_<subchain>` which computes the transition matrix of the subchain. The functionality of this C-function is described in the following subsubsection.

Table 6.3: Elements of a data structure subchain_str describing subchains of a Markov chain.

type	name	description
TransRatesMat*	trans_rates_mat	C-function computing transition matrix
int	i_control	index of control variable for tabulation in u array
real	tmin	minimal value for tabulation
real	tmax	maximal value for tabulation
real	tincr	increment in the tabulation
int	scale	0 for regular grid, 1 for logarithmic (must be reflected in TransRatesMat)

The element i_control is an index of the control variable within the state array u which controls the transition matrix. For instance, if the transition matrix depends on the membrane voltage, and the membrane voltage is the first element of the state array u, than the i_control=0.

A negative value of the i_control is accepted and means that the tabulation for the subchain should be disabled. In this case the transition matrix is computed on-the-fly. When the transition matrix depends on more than one variable then the i_control should be always set to zero, as the tabulation in a multiple variables grid is not allowed in BeatBox.

The scale of the tabulation variable is determined by the element scale. A value of 0 means a regular grid and a value of 1 means a logarithmic grid. The tabulation for the regular grid and the logarithmic grid was described in the Section 6.2.1.

The elements tmin and tmax specify limits of the tabulation and the element tincr is an increment on the tabulation grid for the control variable. The increment in the table tincr is given on the corresponding scale i.e. on the regular grid for scale=0 or logarithmic grid for scale=1. The tincr has to be strictly greater than 0.0, otherwise, the tabulation is disabled.

Elements of channel_str *

The subchains are part of a data structure channel_str. The maximum number of channels is set up through a macro MAX_SUBCHAINS within *src/channel.h* file and is currently set to 4 subchains. Elements of the channel_str data structure are summarised in the Table 6.4.

The number of the subchains in a Markov chain is specified by the element num_sub. The element subchain points to the first subchain of the channel. The element dimensions specifies the dimensionality of the channel. All the subchains must have the same dimensionality.

Table 6.4: Elements of data structure `channel_str` of Markov chains.

type	name	description
`int`	`dimension`	dimensionality of the model
`int`	`num_sub`	number of subchains
`subchain_str *`	`subchain`	subchains of the model

Table 6.5: Elements of data structure `ionic_str` of an `ionic` module (adapted from [26]).

type	name	description
`IonicFtab*`	`ftab`	C-function of voltage-dependent functions that are tabulated
`IonicFddt*`	`fddt`	C-function of right-hand sides of non-gating dynamical equations
`int`	`no`	number of non-gating ("other") variables
`int`	`nn`	number of "non-tabulated" gating variables
`int`	`nt`	number of "tabulated" gating variables
`int`	`ntab`	number of transition rates and voltage-dependent functions that are tabulated
`int`	`V_index`	index of the voltage in the state vector
`Par`	`p`	vector of model elements
`Var`	`var`	description of dependent elements
Related to Markov chains *		
`channel_str *`	`channel`	structures with definition of Markov chains
`int`	`nmc`	number of the Markov chain models
`int`	`nmv`	number of variables in all Markov chains

Elements of `ionic_str`

An `ionic` module can include several Markov chains referred from an `ionic_str` data structure. The structure `ionic_str` contains elements describing characteristics of the `ionic` module. A specification of the data structure `ionic_str` is found in Table 6.5.

The element `ftab` refers to C-functions computing the "tabulated" transition rates of gating variables. The element `fddt` refers to a C-function which computes the time derivative of non-gating "other" variables and also the transition rates of "non-tabulated" gating variables.

The number of the dynamical variables in each of the groups is specified by `no`, `nn`, `nt` for number of "other" variables, "non-tabulated" and "tabulated" gating variables respectively.

The number of tabulated functions is specified by the element `ntab`. This includes the transition rates of "tabulated" gating variables; it can also include other voltage-dependent variables.

The index of the membrane voltage is specified by `V_index`.

Table 6.6: Elements of data structures of `rushlarsen` device (adapted from [26]).

type	name	description
`int`	`whichorder`	numeric code of the order of execution
`ionic_str`	`I`	structure of `ionic` module
`real*`	`u`	array of dynamical variables
`real*`	`du`	array of the time derivatives of `u`
`real*`	`nalp`	array of non-tabulated transition rates α
`real*`	`nbet`	array of non-tabulated transition rates β
`int`	`nV`	number of rows in the table
`real`	`one_o_dV`	inverse of voltage increment in tabulation
`real*`	`tab`	look-up table of values of functions
`real*`	`adhoc`	array of *ad hoc* values of tabulable functions
Related to Markov chains *		
`real*`	`chains`	pointer to transition rates matrices of Markov chains
`int`	`which_exp_mc`	numeric code of integration method for Markov chains (assigned to enumerated type 0: `mcfe`, 1: `tabmrl`, 2: `ntabmrl`)

Dependent parameters are given by structure `var` which contains the conditions of specific parameters that depend on spatial dimensions of the tissue.

A structure of module dependent coefficients is given by the element `p` that refers to a data structure defined within a particular `ionic` module.

The element `nmc` specifies a number of Markov chain models in the module. The element `nmv` specifies the total number of variables in all Markov chain models.

The element `channel` is a pointer to the first Markov chain ion channel structure in the model.

Elements of `rushlarsen` Data Structure

The `ionic_str` data structure is an element of `rushlarsen` data structure `STR`. This structure contains elements which help to set up the simulation protocol within the `rushlarsen` device and saves intermediate results of the calculations. The summary of the elements of `STR` structure is given in Table 6.6.

The element `whichorder` contains the translation of the string `order` accepted from `bbs` script into a numerical value. The value is set depending on the parameters of `order` and `exp_ngate` in the `bbs` script for `rushlarsen`.

The element `I` refers to a data structure `ionic_str` which specifies the cellular module, as was described above.

The array `u` contains the vector of all dynamic variables of the `ionic` model. The array `du` contains time derivative of “other” non-gating variables. The gating and Markov chain variables are excluded from the array `du`, because they are computed separately from the "other" and non-gating variables and the corresponding entry in the vector of dynamical variables u is updated directly.

The arrays nalp and nbet contain non-tabulated transition rates for opening and closing transitions respectively.

The array tab contains the table of tabulated transition rates. The element nV specifies the number of rows in the look-up table. The one_o_dV is an inverse of voltage increment used in the tables. The array adhoc is used for the computation of tabulated functions when the value of membrane potential exceeds the precomputed limits.

The array chains contains the tabulated matrices of Markov chains. Depending on the method used for a particular subchain, it contains either the transition matrix $\boldsymbol{A}$ for forward Euler computations, or the exponential operator matrix $\boldsymbol{T} = \boldsymbol{V} \exp\left(\boldsymbol{\Lambda}\Delta t\right) \boldsymbol{W}^{\boldsymbol{T}}$ for matrix Rush-Larsen integration.

The element which_exp_mc is used to convert the name of the integration method of the Markov chain models into a numerical code. The value 0 corresponds to computation by forward Euler, value 1 to the computation by matrix Rush-Larsen with tabulated transition rate matrices, and value 2 corresponds to matrix Rush-Larsen with computation of transition rate matrices on-the-fly. The value is determined automatically from the code provided by bbs script in the parameter exp_mc. If the subchains are not to be tabulated e.g. due to the dependence on multiple variables, then the corresponding submatrix is computed on-the-fly.

6.6.2 C-Functions and Template Macros

The data structure of the rushlarsen device refers to a number of C-functions defined within ionic modules. This subsection describes the C-functions to provide the necessary information for implementation of ionic modules.

The implementation of C-functions used by rushlarsen device is facilitated by template macros available from *src/ionic.h* in the BeatBox repository. The macros are expanded by C preprocessor (cpp) during the compilation of the BeatBox source code.

The first C-function called by the rushlarsen device is the create_<ionic> where <ionic> is the name of the model. The create_<ionic> is called only once and its purpose is to allocate memory of the data structures for a particular model and assign their entries to values provided from the bbs script or their default values. The input arguments of create_<ionic> C-function are specified in Table 6.7.

The IONIC_CREATE_HEAD(<ionic>) and IONIC_CREATE_TAIL(<ionic>) macros define an IonicCreate C-function. Once the macros have been expanded the name of the C-function becomes create_<ionic>.

Table 6.7: Input arguments of the C-function to initialise an ionic module (IonicCreate create_<ionic>(ionic_str *I,char *w,real **u,int v0))

type	name	description
ionic_str *	I	pointer to ionic structure to be initialised
char *	w	parameters to be assigned from script
real **	u	pointer to array of states variables
int	v0	number of entries in states array

The C-function fddt_<ionic> is a C-function computing the time derivative of the non-gating ("other") variables as defined on the right hand side of the equation (6.2c) and computing the non-tabulated transition rates of gates corresponding to α^j and β^j in equation (6.2b). The input arguments of the fddt_<ionic> C-function are specified in Table 6.8.

The macros IONIC_FDDT_HEAD(<ionic>,NV,NTAB,NO,NN) and IONIC_FDDT_TAIL(<ionic>) define an IonicFddt type C-function. Once the macros have been expanded the name of the C-function becomes fddt_<ionic>. The variables in the template of the macro are the numerical values of the total number of variables NV, the number of voltage-dependent tabulated functions NTAB, the number of non-gating "other" variables NO, and the number of "non-tabulated" gating variables NN.

Table 6.8: Input arguments of the C-function computing the kinetics of non-gating and non-tabulated transition rates of gating variables in an ionic module (IonicFddt fddt_<ionic>(real *u,int nv,real *values,int ntab,Par par, Var var,real *du,int no,real *nalp,real *nbet,int nn))

type	name	description
real *	u	array of dynamical variables
int	nv	total number of dynamical variables
real *	values	array of tabulated transition rates
int	ntab	number of tabulated transition rates
Par	par	parameter structure
Var	var	variable structure
real *	du	pointer to an array of increments non-gating "other" variables
int	no	number of "other" variables
real *	nalp	array of non-tabulated α
real *	nbet	array of non-tabulated β
int	nn	number of non-tabulated gates

The second ftab_<model> is a C-function of voltage-dependent tabulated transition rates of gating variables. The input arguments of the C-function ftab_<ionic> are specified in Table 6.9.

The IONIC_FTAB_HEAD(<ionic>) and IONIC_FTAB_TAIL(<ionic>) macros define IonicFtab C-function. Once the macros have been expanded the name of the C-function becomes ftab_<ionic>.

Table 6.9: Input arguments of the C-function for computing tabulated transition rates of gating variables in ionic module (IonicFtab ftab_<ionic>(real V, real *values, int ntab))

type	name	description
real	V	membrane voltage
real *	values	array to be filled with steady-state coefficients tabulated transition rates
int	ntab	number of tabulated variables

Notice that both C-functions ftab_<ionic> and fddt_<ionic> compute the coefficients of the transition rates of gating variables instead of the time derivative of gating variables. The dynamical equations of the gating variables have a standard form, so the time derivative can be inferred from the known form and given values of transition rates.

Markov Chain Specific *

The *src/channel.h* is a Markov chain specific header file. It contains definitions of the data structures and template macros for the construction of a Markov chain model.

The C-function <ionic>_<subchain> computes the transition rates of a subchain of a Markov chain model. The results are calculated on-the-fly during the simulations and discarded at the end of each time step, or placed to corresponding entries of the transition matrix. The input arguments of the C-function <ionic>_<subchain> are specified in Table 6.10.

The macro CHANNEL_TR_MATRIX(<ionic>_<subchain>) defines a TransRatesMat C-function. After the macro has been expanded the name of the C-function becomes <ionic>_<subchain>.

Table 6.10: Input arguments of a C-function computing transition rates of Markov chains (TransRatesMat <ionic>_<subchain>(real * u, real *tr_mat))

type	name	description
real *	u	array of dynamical variables
real *	tr_mat	transition matrix

The input argument u specifies the pointer to the array of dynamical variables. This serves for the computation of the transition rates functions. The second formal input argument tr_mat is a pointer to the first element of the array of the transition matrix.

```
_RATE(from,to,direct,reverse)
```

(a)

$$S_1 \xrightleftharpoons[\dots]{\dots} \texttt{from} \xrightleftharpoons[\texttt{reverse}]{\texttt{direct}} \texttt{to} \xrightleftharpoons[\dots]{\dots} S_2$$

(b)

$$\begin{array}{c|cccc|}
 & S_1 & \texttt{<channel>\_from} & \texttt{<channel>\_to} & S_2 \\
\hline
S_1 & \ddots & \dots & 0 & 0 \\
\texttt{<channel>\_from} & \dots & -(\texttt{direct}+\dots) & \texttt{reverse} & 0 \\
\texttt{<channel>\_to} & 0 & \texttt{direct} & -(\texttt{reverse}+\dots) & \dots \\
S_2 & 0 & 0 & \dots & \ddots \\
\hline
\end{array}$$

(c)

Figure 6.3: Construction of the transition matrix: (a) a form of the function-like macro with arguments; (b) a diagram of the Markov chain (with additional states S_1 and S_2) corresponding to the function-like macro; (c) part of the transition matrix constructed from the function-like macro.

The C-function `<ionic>_<subchain>` fills the entries of the matrix by the corresponding transition rates. The filling process can be implemented using template macro `TR_MAT(chan,from, to, direct, reverse)`. It assumes that the transition rates are defined by a macro in a form `_RATE(from,to,direct,reverse)` where, as the name suggests, the first two arguments specify two states `from`, `to` between which the transition rates are defined in `direct`, and `reverse` expressions.

Figure 6.3 illustrates the process of filling the entries of the transition matrix. Panel (a) shows the form of the function-like macro with arguments (description of function-like macros can be found on page 228). The arguments are substituted during the preprocessing phase of the compilation with the names of the states and the expressions for the transition rates. A diagram of a part of the Markov chain described by a given function-like macro is shown in panel (b). Panel (c) shows the corresponding entries in the transition matrix filled with the expressions for the transition rates.

In the figure, the number of the position of the state is given by `<channel>_from` and `<channel>_to` where the `from` and `to` are substituted with the arguments of the function-like macro. To find the correct entries of the transition matrix, we use an enumeration of variables of the Markov chain. The enumeration starts with `0` for the first state and `dimension-1` for the last state of the Markov chain.

The *channel.h* also provides other macros to assign the elements of the data structures of the Markov chain models within an `ionic` module. The elements of the structure `subchain_str` should be assigned through a template macro `SUBCHAIN(fun_tr, index, min, max, incr, sc)`. This assigns the transition rates function `trans_rates_mat`, the index of the control (independent) variable for tabulation `i_control`, the minimum `tmin`, and the maximum `tmax` limits of the control variable in the table, the tabulation step `tincr` on the corresponding tabulation scale and the scale of the tabulation `scale`. The predefined macro also calculates the number of subchains (`num_sub`) automatically.

The macro `SUBCHAIN` assumes the existence of two pointers: a pointer to the current channel `channel_str * ch;`, and another one to the current substring as `subchain_str * sbch;`. The pointer `sbch` is incremented within the `SUBCHAIN` macro, while the `ch` must be incremented manually in `create_<ionic>` function which is generated from `IONIC_CREATE_HEAD` block within the `ionic` module.

A minimalist example of the implementation of an `ionic` module with the definitions of two Markov chain models can be found in Appendix Section D.3.3.

6.7 Testing of BeatBox `ionic` Modules with Markov Chains

For the sake of implementation and testing of the matrix Rush-Larsen method we have implemented an `ionic` module with Markov chains defined according to the specifications described in the previous section.

In the previous chapter, we have used the Clancy, Rudy (2002) [2] cellular model including I_{Na} channel. This model can be implemented as a `rhs` module, however not as an `ionic` module due to (a) the implicit definition of buffered Ca^{2+} concentrations, that are based on the values at the previous time steps, and due to (b) time-delayed calcium release from the sarcoplasmic reticulum (SR). The problem (a) can be readily overcome by equivalent reformulation of the relevant equations, however the problem (b) is not straightforward to address within the current BeatBox framework.

The simplest case of a Markov chain model is achieved by a transformation of a gate model. This approach has also an advantage of a straightforward comparison with the solution of the gate model. We have chosen the Hodgkin-Huxley (1952) squid axon for its simplicity and the reader's familiarity with the model, although use of this model has been surpassed by modern models.

To demonstrate the functionality of the `rushlarsen` module on the Markov chain, within a modern biophysically detailed model we have used TenTusscher-Panfilov (2006) human cardiac cell model with added Markov chain models.

Finally, we have also implemented a physiological model published by Faber *et al.* (2007) [1] which is a model of a ventricular cell that includes stiff Markov chains.

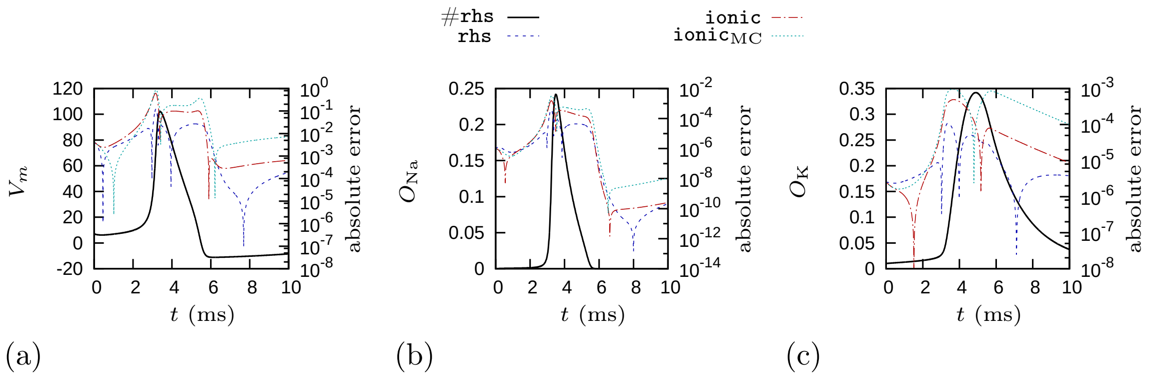


Figure 6.4: Simulation results and absolute error of the Hodgkin-Huxley squid model in BeatBox: (a) membrane voltage $\mathrm{V_m}$, (b) open probability of Na^+ channels O_{Na}, (c) open probability of K^+ channels O_{K}. Black solid line shows the “accurate” simulated traces of `rhs` model with $\Delta t = 0.1$ μs (left axis); the remaining lines show the difference between the “accurate” simulation and simulations with time step $\Delta t = 1$ μs using `rhs` model (blue long-dashed lines), `ionic` model with gate model of ion channels, `ionic` model with ion channels converted to equivalent Markov chain model. Ionic models were solved using exponential integration without tabulation (transition rates computed on-the-fly).

6.7.1 Hodgkin-Huxley Minimalist Model

The code of a minimalist model was implemented in a standalone version, and as three BeatBox modules - one in `rhs` format and two in `ionic` format: the first in the original form with the Hodgkin-Huxley gate formulation of ion channels, the second converts the gate models as the equivalent models in a Markov chain framework.

The code of the implementation of the models, details about the conversion of gate models to Markov chains and code listings can be found in Appendix D.

Figure 6.4 shows the results of the simulations using the three BeatBox versions of the Hodgkin-Huxley squid model. The membrane voltage and open probability traces are shown in black. The colour lines show the difference between the “accurate” simulation using a `rhs` module with a small time step $\Delta t = 0.1$ μs (denoted as #`rhs`) compared with the simulations using the `rhs` module, and both `ionic` modules with a larger time step $\Delta t = 1$ μs (colour line with scale on right axis). The transition rates in the `ionic` models were not tabulated but computed on-the-fly during the simulations. The gating variables in the first `ionic` module and Markov chains in the second were computed using exponential integration methods.

The difference between the “accurate” simulation and the solution `rhs` simulation is caused purely by the time step increase, as the other factors remained identical. The difference in `ionic` modules beyond the one observed in the `rhs` is caused by different order of computation of gating and the Markov chain variables, and membrane voltage in both cases. The absolute error in both `ionic` models visually overlaps, which suggests that the gate and Markov chain models of ion channels are equivalent.

The absolute difference of the solutions from both `ionic` modules remains below 10^{-11} for both open probabilities O_{Na} and O_{K} and below 10^{-8} for the membrane voltage. This small deviation is caused by the use of different methods – Rush-Larsen for gate variables and matrix Rush-Larsen for Markov chain variables (where the exponential is computed using eigenvalue decomposition of the transition matrix).

The minimalist model described in this subsection is a demonstration of the functionality of simple cellular modules with Markov chains. To demonstrate the full capabilities of the `rushlarsen` device we aim to implement an `ionic` module using a modern biophysically detailed cellular model.

6.7.2 TenTusscher-Panfilov (2006) Model

The demonstration of the functionality of the `rushlarsen` device is done using TenTusscher-Panfilov (2006) human ventricular model (TTP) [29]. There are no non-trivial Markov chain models in TTP, therefore we add two such models. The extended cellular model is not based on physiological measurements and should not be considered as a realistic model. The inclusion of the Markov chains was done for the sake of testing of the numerical methods developed into BeatBox. In Section 5.2 we have used MRL method for currents of calcium channel $I_{\mathrm{Ca}(L)}$ [1], Ryanodine Receptor (RyR) [1] and sodium channel I_{Na} [2] as stand-alone code. We have decided to implement two of those Markov chain models into a BeatBox TTP module, namely the $I_{\mathrm{Ca}(L)}$ and I_{Na} models.

The BeatBox distribution already contained `rhs` definition of TTP model which was based on the original authors' code. Before we proceed to the conversion to `ionic` format, we need to replace the implementation of some of the calcium concentrations which are in a true right hand side format. Based on the authors implementation, we have reconstructed the definition of calcium concentration as dynamical equations. The reconstruction is described in detail in Appendix D.4.1.

We converted TTP `rhs`module present in BeatBox into an `ionic` module. Further details about the conversion from `rhs` module can be found in Appendix Section D.4.5.

Figure 6.5 shows the simulation results using TTP model. The open probabilities contribute to the computation of I_{Na} and $I_{\mathrm{Ca}(L)}$ ionic currents. The gating variables of Hodgkin-Huxley type gates were computed using the Rush-Larsen method. The tabulation of voltage-dependent coefficients in the Rush-Larsen method uses a tabulation step of $\Delta V_{m,tab.} = 0.01\ \mathrm{mV}$. The reference solution was obtained using the MRL method. The discrepancy of the MRL solution with

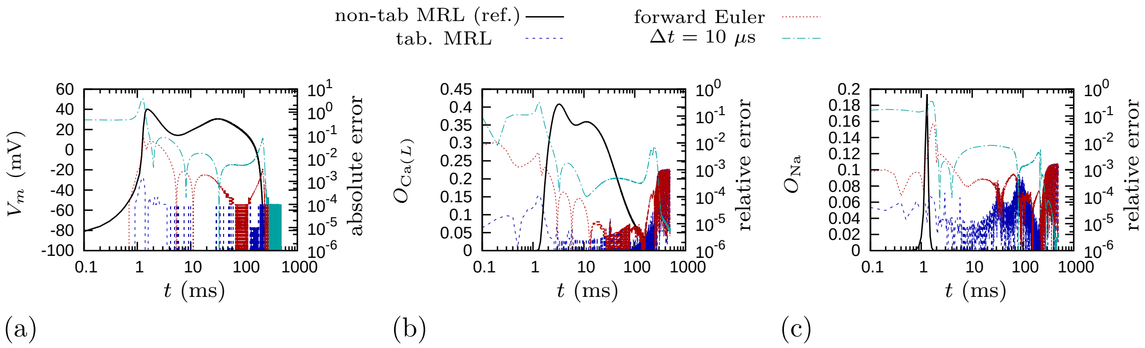


Figure 6.5: Comparison of the simulation methods of the Markov chain models for TTP model. Panel (a) shows membrane voltage $\mathrm{V_m}$, panel (b) shows open probability $I_{\mathrm{Ca}(L)}$, and panel (c) shows open probability I_{Na} currents. Black lines show the reference solution for the Matrix Rush-Larsen (MRL) method at the time-step $\Delta t = 1\ \mu\mathrm{s}$, coloured lines show the relative error of the reference solution (left axis) with: the MRL method with tabulated transition matrix (blue lines), the forward Euler solution (red lines), both methods use the same time-step as the reference solution $\Delta t = 1\ \mu\mathrm{s}$; cyan lines show the simulations with the same method as the reference solution with the time-step $\Delta t = 10\ \mu\mathrm{s}$.

tabulation show a small error comparable with corresponding tabulation inaccuracy for tabulated gating variables (e.g. in Figure D.2).

The discrepancy from the forward Euler solution is negligible compared to the error due to the increase of the time-step size to $\Delta t = 10\ \mu\mathrm{s}$ in the same method as reference solution (non tabulated MRL). The error due to the time-step increase will be discussed in detail in the following subsection.

6.7.3 Faber *et al.* (2007) model

In the previous chapter we have used the Faber *et al.* (2007) model. Although the Faber *et. al* (2007) model was published by the same lab as the Clancy, Rudy (2002) model, it does not contain time-delayed calcium release. Therefore it can be implemented in ionic format.

The Faber et. al (2007) model contains two Markov chain models of $I_{\mathrm{Ca}(L)}$ (which we used in the previous subsection within TTP model), and the RyR model. In the previous chapter we have developed exponential integration of the RyR and $I_{\mathrm{Ca}(L)}$ models within the standalone code of Faber et. al (2007). Here we aim to implement BeatBox ionic module of this model.

The exponential integration of the RyR model was more challenging. Unlike the I_{Na} and $I_{\mathrm{Ca}(L)}$ models, that contained only voltage-dependent transition rates in the operator matrix, the transition matrix of the RyR model depends on multiple variables corresponding to intracellular calcium concentrations. The transition matrix was split into two submatrices which each represented part of the transitions of the Markov chain. Briefly, the fast transition rates compose the first operator matrix computed by MRL method, and the slow transition rates compose the

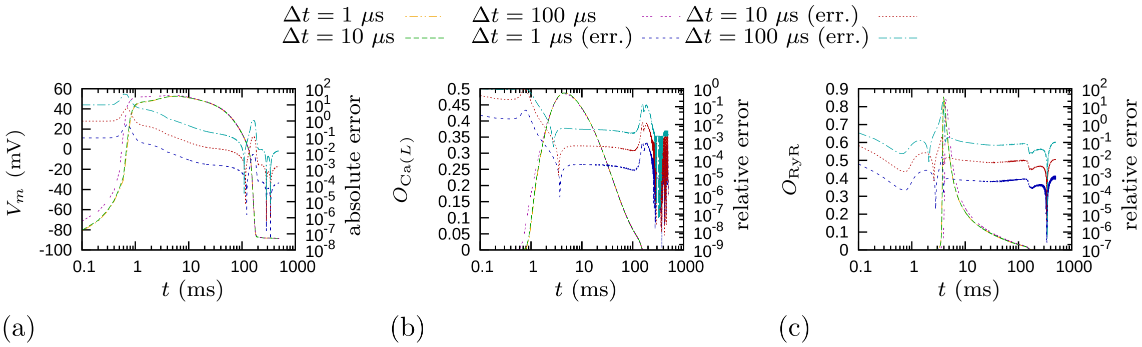


Figure 6.6: Comparison of simulations with exponential methods using Faber *et al.* (2007) model. Panel (a) shows membrane voltage $\mathrm{V_m}$, panel (b) shows open probability of $I_{\mathrm{Ca}(L)}$, and panel (c) shows open probability of RyR currents. The reference solution (not shown) was computed at the time step $\Delta t = 0.1$ μs using the tabulation for the transition rates matrices (`tabmrl`). The simulation using `tabmrl` at the time steps $\Delta t =$ 1, 10, 100 μs are shown in orange, green and magenta lines respectively, and relative error of the solution from the reference are shown in blue, red and cyan lines respectively.

second operator computed by forward Euler. The tabulation in the domain of the calcium concentration was done in a logarithmic scale. More details about the division can be found in Section 5.2.2.

Figure 6.6 shows the simulation results of the Faber *et al.* (2007) model using the MRL method with tabulation (`ntabmrl`) within BeatBox. The figure shows the simulated traces (scale on the left axis) and the error of the approximation as compared with the reference solution (computed with the time step of $\Delta t = 0.1$ μs on the right axis). The error in membrane voltage is shown as absolute (to avoid large values around zero crossing), while the error in open probability is shown as relative to the value.

The morphology of the simulated traces is similar in all cases. The measure of an error using the absolute difference of the simulated traces is inapropriate for parts of the traces, which change fast in time. This could be understood by considering a step function where the change happens infinitely fast. If we compare the difference of two step functions with the same initial and the same final values and where the only difference is the instant when the functions turn from the lower to the higher value, then the resulting deviation between the turns corresponds to the amplitude of the functions. Because the traces of the action potential has similar characteristics at the fast onset, we consider the apparent inacuracy being a limitation of the measure of the error.

Table 6.11 lists the computational cost spend on a simulation of 500 ms of simulation in Faber *et al.* (2007) model. The simulation was performed in a GNU/Linux box with the processor of Intel Core i5-3470 CPU with clock frequency 3.20 GHz. The computational time here shown is the the total time spent by the

Table 6.11: Computational cost [in seconds] spend in a simulation of a duration of 500 ms using the Faber *et al.* [1] cellular model. The first column specifies the integrating method used for the Markov chain models which are set using `bbs` script parameter `exp_mc`, namely: `mcfe` stands for forward Euler, `tabmrl` stands for matrix Rush-Larsen with tabulation, and `ntabmrl` stands for matrix Rush-Larsen with the transition rates computed on-the-fly. The following columns show the computational time at $\Delta t = 1, 10$ and 100 μs.

`exp_mc`	Δt =1	Δt =10	Δt =100 [μs]
`mcfe`	3.6	–	–
`tabmrl`	4.9	1.9	1.6
`ntabmrl`	38.0	4.0	0.6

computation in BeatBox which includes setting up the cellular module, tabulation of the transition rates and the transition rates matrices, and the simulation itself.

The fastest simulations for a fixed time step size are achieved by the forward Euler method. Comparing the computational time at the time step of $\Delta t = 1$ μs the `tabmrl` method is slower by 1.3 s which is due to the computation of the tabulated eigenvalue operator matrices of $I_{\mathrm{Ca}(L)}$ and fast operator matrix of RyR. The computation of the exponential integrator on-the-fly is the slowest method, which is more computationally expensive by ten fold.

The true advantage of the MRL method appears when we increase the time step. With the increasing time step the computational time reduces proportionally. However, the forward Euler method is limited by a value of $\Delta t = 6$ μs above which the solution becomes unstable and therefore cannot be used. Meanwhile the MRL method still provides stable solutions for the Markov chains, so the time step is only limited by instabilities in other components of the cellular model and the accuracy consideration of each particular study.

The table shows that at the time step of $\Delta t = 100$ μs the computation on-the-fly is faster than the one with tabulation. This is because the simulation at this value of the time step requires less computation than precomputing corresponding look-up tables.

6.8 Conclusions

The exponential solvers for Markov chains were implemented into BeatBox package for cardiac simulations and made publicly available for wider community use.

To explain the implementation we provided an overview of the reaction system of the cell and different types of dynamical variables. This includes “gating” variables, “Markov chain” dynamical variables and “other” variables. Previously, the “Markov chains” had to be implemented as part of the “other” variables and solved by a standard solver for a generic dynamical equation. In the format for Markov chain models we also allow operator splitting into subsystems.

The first task was to redefine a data structure of the `ionic` format, which is used to contain the parameters and variables of a cardiac cell. This format now allows distinction between gating variables (tabulated and non-tabulated), other variables and now also Markov chain variables.

The `ionic` modules are solved by a `rushlarsen` device, which is a module that sets up the simulation and integrates cellular models defined as `ionic` modules. We implemented the “hybrid” MRL methods in the `rushlarsen` device. This allows us to split the transition matrix into subsystems, allowing tabulations in cases where the transition matrix depends on multiple variables. In order to compute the diagonalisation we include an extract of the GNU Scientific Library into BeatBox package.

To test the implementation of the newly developed code, we converted three cellular models with Markov chain models of ion channels into the `ionic` format as described in Appendix D.

Finally, we published the documented code along with the BeatBox distribution. The modified files are also listed in Appendix C.

Chapter 7

Conclusions

7.1 Main Results

- Development of efficient and accurate numerical methods for integration of Markov chain models of ion channels:
  - asymptotic reduction of a Markov chain model to a system with zeroth-order and first-order terms in a small parameter which describes the time-scale separation of transition rates;
  - Matrix Rush-Larsen (MRL) method based on exponential integration;
  - "hybrid" method combining the MRL and traditional solvers, based on operator splitting of the transition matrix of a Markov chain;
  - theoretical assessment of the accuracy of the MRL and the hybrid methods.
- Application of the asymptotic, the MRL, and the hybrid methods for the integration of the Markov chain model of sodium channel I_{Na} developed by Clancy and Rudy (2002) [2]. In those examples, the asymptotic methods provided accurate approximations and formally allowed reduction of the number of dynamical equations. However, these methods did not resolve the instability issues. As a result, the time step limitations remained largely the same, and no significant speed-up was achieved with the asymptotic methods. On the other hand, the MRL and hybrid methods provide stable and accurate solutions.
- Application of the MRL and the hybrid exponential solvers to stiff RyR and $I_{\mathrm{Ca}(L)}$ Markov chains in the Faber *et al.* (2007) [30] cellular model. The forward Euler method, as implemented by the authors, requires time steps of

$6\ \mu\mathrm{s}$ in the RyR and of $37\ \mu\mathrm{s}$ in the $I_{\mathrm{Ca}(L)}$ to preserve stability. The exponential solvers provide a stable solution with reasonable accuracy up to $\Delta t = 180\ \mu\mathrm{s}$.

- Implementation of the developed methods into a free cardiac simulation package BeatBox:
  - extension of the format of `ionic` modules to allow specification of Markov chain models;
  - implementation of the MRL and the hybrid methods into the `rushlarsen` solver for exponential integration of `ionic` modules;
  - documentation of the code and testing its functionality based on translation of popular models into the new `ionic` format;
  - releasing the contributions along with BeatBox distribution.

7.2 Limitations

The MRL method relies on being able to diagonalise the transition matrix. All the examples we have worked with satisfy that condition, however, in other Markov chains the diagonalisation might fail. If the transition matrix cannot be diagonalised for only some values of the dependent variable, then an interpolation of the exponential operator at the specific values of the dependent variable might be sufficient. If the transition matrix has a specific form that prohibits the diagonalisation, a Jordan form could be used instead of the diagonal matrix.

We have tested the newly developed methods within a single cell model where they provide good results. In this situation the morphology of the action potential is standard, i.e. it exhibits fast upstroke after which the voltage returns to the resting value. However, in the tissue simulations some cells can exhibit a non-standard voltage profile where there is no fast upstroke, or where the voltage changes slowly around certain intermediate values. The accuracy of the results at those values could be affected, if the truncation error is large. The error in the MRL method is likely to be small, because it is dependent on the rate of change of voltage, which in this case varies slowly. However, the error in the operator splitting does not depend on the rate of change of the voltage. If the errors due to the operator splitting happen to be large at specific intermediate values of membrane voltage, the simulation can be unusable.

The efficiency of the hybrid method partially relies on the possibility of precomputing diagonalised transition matrices and exponential operators into look-up tables. The look-up tables are created for a grid of control variables. In the models we worked with, the transition rates matrices are dependent only on a single variable, or we are able to find an operator splitting in a way that transition matrices of the fast subsystems depend on a single variable. As a result, the tabulation was done only for a single-variable grid. However, if this splitting is impossible for some other Markov chain model, e.g. due to a specific dependence of the transition rates on multiple variables, an alternative approach would be required, such as computation of the diagonalisation on-the-fly, or tabulation on a grid of multiple variables. However, those approaches are only practical, when higher computational demands are compensated by savings due to the time step increase, as we seen in some examples in this thesis.

7.3 Further Work

The time step increase affects accuracy of the solution, but the accuracy requirements depend on each particular study. However, the numerical instabilities limit the time step size absolutely, i.e. an unstable solution is unusable for any purpose. Having addressed the instability, we can develop higher-order methods to improve accuracy, say by extending of the methods described by Perego and Veneziani (2009) [24], or Sundnes *et al.* (2009) [25] to Markov chains. Then, the same accuracy of the solution as in the first-order method could be achieved using larger time steps in higher-order methods.

Computational time is greatly reduced by the numerical methods developed in this thesis. However, in the single cell simulations, it is feasible to complete the simulation in an acceptable time frame, despite of the small time steps required to preserve stability in some Markov chain models. Spatial models of cardiac tissue contain many cells. Such simulations might take a prohibitive amount of computational time. This might be overcome if the exponential integration methods provide accurate and stable results in the spatial models, as promised by the results of the single cell simulations. Therefore, an application of the exponential integration within a spatial model would be an interesting direction for a subsequent study.

The computation of spatial models of cardiac tissue requires solving systems of PDEs. In principle, the exponential solvers could be generalised in a model with spatially distributed processes in a single cell in order to provide more accurate solution for the diffusion within the models [31, 32]. Thus, applying such solvers to spatial models of cardiac tissue would be yet another possibility for future work.

Studies of calcium handling within a single cell are crucial for understanding the excitation-contraction coupling. Such spatial cell models take into account the physical distribution of a single cell, and allow simulation of the diffusion within a spatial model described by partial differential equations (PDEs). Such models contain clusters of a finite number of $I_{\mathrm{Ca}(L)}$ and RyR channels represented by stochastic Markov chain models [33, 34, 35]. An adaptation of the exponential methods for such models would be an appealing direction for future research.

Besides the methods developed as part of this thesis, there are other approaches which aim to address numerical instability issues, such as implicit methods. Although, generic implicit methods are complicated, several numerical libraries provide such routines using implicit solvers. The investigation of the accuracy of those methods and their comparison to the methods developed in this thesis is another possible direction of further work.

A conversion of cellular models from published models into BeatBox format is laborious. A CellML project contains a large repository of cellular models based on the XML standard for physiological models. Tools for importing the CellML models into BeatBox could convert the models automatically to reduce the risk of manual introduction of errors. This future work would also require an extension of the CellML format in order to distinguish between the Markov chain, gating, and other variables.

Appendix A

Definition of Clancy-Rudy (2002) Model

This appendix contains the definition of the model of Clancy and Rudy (2002) [2] according to the author's code. The format of equations and subsections corresponds to the papers where those equations were published to facilitate a straightforward comparison. The differences between the authors code and the published papers are marked by the sign $^{\#}$.

The units of measurements are not essential for the context of the thesis and for practical reasons we do not mention them. We have not done any rescaling so all units correspond to the published papers.

The units of measurements were omitted as they are not essential in the context of the thesis. We have work only with the equations as provided by the authors code. As we did not do any rescaling, the units correspond to the original papers.

Standard ionic concentrations

$$
\begin{aligned}
[\mathrm{Na}^+]_i =& 7.9 \quad ^{\#}\text{(initial value of dynamical variable)}, && \text{(A.1)}\\
[\mathrm{Na}^+]_o =& 140, \quad ^{\#} && \text{(A.2)}\\
[\mathrm{K}^+]_i =& 147.23 \quad ^{\#}\text{(initial value of dynamical variable)}, && \text{(A.3)}\\
[\mathrm{K}^+]_o =& 4.5, \quad ^{\#} && \text{(A.4)}\\
[\mathrm{Ca}^{2+}]_o =& 1.8, && \text{(A.5)}
\end{aligned}
$$

which differs from the [36] where $[\mathrm{Na}^+]_o = 150$; $[\mathrm{K}^+]_i = 145$; $[\mathrm{K}^+]_o = 5.4$; $[\mathrm{Na}^+]_i = 10$ mmol/L.

Initial Values of Variables and Parameters

$$x_{s1} = 0^* \tag{A.6}$$

$$x_{s2} = 0^* \tag{A.7}$$

$$V = -95 \tag{A.8}$$

$$[\mathrm{Ca}^{2+}]_{\mathrm{NSR}} = 1.8 \tag{A.9}$$

$$[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} = 1.8 \tag{A.10}$$

$$[\mathrm{Ca}^{2+}]_i = 0.00012 \tag{A.11}$$

$$b = 0.00141379 \tag{A.12}$$

$$g = 0.98831 \tag{A.13}$$

$$d = 6.17507 \cdot 10^{-6} \tag{A.14}$$

$$f = 0.999357 \tag{A.15}$$

$$X_r = 2.14606 \cdot 10^{-4} \tag{A.16}$$

The values marked by * lack explicit initialisation in the author's code. This implies intialisation by the C compiler, which in our version of gcc corresponds to 0.

Physical Constants

$$R = 8314 \tag{A.17}$$

$$F = 96485 \tag{A.18}$$

$$T = 310 \tag{A.19}$$

Cell geometry

$$L = 0.01 \tag{A.20}$$

$$r = 0.0011 \tag{A.21}$$

$$V_{cell} = 3.801 \cdot 10^{-5} \tag{A.22}$$

$$A_{Geo} = 2\pi r^2 + 2\pi r L \tag{A.23}$$

$$A_{Cap} = 2A_{Geo} \tag{A.24}$$

$$V_{myo} = 2.58468 \cdot 10^{-5} \tag{A.25}$$

$$V_{\mathrm{NSR}} = V_{cell} 0.0552 \tag{A.26}$$

$$V_{\mathrm{JSR}} = V_{cell} 0.0048 \tag{A.27}$$

Na^{+}-K^{+} **pump :** I_{NaK}

$$I_{\mathrm{NaK}} = 1.5 f_{\mathrm{NaK}} \frac{1}{1+(10/[\mathrm{Na}^{+}]_i)^{1.5}} \cdot \frac{[\mathrm{K}^{+}]_o}{[\mathrm{K}^{+}]_o + 1.5}, \tag{A.28}$$

$$f_{\mathrm{NaK}} = \frac{1}{1 + 0.1245 \exp\left(-0.1 \cdot \frac{VF}{RT}\right) + 0.0365\sigma \exp((-VF)/(RT))}, \tag{A.29}$$

$$\sigma = \frac{1}{7} \exp\left(\frac{[\mathrm{Na}^{+}]_o}{67.3}\right) - 1, \tag{A.30}$$

which is identical to Luo-Rudy model [36].

$I_{\mathrm{K}s}$, the Slow Component of the Delayed Rectifier K^{+} Current

$$I_{\mathrm{K}s} = \bar{G}_{\mathrm{K}s} x_{s1} x_{s2} (V - E_{\mathrm{K}s}), \tag{A.31}$$

$$E_{\mathrm{K}s} = (RT/F)\log((4.5 + P_{\mathrm{NaK}}150)/([\mathrm{K}^{+}]_i + P_{\mathrm{NaK}}[\mathrm{Na}^{+}]_o)) \quad {}^{\#}, \tag{A.32}$$

where the definition of $E_{\mathrm{K}s}$ equation (A.32) differs from the Viswanathan et al. (1999) [37]. The difference is in the term $[\mathrm{K}^{+}]_o = 4.5$ and $[\mathrm{Na}^{+}]_o = 150$, which is “hard-coded” inconsistent with the equations (A.4) (A.2) where $[\mathrm{K}^{+}]_o = 4.5$ and $[\mathrm{Na}^{+}]_o = 140$.

$$P_{\mathrm{NaK}} = 0.01833, \tag{A.33}$$

$$\bar{G}_{\mathrm{K}s} = (0.433(1 + 0.6/(1 + (0.000038/[\mathrm{Ca}^{2+}]_i)^{1.4}))) \cdot 0.615, \tag{A.34}$$

$$x_{s1\infty} = 1/(1 + \exp(-(V - 1.5)/16.7)), \tag{A.35}$$

$$x_{s2\infty} = x_{s1\infty}, \tag{A.36}$$

$$\tau_{xs1} = \left(\frac{7.19 \cdot 10^{-5}(V+30)}{1 - \exp(-0.148(V+30))} + \frac{1.31 \cdot 10^{-4}(V+30)}{\exp(0.0687(V+30)) - 1}\right)^{-1}, \tag{A.37}$$

$$\tau_{xs2} = 4\tau_{xs1}, \tag{A.38}$$

which is identical to Viswanathan et al. (1999) [37].

The gating variables are updated using Rush-Larsen method. In this appendix we have reconstructed corresponding dynamical equations for the gating variables as implied by the code. The gating variables for $I_{\mathrm{K}s}$ are

$$\frac{\mathrm{d}x_{s1}}{\mathrm{d}t} = \frac{x_{s1\infty} - x_{s1}}{\tau_{xs1}}, \tag{A.39}$$

$$\frac{\mathrm{d}x_{s2}}{\mathrm{d}t} = \frac{x_{s2\infty} - x_{s2}}{\tau_{xs2}}. \tag{A.40}$$

$I_{\mathrm{K}r}$, the Fast Component of the Delayed Rectifier K^+ Current

$$I_{\mathrm{K}r} = \bar{G}_{\mathrm{K}r} X_r R_{\mathrm{K}r}(V - E_{\mathrm{K}r}), \tag{A.41}$$

$$\bar{G}_{\mathrm{K}r} = 0.02614\sqrt{[\mathrm{K}^+]_o/5.4}, \tag{A.42}$$

$$X_{r\infty} = 1/(1 + \exp(-(V + 21.5)/7.5)), \tag{A.43}$$

$$R_{\mathrm{K}r} = 1/(1 + \exp((V + 9)/22.4)), \tag{A.44}$$

$$E_{\mathrm{K}r} = ((RT)/F)\log([\mathrm{K}^+]_o/[\mathrm{K}^+]_i), \tag{A.45}$$

$$\tau_{xr} = \left(0.00138\frac{V + 14.2}{1 - \exp(-0.123(V + 14.2))} + 0.00061\frac{V + 38.9}{\exp(0.145(V + 38.9)) - 1}\right)^{-1}, \tag{A.46}$$

which is identical to Zeng et al. (1995) [38]. The original notation for $R_{\mathrm{K}r}$ is R (here the R is used for the gas constant). The gating variable is

$$\frac{\mathrm{d}X_r}{\mathrm{d}t} = \frac{X_{r\infty} - X_r}{\tau_{xr}}. \tag{A.47}$$

Time-independent K^+ current: I_{K1}

$$I_{\mathrm{K1}} = \bar{G}_{\mathrm{K1}}\mathrm{K1}_\infty(V - E_{\mathrm{K1}}), \tag{A.48}$$

$$E_{\mathrm{K1}} = (RT/F)\log([\mathrm{K}^+]_o/[\mathrm{K}^+]_i), \tag{A.49}$$

$$\bar{G}_{\mathrm{K1}} = 0.75 \cdot \sqrt{([\mathrm{K}^+]_o/5.4)}, \tag{A.50}$$

$$\alpha_{\mathrm{K1}} = 1.02/(1 + \exp(0.2385(V - E_{\mathrm{K1}} - 59.215))), \tag{A.51}$$

$$\beta_{\mathrm{K1}} = \frac{0.49124\exp(0.08032(V - E_{\mathrm{K1}} + 5.476)) + \exp(0.06175(V - E_{\mathrm{K1}} - 594.31))}{1 + \exp(-0.5143(V - E_{\mathrm{K1}} + 4.753))}, \tag{A.52}$$

which is identical to Luo-Rudy model [36].

The gating variable is described by algebraic relation as

$$\mathrm{K1}_\infty = \alpha_{\mathrm{K1}}/(\alpha_{\mathrm{K1}} + \beta_{\mathrm{K1}}). \tag{A.53}$$

Plateau K^+ current: $I_{\mathrm{K}p}$

$$I_{\mathrm{K}p} = 0.00552K_p(V - E_{\mathrm{K1}}), \tag{A.54}$$

$$K_p = 1/(1 + \exp((7.488 - V)/5.98)), \tag{A.55}$$

which is equivalent to Luo-Rudy [36] with update from Zeng et al. (1995) [38], and

$$i_{\mathrm{K}} = I_{\mathrm{K1}} + I_{\mathrm{K}p}. \tag{A.56}$$

Currents through the L-type Ca^{+2} channel $I_{\mathrm{Ca}L}$

$$
\begin{aligned}
I_{\mathrm{Ca}L} =& I_{\mathrm{Ca}} + I_{\mathrm{CaK}} + I_{\mathrm{CaNa}}, && \text{(A.57)}\\
I_{\mathrm{Ca}} =& df f_{\mathrm{Ca}} \bar{I}_{\mathrm{Ca}}, && \text{(A.58)}\\
I_{\mathrm{CaK}} =& df f_{\mathrm{Ca}} \bar{I}_{\mathrm{CaK}}, && \text{(A.59)}\\
I_{\mathrm{CaNa}} =& df f_{\mathrm{Ca}} \bar{I}_{\mathrm{CaNa}}, && \text{(A.60)}\\
\bar{I}_{\mathrm{Ca}} =& P_{\mathrm{Ca}} z_{\mathrm{Ca}}^2 \frac{(VF^2)}{RT} \cdot \frac{\gamma_{\mathrm{Ca}i}[\mathrm{Ca}^{2+}]_i \exp((z_{\mathrm{Ca}}VF)/(RT)) - \gamma_{\mathrm{Ca}o}[\mathrm{Ca}^{2+}]_o}{\exp((z_{\mathrm{Ca}}VF)/(RT)) - 1}, && \text{(A.61)}\\
\bar{I}_{\mathrm{CaNa}} =& P_{\mathrm{Na}} z_{\mathrm{Na}}^2 \frac{(VF^2)}{RT} \cdot \frac{\gamma_{\mathrm{Na}i}[\mathrm{Na}^{+}]_i \exp((z_{\mathrm{Na}}VF)/(RT)) - \gamma_{\mathrm{Na}o}[\mathrm{Na}^{+}]_o}{\exp((z_{\mathrm{Na}}VF)/(RT)) - 1}, && \text{(A.62)}\\
\bar{I}_{\mathrm{CaK}} =& P_{\mathrm{K}} z_{\mathrm{K}}^2 \frac{(VF^2)}{RT} \cdot \frac{\gamma_{\mathrm{K}i}[\mathrm{K}^{+}]_i \exp((z_{\mathrm{K}}VF)/(RT)) - \gamma_{\mathrm{K}o}[\mathrm{K}^{+}]_o}{\exp((z_{\mathrm{K}}VF)/(RT)) - 1}, && \text{(A.63)}\\
P_{\mathrm{Ca}} =& 5.4{\cdot}10^{-4} \qquad \gamma_{\mathrm{Ca}i} = 1 \qquad \gamma_{\mathrm{Ca}o} = 0.341, && \text{(A.64)}\\
P_{\mathrm{Na}} =& 6.75{\cdot}10^{-7} \qquad \gamma_{\mathrm{Na}i} = 0.75 \qquad \gamma_{\mathrm{Na}o} = 0.75, && \text{(A.65)}\\
P_{\mathrm{K}} =& 1.93{\cdot}10^{-7} \qquad \gamma_{\mathrm{K}i} = 0.75 \qquad \gamma_{\mathrm{K}o} = 0.75, && \text{(A.66)}\\
f_{\mathrm{Ca}} =& 1/(1 + [\mathrm{Ca}^{2+}]_i / K_{m\mathrm{Ca}}), && \text{(A.67)}\\
K_{m\mathrm{Ca}} =& 0.0006, && \text{(A.68)}\\
d_{\infty} =& 1/(1 + \exp(-(V + 10)/6.24)), && \text{(A.69)}\\
\tau_d =& d_{\infty}(1 - \exp(-(V + 10)/6.24))/(0.035(V + 10)), && \text{(A.70)}\\
f_{\infty} =& (1/(1 + \exp((V + 32)/8))) + (0.6/(1 + \exp((50 - V)/20))) \quad {}^{\#}, && \text{(A.71)}\\
\tau_f =& 1/(0.0197 \exp(-(0.0337(V + 10)^2)) + 0.02), && \text{(A.72)}
\end{aligned}
$$

where the (A.71) differs from papers in expression from the code

$$\exp((V + 32)/8),$$

which is

$$\exp((V + 32)/8.6)$$

in Luo, Rudy (1994) [36]. The remaining equations are exactly the same as in Luo-Rudy model [36]. The constants are

$$
\begin{aligned}
z_{\mathrm{Na}} =& 1, && \text{(A.73)}\\
z_{\mathrm{K}} =& 1, && \text{(A.74)}\\
z_{\mathrm{Ca}} =& 2, && \text{(A.75)}
\end{aligned}
$$

and the gating variables are

$$\frac{\mathrm{d}d}{\mathrm{d}t} = \frac{d_\infty - d}{\tau_d}, \tag{A.76}$$

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{f_\infty - f}{\tau_f}. \tag{A.77}$$

Ca^{2+} **Current Through T-Type** Ca^{2+} **Channels** $I_{\mathrm{Ca}(T)}$ [38]

$$I_{\mathrm{Ca}(T)} = \bar{G}_{\mathrm{Ca}(T)} b^2 g (V - E_{\mathrm{Ca}}), \tag{A.78}$$

$$\bar{G}_{\mathrm{Ca}(T)} = 0.05, \tag{A.79}$$

$$b_\infty = 1/(1 + \exp(-(V + 14)/10.8)), \tag{A.80}$$

$$g_\infty = 1/(1 + \exp((V + 60)/5.6)), \tag{A.81}$$

$$E_{\mathrm{Ca}} = (RT/(2F)) \log([\mathrm{Ca}^{2+}]_o/[\mathrm{Ca}^{2+}]_i), \tag{A.82}$$

$$\tau_b = 3.7 + 6.1/(1 + \exp((V + 25)/4.5)), \tag{A.83}$$

$$\tau_g = -0.875V + 12 \text{ for: } V \leq 0\text{; and } \tau_g = 12 \text{ for: } V > 0, \tag{A.84}$$

which is identical to Zeng et al. (1995) [38], and the gating variables are

$$\frac{\mathrm{d}b}{\mathrm{d}t} = \frac{b_\infty - b}{\tau_b}, \tag{A.85}$$

$$\frac{\mathrm{d}g}{\mathrm{d}t} = \frac{g_\infty - g}{\tau_g}. \tag{A.86}$$

Na^+-Ca^+ **exchanger:** I_{NaCa}

$$I_{\mathrm{NaCa}} = \frac{2.5{\cdot}10^{-4} \exp((\eta - 1)V\frac{F}{RT}) \exp(V\frac{F}{RT})[\mathrm{Na}^+]_i^3[\mathrm{Ca}^{2+}]_o - [\mathrm{Na}^+]_o^3[\mathrm{Ca}^{2+}]_i}{1 + 1{\cdot}10^{-4} \exp((\eta - 1)V\frac{F}{RT})(\exp(V\frac{F}{RT})[\mathrm{Na}^+]_i^3[\mathrm{Ca}^{2+}]_o + [\mathrm{Na}^+]_o^3[\mathrm{Ca}^{2+}]_i)} \quad {}^{\#}, \tag{A.87}$$

$$\eta = 0.15 \quad {}^{\#}, \tag{A.88}$$

where the definition of I_{NaCa} in the Luo-Rudy model [36] depends on concentrations $[\mathrm{Ca}^{2+}]_o, [\mathrm{Na}^+]_o$ only, whereas in this model it depends also on intracellular concentrations $[\mathrm{Na}^+]_i, [\mathrm{Ca}^{2+}]_i$.

Nonspecific Ca^{2+}**-activated current:** $I_{ns(\mathrm{Ca})}$

$$\bar{I}_{ns\mathrm{K}} = 1.75{\cdot}10^{-7} \frac{VF^2}{RT} \cdot \frac{0.75[\mathrm{K}^+]_i \exp((VF)/(RT)) - 0.75[\mathrm{K}^+]_o}{\exp(VF/(RT)) - 1}, \tag{A.89}$$

$$I_{ns\mathrm{K}} = \bar{I}_{ns\mathrm{K}} \frac{1}{1 + (0.0012/[\mathrm{Ca}^{2+}]_i)^3}, \tag{A.90}$$

$$\bar{I}_{ns\mathrm{Na}} = 1.75{\cdot}10^{-7}\frac{VF^2}{RT} \cdot \frac{0.75[\mathrm{Na}^+]_i \exp((VF)/(RT)) - 0.75[\mathrm{Na}^+]_o}{\exp(VF/(RT)) - 1}, \tag{A.91}$$

$$I_{ns\mathrm{Na}} = \bar{I}_{ns\mathrm{Na}}\frac{1}{1 + (0.0012/[\mathrm{Ca}^{2+}]_i)^3,} \tag{A.92}$$

$$I_{ns(\mathrm{Ca})} = I_{ns\mathrm{K}} + I_{ns\mathrm{Na}}, \tag{A.93}$$

$$P_{ns(\mathrm{Ca})} = 1.75{\cdot}10^{-7}, \tag{A.94}$$

which is identical to Luo-Rudy model [36].

Sarcolemmal Ca^{+2} pump: $I_{p(\mathrm{Ca})}$

$$I_{p(\mathrm{Ca})} = 1.15\frac{[\mathrm{Ca}^{2+}]_i}{0.0005 + [\mathrm{Ca}^{2+}]_i}, \tag{A.95}$$

which is identical to Luo-Rudy model [36].

Ca^{+2} background current: $I_{\mathrm{Ca}b}$

$$I_{\mathrm{Ca}b} = 0.003016(V - E_{\mathrm{Ca}}), \tag{A.96}$$

$$E_{\mathrm{Ca}} = RT/(2F)\log([\mathrm{Ca}^{2+}]_o/[\mathrm{Ca}^{2+}]_i), \tag{A.97}$$

which is identical to Luo-Rudy model [36].

Na^{+} background current: $I_{\mathrm{Na}b}$

$$I_{\mathrm{Na}b} = 0.00141(V - E_{\mathrm{Na}}), \tag{A.98}$$

which is identical to Luo-Rudy model [36].

Ca^{2+} uptake and leakage of NSR: I_{up} and I_{leak}

$$I_{up} = 0.00875[\mathrm{Ca}^{2+}]_i/([\mathrm{Ca}^{2+}]_i + 0.00092), \tag{A.99}$$

$$K_{leak} = 0.005/15, \tag{A.100}$$

$$I_{leak} = K_{leak}[\mathrm{Ca}^{2+}]_{\mathrm{NSR}}, \tag{A.101}$$

where the definition of I_{up} in the Luo, Rudy (1994) [36] is ambiguous. This version is consistent with one possible understanding.

Ca^{+2} Fluxes in NSR

$$\frac{\mathrm{d}[\mathrm{Ca}^{2+}]_{\mathrm{NSR}}}{\mathrm{d}t} = (I_{up} - I_{leak} - I_{tr} V_{\mathrm{JSR}}/V_{\mathrm{NSR}}) \tag{A.102}$$

Ca^{2+} Fluxes in Myoplasm

$$I_{t\mathrm{Ca}} = I_{\mathrm{Ca}} + I_{\mathrm{Ca}b} + I_{p(\mathrm{Ca})} - 2I_{\mathrm{NaCa}} + I_{\mathrm{Ca}(T)} \tag{A.103}$$

$$\begin{aligned}\Delta[\mathrm{Ca}^{2+}]_i = & -\Delta t\Big(((I_{t\mathrm{Ca}} A_{Cap})/(V_{myo} 2F)) + ((I_{up} - I_{leak}) V_{\mathrm{NSR}}/V_{myo}) - \\ & - (I_{rel} V_{\mathrm{JSR}}/V_{myo})\Big)\end{aligned} \tag{A.104}$$

$$[\mathrm{Ca}^{2+}]_{ion} = \mathrm{TRPN} + \mathrm{CMDN} + \Delta[\mathrm{Ca}^{2+}]_i + [\mathrm{Ca}^{2+}]_i \tag{A.105}$$

$$B = 0.05 + 0.07 - [\mathrm{Ca}^{2+}]_{ion} + 0.0005 + 0.00238 \tag{A.106}$$

$$\begin{aligned}C = & (0.00238 \cdot 0.0005) - ([\mathrm{Ca}^{2+}]_{ion}(0.0005 + 0.00238)) + \\ & + (0.07 \cdot 0.00238) + (0.05 \cdot 0.0005)\end{aligned} \tag{A.107}$$

$$D = -0.0005 \cdot 0.00238 [\mathrm{Ca}^{2+}]_{ion} \tag{A.108}$$

$$F_{ab} = \sqrt{(B^2 - 3C)} \tag{A.109}$$

$$[\mathrm{Ca}^{2+}]_i = 1.5 F_{ab} \cos(\arccos((9BC - 2B^3 - 27D)/(2(B^2 - 3C)^{1.5}))/3) - (B/3) \tag{A.110}$$

Ca^{2+} Fluxes in JSR

$$\Delta[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} = \Delta t (I_{tr} - I_{rel}) \tag{A.111}$$

$$b_{\mathrm{JSR}} = 10 - \mathrm{CSQN} - \Delta[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} - [\mathrm{Ca}^{2+}]_{\mathrm{JSR}} + 0.8 \tag{A.112}$$

$$c_{\mathrm{JSR}} = 0.8(\mathrm{CSQN} + \Delta[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} + [\mathrm{Ca}^{2+}]_{\mathrm{JSR}}) \tag{A.113}$$

$$[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} = (\sqrt{(b_{\mathrm{JSR}}^2 + 4c_{\mathrm{JSR}})} - b_{\mathrm{JSR}})/2 \tag{A.114}$$

Sodium Ion Fluxes

$$I_{t\mathrm{Na}} = i_{\mathrm{Na}} + I_{\mathrm{Na}b} + I_{\mathrm{CaNa}} + I_{ns\mathrm{Na}} + 3I_{\mathrm{NaK}} + 3I_{\mathrm{NaCa}} \tag{A.115}$$

$$\frac{\mathrm{d}[\mathrm{Na}^+]_i}{\mathrm{d}t} = -(I_{t\mathrm{Na}} A_{Cap})/(V_{myo} F) \tag{A.116}$$

Potassium Ion Fluxes

$$I_{t\mathrm{K}} = I_{\mathrm{K}r} + I_{\mathrm{K}s} + i_{\mathrm{K}} + I_{\mathrm{CaK}} + I_{ns\mathrm{K}} - 2I_{\mathrm{NaK}} + I_{to} + I_{st} \tag{A.117}$$

$$\frac{\mathrm{d}[\mathrm{K}^+]_i}{\mathrm{d}t} = -(I_{t\mathrm{K}} A_{Cap})/(V_{myo} F) \tag{A.118}$$

Ca^{2+} buffers in the myoplasm

$$\mathrm{TRPN} = 0.07[\mathrm{Ca}^{2+}]_i/([\mathrm{Ca}^{2+}]_i + 0.0005), \tag{A.119}$$

$$\mathrm{CMDN} = 0.05[\mathrm{Ca}^{2+}]_i/([\mathrm{Ca}^{2+}]_i + 0.00238), \tag{A.120}$$

which is identical to Luo-Rudy model [36].

Ca^{2+} buffer in JSR and SCQN

$$\mathrm{CSQN} = 10([\mathrm{Ca}^{2+}]_{\mathrm{JSR}}/([\mathrm{Ca}^{2+}]_{\mathrm{JSR}} + 0.8)), \tag{A.121}$$

which is identical to Luo-Rudy model [36].

CICR From Junctional SR (JSR)

$$I_{rel} = G_{rel}\mathrm{ryr}_{open}\mathrm{ryr}_{close}([\mathrm{Ca}^{2+}]_{\mathrm{JSR}} - [\mathrm{Ca}^{2+}]_i), \tag{A.122}$$

$$G_{rel} = 150/(1 + \exp(I_{t\mathrm{Ca}} + 5)/0.9), \tag{A.123}$$

$$\mathrm{ryr}_{open} = 1/(1 + \exp((-t_c + 4)/0.5)), \tag{A.124}$$

$$\mathrm{ryr}_{close} = 1 - (1/(1 + \exp((-t_c + 4)/0.5))), \tag{A.125}$$

where variables $\mathrm{ryr}_{open} = 1 - \mathrm{ryr}_{close}$ ensure, that the channel is open at the time interval around 4 ms after the $\frac{\mathrm{d}V}{\mathrm{d}t}$ reaches its maximum (at the upstroke of the action potential). This is done using additional time variable t_c which is linked to the t and is reset to zero at the time when the $\frac{\mathrm{d}V}{\mathrm{d}t}$ reaches significant maximum, that is greater than 1.

This mathematical description can be interpreted as a delayed release of calcium in a short time interval at about 4 ms after the onset of action potential, when the $\mathrm{ryr}_{open}\mathrm{ryr}_{close}$ peaks. In the Viswanathan et al. (1999) [37] and Luo-Rudy model the delay of calcium release was around 2 ms after the the time of the maximum $\frac{\mathrm{d}V}{\mathrm{d}t}$.

The delay variable t_c as described in the papers and implemented in the code causes that the system is not a system of differential equations. For this reason the implementation into BeatBox as `rhs` or `ionic` model is not straightforward, and has not been done.

Translocation of Ca^{2+} ions from NSR to JSR: I_{tr}

$$I_{tr} = ([\mathrm{Ca}^{2+}]_{\mathrm{NSR}} - [\mathrm{Ca}^{2+}]_{\mathrm{JSR}})/180, \tag{A.126}$$

which is identical to Luo-Rudy model [36].

Total time-independent current: I_v

$$I_v = I_{\mathrm{Na}b} + I_{\mathrm{NaK}} + I_{p(\mathrm{Ca})} + I_{\mathrm{K}p} + I_{\mathrm{Ca}b} + I_{\mathrm{K1}}, \tag{A.127}$$

which is identical to Luo-Rudy model [36].

Total Current

$$\begin{aligned} I_t =& I_{\mathrm{K}r} + I_{\mathrm{K}s} + i_{\mathrm{K}} + I_{\mathrm{CaK}} + I_{ns\mathrm{K}} - 2I_{\mathrm{NaK}} + i_{\mathrm{Na}} + I_{\mathrm{Na}b} + I_{\mathrm{CaNa}} + I_{ns\mathrm{Na}} + 3I_{\mathrm{NaK}} + \\ & 3I_{\mathrm{NaCa}} + I_{\mathrm{Ca}} + I_{\mathrm{Ca}b} + I_{p(\mathrm{Ca})} - 2I_{\mathrm{NaCa}} + I_{\mathrm{Ca}(T)} \end{aligned} \tag{A.128}$$

Membrane Potential

$$\frac{\mathrm{d}V}{\mathrm{d}t} = - I_t \tag{A.129}$$

Appendix B

Eigenvalue Computation

B.1 Overview of Subroutines for Finding Eigenvalues

Matrix Rush-Larsen method requires to obtain the eigenvalues and eigenvector of the transition matrix. Text books on numerical analysis recommend to use the established packages for solution of eigenvalue problem, rather than developing our version [39].

A large number of libraries provide subroutines to solve eigenvalue problem [40, 41]. Table B.1 shows some examples which satisfy the copyright conditions of BeatBox GNU GPL license. The prerequisite of all those packages is a library implementing Basic Linear Algebra Subprograms BLAS and many also require LAPACK.

Table B.1: Numerical libraries for solving eigenvalue problem (based on [41])

Package	Dependencies	License	Language	Size
ARPACK	BLAS	BSD	Fortran	664KB
FEAST	LAPACK/BLAS	BSL	C/Fortran	5.5MB
GSL	BLAS	GNU GPL	C	1.4MB
FILTLAN	MATKIT/LAPACK	GNU LGPL	C/C++	1.6MB
PRIMME	BLAS/LAPACK	GNU LGPL	C/Fortran	4.5MB
PROPACK	BLAS/LAPACK	BSD	Fortran/Matlab	49MB

B.2 Linear Algebra Package LAPACK

B.2.1 Overview of LAPACK

LAPACK is a standard software library containing a large collection of subroutines for linear algebra – with the eigenvector solution amongst them. The LAPACK is distributed under the terms of the BSD license, that grants the rights to distribute the software freely, and even allows the modified version to be redistributed under different conditions (re-licensed) provided that a proper credit is given to the authors of LAPACK.

The library is written in Fortran 90. If we program in other languages the implementation of LAPACK functions is not straightforward due to the differences between the languages, e.g. structures of data, ways scalar values are passed to the functions. The C programmes using LAPACK library have to use an interface which “wraps” the LAPACK functions for C.

A popular LAPACK C interface called lapacke is already included in LAPACK installation. So, the C code can use LAPACK function through lapacke interface, and be linked against lapacke precompiled lapacke libraries at the compilation. However, to install the LAPACK a Fortran compiler is required, which might not be available on the target system.

The LAPACK library depends on subroutines of Basic Linear Algebra Subprograms (BLAS). than a particular software package BLAS refers to a standard interface for the implementation of those subprograms and there are various BLAS implementation such as ATLAS, GoToBLAS, CBLAS or the official Netlib BLAS (often shorten as BLAS).

The Netlib BLAS package comes packed along the LAPACK distribution, however it should be used only when there is no other implementation of BLAS on the intended machine. This is because, the preinstalled libraries are normally more optimised for the particular computer architecture and as the efficiency of LAPACK depends on the efficiency of BLAS implementation.

B.2.2 Standalone LAPACK Code for Eigenvalue Computation

We have implemented standalone code for eigenvalue computation of I_{Na} Markov chain model. For the specification of the transition rates matrices we use the same macros as described on page 168 and in Figure 6.3 The code of main file *inaEigenLAPACK.c* follows

```
/**
 *  Copyright 2015 Vadim Biktashev, Tomas Stary
 *
 *  Free software under GNU GPLv3.
 *  See <http://www.gnu.org/licenses/>.
 */
```

```
#include <stdio.h>
#include <lapacke.h>
#include <math.h>
#include "inaEigen.h"

/* dimension of the system */
#define LDA     DIM
#define LDVL    DIM
#define LDVR    DIM

int
main (int argc, const char * argv[] )
{
  /* initialize lapack variables */
  /* ld stands for leading dimension of an array -- for example it
     would be 20 for an array (20, 10) */
  const int     n = DIM, lda = LDA, ldvl = LDVL, ldvr = LDVR;
  int           lapack_exit_status;

  /****************************************/
  /* create array and allocate the memory */
  /* transition rates matrix of INa */
  MAKE_ARRAY(double, trans_rates_matrix, DIM*DIM);
  /* real part of eigenvalues */
  MAKE_ARRAY(double, eval_real, DIM);
  /* imaginary part of eigenvalues */
  MAKE_ARRAY(double, eval_imag, DIM);
  /* left eigenvectors */
  MAKE_ARRAY(double, evec_left, LDVL*DIM);
  /* right eigenvectors */
  MAKE_ARRAY(double, evec_right, LDVR*DIM);

  /**********************************************/
  /* create pointers and open the output files  */
  /* name of output file */
  MAKE_ARRAY(char, filename, 64);
  /* file for voltage */
  OPEN_FILE(fvolt, FVM, "LAPACK");
  /* file for eigenvalues */
  OPEN_FILE(feval, FEVINA, "LAPACK");
  /* file for left eigenvectors */
  OPEN_FILE(fevec_left, FLEVINA, "LAPACK");
  /* file for right eigenvectors */
  OPEN_FILE(fevec_right, FREVINA, "LAPACK");
  free(filename);

  /* membrane potential in mV */
  double volt;
  /* number of voltage steps */
  const int volt_steps_N = (( VMAX - VMIN )/DV);

  int   i;
  for (i = 0;i <= volt_steps_N;i++ )
    {/* Membrane potential loop */
      /* get membrane potential */
      volt = VMIN+i*DV;
      /* get transition rates matrix */
      ina_trans_rates_matrix(volt, trans_rates_matrix);
      /* calculate the eigenvalues and right and left
         eigenvectors */
      lapack_exit_status =
        LAPACKE_dgeev(LAPACK_ROW_MAJOR, 'V', 'V',
                      n, trans_rates_matrix, lda,
                      eval_real, eval_imag, evec_left,
                      ldvl, evec_right, ldvr);
      /* Check for convergence */
      if( lapack_exit_status != 0 )
        ERROR("Eigenvalue␣computation␣failed.\n");
      /* write results */
      fprintf(fvolt, "%.2f\n", volt);
      WRITE_EVAL(feval, n, eval_real, eval_imag);
      WRITE_EVEC(fevec_left, n, eval_imag, evec_left);
      WRITE_EVEC(fevec_right, n, eval_imag, evec_right);
    }   /* end of membrane potential loop */
  /* free memory */
  free(trans_rates_matrix);
  free(eval_imag);
  free(evec_left);
  free(evec_right);

  /* close files */
  fclose(fvolt);
  fclose(feval);
  fclose(fevec_left);
  fclose(fevec_right);

  return 0;
}
```

where the header file *inaEigen.h* is

```
/**
 *  Copyright 2015 Vadim Biktashev, Tomas Stary
 *
 *  Free software under GNU GPLv3.
 *  See <http://www.gnu.org/licenses/>.
 */

/* voltage range  */
#define VMIN    -100.0
#define DV      0.01
#define VMAX    70.0

/* file with values of voltage */
#define FVM     "dat/vm_INa_%s.dat"
/* file with eigen values */
#define FEVINA  "dat/evals_INa_%s.dat"
/* file with left eigenvectors */
#define FLEVINA "dat/left_evecs_INa_%s.dat"
/* file with right eigenvectors */
#define FREVINA "dat/right_evecs_INa_%s.dat"

#define ERROR(msg){fprintf(stderr,msg);exit(1);}
#define CALLOC(p,a,b)                                   \
  if(0==(p=calloc(a,b)))ERROR("not␣enough␣memory\n")
#define MAKE_ARRAY(type, name, length)                  \
  type * name; CALLOC(name, length, sizeof(type));
#define OPEN_FILE(fileid, name,suffix)                     \
  FILE * fileid;                                           \
  sprintf(filename,name,suffix);                           \
  if ( ( fileid = fopen(filename,"w")) == NULL){           \
    fprintf(stderr,"Error␣while␣openning␣the␣file:␣%s.\n", \
            filename);                                     \
    exit(1);}
#define WRITE_EVAL(fileid, dimension, eval_Re, eval_Im) {\
    int ii;                                             \
    for( ii = 0; ii < dimension; ii++ ) {               \
      if( eval_Im[ii] == (double)0.0 ) {                \
        fprintf(fileid, "␣%.10e", eval_Re[ii] );        \
      } else {                                          \
        ERROR("The␣imaginary␣part␣is␣not␣zero.\n");     \
      }                                                 \
      /* separators */                                  \
      (ii < (dimension - 1)) ?                          \
        fprintf (fileid, "\t") :                        \
        fprintf (fileid, "\n");                         \
    }                                                   \
  }
#define WRITE_EVEC(fileid, dimension, eval_Im, evec) {      \
    int ii, jj;                                             \
    for( jj = 0; jj < dimension; jj++ ) {                   \
      ii      = 0;                                          \
      while( ii < dimension ) {                             \
        if( eval_Im[ii] == (double)0.0 ) {                  \
          fprintf(fileid, "%.10e", evec[jj*dimension+ii] ); \
          ii++;                                             \
        } else {                                            \
          ERROR("The␣imaginary␣part␣is␣not␣zero.\n");       \
        }                                                   \
        /* separators */                                     \
        ((jj * dimension + ii) < (dimension * dimension)) ? \
          fprintf (fileid, "\t") :                           \
          fprintf (fileid, "\n");                            \
      }                                                      \
    }                                                        \
  }

/* Enumerate the markov chain states */
enum
  {
    #define _(n,i) markov_##n,
    #include "clancy_markov.h"
    #undef _
    DIM /* total number of Markov variables */
  };

void
ina_trans_rates_matrix(double V, double *tr)
{
  /* Updates the transition rates matrix of INa Markov chain
     model published by Clancy, Rudy (2002) */
  /* input variables: V -- membrane voltage; tr -- pointer to
     the matrix */
  int i;
  for (i=0; i<DIM*DIM; i++)
    {
      /* reset entries */
      tr[i]=0;
    }
  /* recompute the tr matrix for new value of V */
  #define _VFUN(name,expression) double name=expression;
  #define _RATE(from,to,direct,reverse)         \
    tr[markov_##to*DIM+markov_##from]=direct;   \
    tr[markov_##from*DIM+markov_##from]-=direct;\
    tr[markov_##from*DIM+markov_##to]=reverse;  \
    tr[markov_##to*DIM+markov_##to]-=reverse;
  #include "clancy_rates.h"
  #undef _VFUN
  #undef _RATE
}
```

This section shows the code for computation of eigenvalues and eigenvectors of I_{Na} Markov chain model by Clancy and Rudy (2002).

```
/* ina mc states */
_(O,4.38587098159465e-08)
_(C1,5.32914708198526e-05)
_(C2,0.010642045025986)
_(C3,0.801808970977337)
_(IC3,0.143555231208206)
_(IC2,0.00190739109198479)
_(IC1,1.11107023501962e-05)
_(IM1,0.000841692031966022)
_(IM2,0.041180223632675)
/* impose state conservation law */
/* _(IM1,1-(O+C1+C2+C3+IC3+IC2+IC1+IM2)) */
```

```
/* Transition rates from Clancy, Rudy (2002) */
        /* C3->C2 (R->Q); IC3->IC2 (S->T) */
_VFUN(alpha11,3.802/(0.1027*exp(-V/17.0)+0.20*exp(-V/150.)))
        /* C2->C1 (Q->P); IC2->IF (T->U) */
_VFUN(alpha12,3.802/(0.1027*exp(-V/15.0)+0.23*exp(-V/150.)))
        /* C1->O (P->O) */
_VFUN(alpha13,3.802/(0.1027*exp(-V/12.0)+0.25*exp(-V/150.)))
        /* C2->C3 (Q->R); C2->C3 (T->S) */
_VFUN(beta11,0.1917*exp(-V/20.3))
        /* C1->C2 (P->Q); C1->C2 (U->T) */
_VFUN(beta12,0.20*exp(-(V-5)/20.3))
        /* O->C1 (O->P) */
_VFUN(beta13,0.22*exp(-(V-10)/20.3))
        /* IF->C1 (U->P); IC2->C2 (T->Q); IC3->C3 (S->R) */
```

```
_VFUN(alpha3,3.7933e-7*exp(-V/7.7))
        /* C1->IF (P->U); C2->IC2 (Q->T); C3->IC3 (R->S) */
_VFUN(beta3,8.4e-3+2e-5*V)
        /* O->IF (O->U) */
_VFUN(alpha2,9.178*exp(V/29.68))
        /* IF->O (U->O) */
_VFUN(beta2,(alpha13*alpha2*alpha3)/(beta13*beta3))
        /* IF->IM1 (U->V) */
_VFUN(alpha4,alpha2/100.)
        /* IM1->IF (V->U) */
_VFUN(beta4,alpha3)
        /* IM1->IM2 (V->W) */
_VFUN(alpha5,alpha2/(9.5e4))
        /* IM2->IM1 (W->V) */
_VFUN(beta5,alpha3/50.)

_RATE(O,C1,beta13,alpha13)
_RATE(C1,C2,beta12,alpha12)
_RATE(C2,C3,beta11,alpha11)
_RATE(C3,IC3,beta3,alpha3)
_RATE(IC3,IC2,alpha11,beta11)
_RATE(IC2,IC1,alpha12,beta12)
_RATE(IC1,IM1,alpha4,beta4)
_RATE(IM1,IM2,alpha5,beta5)
_RATE(IC2,C2,alpha3,beta3)
_RATE(IC1,C1,alpha3,beta3)
_RATE(IC1,O,beta2,alpha2)
```

B.3 GNU Scientific Library (GSL)

B.3.1 Overview of GSL

GNU Scientific Library is a collection of routines for numerical computing [42]. The GSL is released under GNU GPL license. The GNU GPL contains a copyleft clause which ensures, that any software using the library must remain under free license.

B.3.2 Standalone GSL Code

We have implemented the standalone code for the eigenvalue and eigenvector computations using GSL. This subsection shows the application of this code to I_{Na} Markov chain models. The code of main file *inaEigenGSL.c* follows

```
#include <stdio.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>
#include <gsl/gsl_blas.h>

/* voltage range  */
#define VMIN    -100.0
#define DV       0.01
#define VMAX     70.0

/* file with values of voltage */
#define FVM      "dat/vm_INa_%s.dat"
/* file with right eigen values */
#define FREVAL  "dat/right_evals_INa_%s.dat"
/* file with left eigen values */
#define FLEVAL  "dat/left_evals_INa_%s.dat"
/* file with left eigenvectors */
#define FLEVEC  "dat/left_evecs_INa_%s.dat"
/* file with right eigenvectors */
```

```
#define FREVEC  "dat/right_evecs_INa_%s.dat"

#define ERROR(msg){fprintf(stderr,msg);exit(1);}
#define CALLOC(p,a,b) if(0==(p = calloc(a,b)))ERROR("not␣enough␣memory\n")
#define MAKE_ARRAY(type, name, length) type * name; CALLOC(name, length, \
 sizeof(type));
#define OPEN_FILE(fileid, name, suffix);                                \
  FILE * fileid;                                                        \
  sprintf(filename,name,suffix);                                        \
  if ( ( fileid = fopen(filename,"w")) == NULL)                         \
    {                                                                   \
      fprintf(stderr,"Error␣while␣openning␣the␣file:␣%s.\n",filename);  \
      exit(1);                                                          \
    }

#define ASSERT_VECTOR_NON_IMAG(vector_gsl);                     \
  {                                                             \
    gsl_vector_view vec_imag;                                   \
    double min_out, max_out;                                    \
    vec_imag = gsl_vector_complex_imag(vector_gsl);             \
    gsl_vector_minmax (&vec_imag.vector, &min_out, &max_out);   \
    if ( min_out != max_out || min_out != (double) 0.0  )       \
      ERROR("Non-zero␣imaginary␣part.\n");                      \
  }

/* This macro saves only real part of the matrices. */
/* there is not a gsl function for saving only real part of complex
   matrices, neither a complex_real conversion function as it is the
   case with vectors. */
#define GSL_MATRIX_REAL_FPRINTF(fileid, matrix, format);                \
  {                                                                     \
    gsl_vector_complex_view vec_column; /* vector column */             \
    gsl_vector_view vec_column_real;    /* real vector column */        \
    for(j = 0; j < matrix->size2; j++)                                  \
      {                                                                 \
        /* get vector number j, ... */                                  \
        vec_column = gsl_matrix_complex_column(matrix, j);              \
        /* ...convert it to real... */                                  \
        vec_column_real = gsl_vector_complex_real(&vec_column.vector);  \
        /* ...and save it. */                                           \
        gsl_vector_fprintf(fileid, &vec_column_real.vector,format);     \
        /* Assert non-imaginary parts in the matrix */                  \
        ASSERT_VECTOR_NON_IMAG(&vec_column.vector);                     \
      }                                                                 \
  }

#define MULTIPLY_ZGEMM(target, A, B);                           \
  info = gsl_blas_zgemm (CblasNoTrans, CblasNoTrans, alpha,     \
                         A, B, beta, target);                   \
  if ( info != 0 ) ERROR("Error␣in␣matrix␣multiplication");

/* Enumerate the markov chain states */
enum
{
#define _(n,i) markov_##n,
#include "clancy_markov.h"
#undef _
  DIM                           /* total number of Markov variables */
};

void
ina_rates_matrix (double V, gsl_matrix * matrix)
{
  /* Updates the transition rates matrix of INa Markov chain model
     published by Clancy, Rudy (2002) */
  /* V -- membrane voltage */
  if (matrix->size1 != matrix->size2 || matrix->size1 != DIM)
    ERROR ("Transition␣rates␣matrix␣is␣wrong.");
  /* reset the matrix */
  gsl_matrix_set_zero (matrix);
  /* recompute the tr matrix for new value of V */
#define _VFUN(name,expression) double name = expression;
#define _RATE(from,to,direct,reverse)                                   \
  matrix->data[markov_##to*matrix->tda+markov_##from]=direct;           \
  matrix->data[markov_##from*matrix->tda+markov_##from]-=direct;        \
  matrix->data[markov_##from*matrix->tda+markov_##to]=reverse;          \
  matrix->data[markov_##to*matrix->tda+markov_##to]-=reverse;
#include "clancy_rates.h"
#undef _VFUN
#undef _RATE
}

void
vector_to_matrix_diagonal (const gsl_vector_complex * vector,
                           gsl_matrix_complex * matrix)
{
  /* copy the diagonal vector elements to the diagonal entries of the matrix */
  /* assert the sizes of the matrices and vectors correspond */
  if ((matrix->size1 != matrix->size2) || (matrix->size1 != vector->size))
    {
      fprintf (stderr, "The␣matrix␣must␣be␣square␣of␣the␣size␣%d.\n",
               (int) vector->size);
      exit (1);
    }

  /* initialize elements to zero */
  gsl_matrix_complex_set_zero (matrix);
  /* copy the elements to the matrix diagonal */
  unsigned int i;
  for (i = 0; i < 2 * vector->size; i = i + 2)
    {
      matrix->data[i * matrix->tda + i] = vector->data[i * vector->stride];
      matrix->data[i * matrix->tda + i + 1] =
        vector->data[i * vector->stride + 1];
    }
}

void
gsl_matrix_real_complex (gsl_matrix_complex * dest, const gsl_matrix * source)
{
  /* convert real matrix into complex */
  if ((dest->size1 != source->size1) || (dest->size2 != source->size2))
    ERROR ("The␣matrix␣dimensions␣must␣correspond.\n");
  unsigned int i, j;
  for (i = 0; i < source->size1; i++)
    {
      for (j = 0; j < source->size2; j++)
        dest->data[(2 * i) * dest->tda + (2 * j)] =
          source->data[i * source->tda + j];
    }
}

void
eval_scale_identity (gsl_matrix_complex * target,
                     const gsl_matrix_complex * source)
{
  /* scale target matrix to give one after a multiplication with the
     source matrix. */
  /* after this operation on the left_evecs*right_evecs should
     approximate identity matrix */
  gsl_complex scale_factor, scale_factor_inverse;
  gsl_vector_complex_view vec_row;
  unsigned int i;
  for (i = 0; i < target->size1; i++)
    {
      /* get the corresponding vectors from matrices */
      vec_row = gsl_matrix_complex_row (target, i);
      gsl_vector_complex_const_view vec_column =
        gsl_matrix_complex_const_column (source, i);
      /* perform a vector multiplication */
      gsl_blas_zdotu (&vec_row.vector, &vec_column.vector,
                      &scale_factor_inverse);
      /* assert zero imaginary part */
      if (GSL_IMAG (scale_factor_inverse) != (double) 0.0)
        ERROR ("Unexpected␣non-zero␣imaginary␣part.\n");
      /* find scaling factor */
      GSL_SET_COMPLEX (&scale_factor, 1.0 / GSL_REAL (scale_factor_inverse),
                       0.0);
      /* scale the target vector */
      gsl_vector_complex_scale (&vec_row.vector, scale_factor);
    }
}
```

```
void
assert_vector_relat_diff (const gsl_vector * A, const gsl_vector * B,
                          const double tol, const char *object)
{
  /* compares vectors A and B by entries and release warning, if the
     difference is larger than tolerance tol. Variable object is used
     to specify the object for the error message. */
  if (A->size != B->size)
    ERROR ("The␣matrix␣dimensions␣must␣correspond.\n");

  /* alocate space for the intermediate calculations */
  gsl_vector *diff = gsl_vector_alloc (A->size);
  /* diff = A */
  gsl_vector_memcpy (diff, A);
  /* diff = A - B */
  gsl_vector_sub (diff, B);
  /* diff = (A - B)/B */
  /* gsl_vector_div(diff, B); */
  /* assert the enries of the diff are small */
  unsigned int ii;
  double entry;
  for (ii = 0; ii < diff->size; ii++)
    {
      entry = diff->data[ii * diff->stride];
      entry = (entry > 0.0) ? entry : -entry;
      if (entry > tol /* && (A->data[ii*A->stride])> 1e-10 */ )
        {
          fprintf (stderr, "Warning:␣%s␣absolute␣diffence", object);
          fprintf (stderr, "␣is␣%g\t␣for␣(%g,␣%g).\n", entry,
                   A->data[ii * A->stride], B->data[ii * B->stride]);
        }
    }
  gsl_vector_free (diff);
}

void
assert_reconst_matrix (gsl_matrix_complex * computed, gsl_matrix_complex * W,
                       gsl_vector_complex * lambda, gsl_matrix_complex * V)
{
  /* reconstruct original matrix from right_evec*diag(eval)*left_evec
     (W*diag(lambda)*V) and assert it is a good approximation of
     the matrix computed by definition */

  /* assert the dimensions agree */
  if ((W->size1 != W->size1) || (V->size1 != V->size2)
      || (computed->size1 != computed->size2)
      || (W->size1 != V->size1) || (W->size1 != computed->size1)
      || (V->size1 != lambda->size))
    ERROR ("The␣dimensions␣must␣correspond.\n");

  /* alocate reconstructed transition rates matrix of INa */
  gsl_matrix_complex *reconst =
    gsl_matrix_complex_alloc (computed->size1, computed->size2);
  gsl_matrix_complex *intermed =
    gsl_matrix_complex_alloc (computed->size1, computed->size2);

  /* allocate diagonal eigenvalue matrix...  */
  gsl_matrix_complex *D =
    gsl_matrix_complex_alloc (computed->size1, computed->size2);
  /* ...and fill it up with the eigenvalues on diagonal */
  vector_to_matrix_diagonal (lambda, D);

  /* matrix and vector views */
  gsl_vector_complex_view vec_reconst_row, vec_computed_row;
  gsl_vector_view vec_reconst_row_real, vec_computed_row_real;

  int info;                     /* variable used for exit status */
  /* blas input parameters */
  gsl_complex alpha, beta;

  GSL_SET_COMPLEX (&alpha, 1.0, 0.0);
  GSL_SET_COMPLEX (&beta, 0.0, 0.0);

  /* matrix multiplication (using blas) */
  MULTIPLY_ZGEMM (intermed, W, D);
  MULTIPLY_ZGEMM (reconst, intermed, V);

  /* assert that the reconstructed matrix approximates the computed */
  unsigned int i;
  for (i = 0; i < reconst->size1; i++)
    {
      vec_reconst_row = gsl_matrix_complex_row (reconst, i);
      vec_computed_row = gsl_matrix_complex_row (computed, i);
      /* assert non imag */
      ASSERT_VECTOR_NON_IMAG (&vec_reconst_row.vector);
      ASSERT_VECTOR_NON_IMAG (&vec_computed_row.vector);
      /* ...convert it to real... */
      vec_reconst_row_real =
        gsl_vector_complex_real (&vec_reconst_row.vector);
      vec_computed_row_real =
        gsl_vector_complex_real (&vec_computed_row.vector);
      /* ... warn if the difference of the entries is large. */
      assert_vector_relat_diff (&vec_computed_row_real.vector,
                                &vec_reconst_row_real.vector, 1e-5,
                                "Transition␣rates");
    }
  /* free memory */
  gsl_matrix_complex_free (reconst);
  gsl_matrix_complex_free (intermed);
  gsl_matrix_complex_free (D);
}

int
main (void)
{
      /******************************************************/
  /* create and allocate memory in gsl format structure */
  /* transition rates matrix of INa */
  gsl_matrix *rates_matrix = gsl_matrix_alloc (DIM, DIM);
  /* transposed transition rates matrix of INa */
  gsl_matrix *rates_matrix_transp = gsl_matrix_alloc (DIM, DIM);
  /* auxilary matrix for reconstructed transition rates matrix calculation */
  gsl_matrix_complex *rates_matrix_complex =
    gsl_matrix_complex_alloc (DIM, DIM);

  /* right eigenvalues  */
  gsl_vector_complex *eval_right = gsl_vector_complex_alloc (DIM);
  /* left eigenvalues */
  gsl_vector_complex *eval_left = gsl_vector_complex_alloc (DIM);
  /* right eigenvector matrix */
  gsl_matrix_complex *evec_right = gsl_matrix_complex_alloc (DIM, DIM);
  /* left eigenvector matrix */
  gsl_matrix_complex *evec_left = gsl_matrix_complex_alloc (DIM, DIM);
  /* workspace for nonsymetric eigenvalue problem */
  gsl_eigen_nonsymmv_workspace *workspace = gsl_eigen_nonsymmv_alloc (DIM);

  /* view to gsl structures */
  gsl_vector_view eval_right_real, eval_left_real;      /* real eigenvalues */

      /**********************************************/
  /* create pointers and open the output files  */
  MAKE_ARRAY (char, filename, 64);      /* name of output file */
  OPEN_FILE (fvolt, FVM, "GSL");         /* file for voltage */
  OPEN_FILE (feval_right, FREVAL, "GSL");        /* file for right eigenvalues */
  OPEN_FILE (fevec_left, FLEVEC, "GSL");         /* file for left eigenvectors */
  OPEN_FILE (fevec_right, FREVEC, "GSL");        /* file for right eigenvectors */
  free (filename);

      /********************************/
  /* COMPUTE AND SAVE THE RESULTS */
  /* number of voltage steps */
  const int volt_steps_N = (VMAX - VMIN) / DV;
  /* membrane potential in mV */
  double volt;
  /* loop counters */
  int i;
  unsigned int j;
  for (i = 0; i <= volt_steps_N; i++)
    {
      /* ********************************* */
      /* compute transition rates matrix   */
      /* get the membrane voltage */
      volt = VMIN + i * DV;
      /* get transition rates matrix for given voltage */
      ina_rates_matrix (volt, rates_matrix);
      /* get transposed transition rates matrix */
      gsl_matrix_transpose_memcpy (rates_matrix_transp, rates_matrix);
      /* copy the elements into complex transition rates matrix */
      gsl_matrix_real_complex (rates_matrix_complex, rates_matrix);

      /* ************************************ */
      /* compute eigenvalues and eigenvectors */
      /* get the RIGHT evals and evecs */
      gsl_eigen_nonsymmv (rates_matrix, eval_right, evec_right, workspace);
      /* get the LEFT evals and evecs */
      gsl_eigen_nonsymmv (rates_matrix_transp, eval_left, evec_left,
                          workspace);

      /* ****************************** */
      /* process eval and evec */
      /* sort the eigenvalues and eigenvectors in descending order */
      gsl_eigen_nonsymmv_sort (eval_right, evec_right,
                               GSL_EIGEN_SORT_ABS_DESC);
      gsl_eigen_nonsymmv_sort (eval_left, evec_left, GSL_EIGEN_SORT_ABS_DESC);
      /* transpose the left eigenvector matrix in place */
      gsl_matrix_complex_transpose (evec_left);
      /* scale left_evals to satisfy left_evals*right_evals = Identity */
      eval_scale_identity (evec_left, evec_right);

      /* create a view to real part of eigenvalues... */
      eval_right_real = gsl_vector_complex_real (eval_right);
      eval_left_real = gsl_vector_complex_real (eval_left);

      /* **************************************** */
      /* assert some assumed properties */
      /* ... warn if the difference of eigenvalues is large. */
      assert_vector_relat_diff (&eval_left_real.vector,
                                &eval_right_real.vector, 1e-10,
                                "Eigenvalues");
      /* non-imaginary parts in eigenvectors */
      ASSERT_VECTOR_NON_IMAG (eval_right);
      ASSERT_VECTOR_NON_IMAG (eval_left);
      /* assert the reconstructed matrix is a good approximation */
      assert_reconst_matrix (rates_matrix_complex, evec_right, eval_right,
                             evec_left);

      /* ****************************** */
      /*        Saving results          */
      /* Save real part of eigenvalues. */
      gsl_vector_fprintf (feval_right, &eval_right_real.vector, "%.10g");
      /* save real part of eigenvector matrices */
      GSL_MATRIX_REAL_FPRINTF (fevec_left, evec_left, "%.10g");
      GSL_MATRIX_REAL_FPRINTF (fevec_right, evec_right, "%.10g");
      /* save the voltage */
      fprintf(fvolt, "%.2f\n", volt);
    }

      /************/
  /* CLEAN UP */
  /* close files */
  fclose (fvolt);
  fclose (feval_right);
  fclose (fevec_left);
  fclose (fevec_right);

  /* free memory */
  gsl_vector_complex_free (eval_right);
  gsl_vector_complex_free (eval_left);
  gsl_matrix_free (rates_matrix);
  gsl_matrix_free (rates_matrix_transp);
  gsl_matrix_complex_free (rates_matrix_complex);
  gsl_matrix_complex_free (evec_right);
  gsl_matrix_complex_free (evec_left);
  gsl_eigen_nonsymmv_free (workspace);

  return 0;
}
```

This code uses *inaEigen.h*, *clancy_markov.h*, and *clancy_rates.h* are shared with the LAPACK code and were listed in the previous section.

B.4 Including GSL to BeatBox

Both LAPACK and GSL implementation have been successful in diagonalisation of the matrices of I_{Na} Markov chain. The computation was done for membrane voltages from -100, to 70 mV with the grid step of 0.01 mV. The computational time including saving of the eigenvalues and eigenvector matrices is 1.8 s for the GSL implementation and 1.5 s for LAPACK implementation.

The license of LAPACK is permissible, and allows including the library even in non-free (proprietary) software. The GSL GNU GPL license requires the software to be free. So both packages could be possibly included into BeatBox, which itself is released under free GNU GPL license.

We have not compared the the efficiency of the computation of the diagonalisation of the matrix.

Eventually, we have opted to include the GSL library. Although, the LAPACK computation seems more efficient, the efficiency of the diagonalisation is not crucial for the speed of the integration. This is because we normally do not compute the decomposition on-the-fly, but instead use tabulation of the eigenvalues and eigenvectors before the main computation starts. Using the LAPACK would imply the user to have Fortran compiler in addition to the C compiler, which could be inconvenient for some users.

The complete build of GSL library takes several minutes. However, many of the function provided by the library, are not required for the diagonalisation. For that reason, we decided to include only required GSL functions.

The extract of the required functions from GSL was not a trivial step. The main complication in the process is that many of the files define a great number of functions, which themselves refer to additional functions, which are sometimes present in other files. When that happens, the object code for the unnecessary functions has to be included the binary, although the specific function is actually never used.

To overcome this problem, we have decided to include only the required functions and comment out all the irrelevant function. To find out which function are those, we have developed a script, which would search for all the functions used within the eigenvalue solver and its called function.

In the end we identify a list of about 60 essential functions and commented out the remaining part of the files. Also the files, which were not used at all were removed from the build system. Such extracted GSL library can be compiled within a few seconds and is included as part of BeatBox distribution.

Similarly to BeatBox source code GSL uses GNU Build system, which is a name for a set of tools used to compile the source code and allow portability of the code to machines with different architecture. The standard for the GNU packages requires to provide *configure* and *Makefile* scripts for the configuration and installation of a package. This is conveniently done by GNU Autotools, where GNU Autoconf creates the configuration files, GNU Automake creates *Makefile*s. The complete build and installation in the simplest scenario is achieved by a combination of commands:

```
./configure
make
make install
```

BeatBox aims to be self sustained package with a minimal number of dependencies. For that reason it is desirable to include the eigenvalue solver into the BeatBox distribution, rather than relying on the user to install GSL as a separate package. This can be done using GNU Autoconf as a “nested” package. This is done by adding the following line into *configure.ac* in the top directory of the BeatBox repository.

```
AC_CONFIG_SUBDIRS([src/gsl-1.16.extract])
```

The *src/gsl-1.16.extract* is the directory containing the GSL library. During the build, the `make` will descend to the GSL library and perform the build as needed. At the linking stage of the BeatBox the compiler will include the functions from the GSL library into the binary files. For that the Automake file *scr/Makefile.am* needs to refer to the path to the GSL libraries as follows:

```
beatbox_CPPFLAGS += -I$(GSL)
beatbox_SEQ_CPPFLAGS += -I$(GSL)/
beatbox_LDADD += -lgsl -lgslcblas
```

Appendix C

rushlarsen Source Code

C.1 Source Code of *ionic.h*

Listing C.1: *ionic.h*

```
/**
 * Interface with cardiac cell model description,
 * describing HH-type gates separately.
 * Also modified to describe MC models separaterly.
 */

#ifndef _ionic
#define _ionic
typedef struct {                /* description of dependent parameters */
  int n;
  int *src;
  real **dst;
} Var;

#define IONICFTAB(name) int name(real V, real *values, int ntab)
typedef IONICFTAB(IonicFtab);
#define IONIC_FTAB_HEAD(name)   \
IONICFTAB(ftab_##name) {
#define IONIC_FTAB_TAIL(name)   \
  return 1;                     \
}

/* Type of function definining the right-hand side for non-gate variables */
/* u: vector of dynamic variables */
/* nv: number of elements in v */
/* values: table of tabulated functions */
/* ntab: number of rows in the table (number of voltage values) */
/* Par: the array of parameters of this ionic model */
/* Var: the array of descriptors of variable parameters of this ionic model */
/* du: vector for the derivatives of dynamic variables (output) */
/* no: the number of "other" variables */
/* nalp: the array of non-tabulated alpha-rates */
/* nbet: the array of non-tabulated beta-rates */
/* nn: the number of non-tabulated gates */

#include "channel.h"

#define IONICFDDT(name) int name(real *u,int nv,real *values,int ntab,Par par,\
                            Var var,real *du,int no,real *nalp,real *nbet,int nn)
typedef IONICFDDT(IonicFddt);
/* Header of a standard ionic rhs calculator: */
/* - nostrify the parameters list, */
/* - check dimensions of subvectors, */
/* - implement parameter substitution if needed. */
#define IONIC_FDDT_HEAD(name,NV,NTAB,NO,NN)     \
IONICFDDT(fddt_##name) {        \
  STR *S = (STR *)par;  \
  int ivar;             \
  if(nv!=NV) ABORT("nv=%d␣!=␣NV=%d\n",nv,NV);   \
  ASSERT(ntab==NTAB);   \
  ASSERT(no==NO);       \
  ASSERT(nn==NN);       \
  if (var.n) for(ivar=0;ivar<var.n;ivar++) *(var.dst[ivar])=u[var.src[ivar]];

#define IONIC_FDDT_TAIL(name)           \
  return 1;             \
}

/* Solver-independent entities exported by an ionic model description */
typedef struct {
  IonicFtab *ftab;      /* voltage dependent functions that can be tabulated */
  int nmc;              /* number of Markov chain models */
  int nmv;              /* total number of Markov chain variables */
  IonicFddt *fddt;      /* right-hand sides of non-gate equations */
  int no;               /* number of non-gate variables */
  int nn;               /* number of nontab gate variables */
  int nt;               /* number of tab gate variables */
  int ntab;             /* number of tabulated functions  */
  int V_index;          /* index of voltage in the state vector */
  Par p;                /* vector of model parameters */
  Var var;              /* description of dependent parameters */
  channel_str * channel;        /* definitions of ion channel */
} ionic_str;


#define IONICCREATE(name) int name(ionic_str *I,char *w,real **u,int v0)
typedef IONICCREATE(IonicCreate);

#define IONIC_CREATE_HEAD(name)                         \
IONICCREATE(create_##name) {                            \
  STR *S = (STR *)Calloc(1,sizeof(STR));                \
  char *p=w;                                            \
  Var *var=&(I->var);                                   \
  int ivar=0;                                           \
  int ig;       /* gates counter */                     \
  real V0;      /* initial voltage */                   \
  if (!S) ABORT("cannot␣create␣%s",#name);              \
  for(var->n=0;*p;var->n+=(*(p++)==AT));                \
  if(var->n){CALLOC(var->dst,var->n,sizeof(real *));    \
  CALLOC(var->src,var->n,sizeof(int));}                 \
  else {var->src=NULL;(var->dst)=NULL;}                 \
  I->V_index=V_index;                                   \
  /* Accept the non-cell parameter values by the standard macro */      \
  ACCEPTP(IV,0,RNONE,RNONE);            /* By default, no stimulation */ \
  /* Create the vector of initial  values of dynamic (state) variables. */\
  MALLOC(*u,(long int)NV*sizeof(real));\
  int ii;        /* TODO: rewrite */                     \
  /* for (ii=0; ii < NMC; ii++) {nmv+=nm[ii]; nme+=nm[ii]*nm[ii];} */\
  /* allocate the channel structure */ \
  channel_str * ch;                             \
  subchain_str * sbch;                          \
  CALLOC(I->channel,NMC,sizeof(channel_str));\
  ch = &(I->channel[0]);                                \
  for (ii=0; ii <NMC; ii++)                             \
    CALLOC(I->channel[ii].subchain, MAX_SUBCHAINS, sizeof(subchain_str));\

#define IONIC_CREATE_TAIL(name,rc)      \
  var->n=ivar;                          \
  if(ivar){REALLOC(var->dst,1L*ivar*sizeof(real *));\
  REALLOC(var->src,1L*ivar*sizeof(int));}       \
  else{FREE(var->dst);FREE(var->src);}  \
  I->p = S;                             \
  I->no = NO;                           \
  I->nn = NN;                           \
  I->nt = NT;                           \
  I->ntab = NTAB;                       \
  I->nmc = NMC;                                                         \
  int nmv=0;                                     \
  for (ii=0; ii < NMC; ii++){                                            \
    nmv += I->channel[ii].dimension;                                     \
  }                                                                      \
  I->nmv=nmv;                                                            \
  ASSERT(NV==NO+NN+NT+nmv);                                              \
  return rc;                                                             \
}

#define IONIC_CONST(type,name) type name=S->I.name;
#define IONIC_ARRAY(type,name) type *name=&(S->I.name[0]);

#endif
```

C.2 Source Code of *channel.h*

Listing C.2: *channel.h*

```
/* maximal number of allowed subchains in a Markov chain models */
#define MAX_SUBCHAINS 4

#define SUBCHAIN(fun_tr, index, min, max, incr, sc)      \
  sbch->trans_rates_mat = fun_tr;                \
  sbch->i_control = index;                       \
  sbch->tmin = min;                              \
  sbch->tmax = max;                              \
  if (min > max) {                                       \
  URGENT_MESSAGE("min␣>␣max␣for␣subchain␣tabulation␣of␣%s", #fun_tr);   \
  ABORT("");                                                             \
  }                                              \
  sbch->tincr = incr;                            \
  sbch->scale = sc;                              \
  ch->num_sub += 1;                              \
  sbch+=1;

#define TR_MAT(channel,from, to, direct, reverse)                        \
  tr_mat[channel##_##to*NM_##channel+channel##_##from]=direct;           \
  tr_mat[channel##_##from*NM_##channel+channel##_##from]-=direct;        \
  tr_mat[channel##_##from*NM_##channel+channel##_##to]=reverse;          \
  tr_mat[channel##_##to*NM_##channel+channel##_##to]-=reverse;

#define CHANNEL_TR_MATRIX(name) int name(real * u, real *tr_mat)
typedef CHANNEL_TR_MATRIX(TransRatesMat);

/* subchannel of Markov chain model */
typedef struct {
  TransRatesMat *trans_rates_mat; /* function to get transition rates matrix */
  int i_control;          /* index of controling dynamical variable */
  real tmin;              /* minimal value for tabulation */
  real tmax;              /* maximal value for tabulation */
  real tincr;             /* increment in the tabulation */
  int scale;              /* 0 for linar, 1 for logarithmic */
} subchain_str;

/* general structure of ionic channel to embarace Markov chain and gate model */
typedef struct {
  int dimension;                /* dimensionality of the model */
  int num_sub;                  /* number of subchains */
  subchain_str * subchain;      /* subchains of the model */
  /* TODO: add conservation variable */
} channel_str;
```

C.3 Source Code of *rushlarsen.c*

Listing C.3: *rushlarsen.c*

```
/**
 * Rush-Larsen solver for cardiac excitability kinetic models.
 * Utilizes a special kinetics format, describing HH-type gates separately.
 * It also adds Matrix Rush Larsen solver for Markov chain.
 */
```

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>
#include <gsl/gsl_permutation.h>

#include "system.h"
#include "beatbox.h"
#include "device.h"
#include "state.h"
#include "bikt.h"
#include "ionic.h"
#include "qpp.h"

typedef struct {
  real ht;              /* time step */
  Name order;           /* text code of the order of execution */
  int whichorder;       /* numeric code of the order of execution */
  Name ionic;           /* ionic of the ionic cell model */
  ionic_str I;          /* ionic cell description */
  int exp_ngate;        /* if nonzero, non-tabulated gates are stepped by RL,
                           if zero by FE */
  Name exp_mc;          /* text code of the MC method */
  int which_exp_mc;     /* numeric code of the MC method */
  int rest;             /* how many steps to do to find resting values */
  real Vmin, Vmax;      /* limits of voltage in the table (Vmax is approximate) */
  real dV;              /* voltage increment in the table */
  int equilibrate_gates;/* whether to equilibrate gates in the initial state */
  real *u;              /* vector of vars for steady state if any */
  real *du;             /* vector of derivatives */
  real *nalp;           /* vector of nontab alphas */
  real *nbet;           /* vector of nontab betas */
  int nV;               /* number of rows in the table */
  real one_o_dV;        /* inverse of voltage increment in the table */
  real *tab;            /* the table of values of functions */
  real *adhoc;          /* array of ad hoc values of tabulable functions */
  real *chains;         /* matrix of MC transition rates */
} STR;

/************************************************/
/* Structure of the state vector:               */
/* 0 .. no-1: no "other variables"              */
/* no .. no+nn-1: nn non-tab gates              */
/* no+nn .. no+nn+nt-1=nv-1":  nt tab gates     */
/* So nv=no+nn+nt                               */

/****************************************************************/
/* Structure of the vector of tabulated functions:              */
/* 0 .. nt-1            alphas/a-coefficients                   */
/* nt .. 2*nt-1         betas/b-coefficients                    */
/* 2*nt .. ntab-1       everything else                         */
/* NB: the first 2*nt values are dual purpose (alp/bet->a/b)    */

/********************************************************/
/* Conversion of alpha anb beta into the                */
/* Rush-Larsen linear combination coefficients:         */
/* a=alpha/(alpha+beta)*(1-exp(-(alpha+beta)*ht));      */
/* b=exp(-(alpha+beta)*ht).                             */
/* This is a bit tricky; for debugging might be an idea */
/* to do this in a more straightforward way...          */
static inline void calcab (real *a,real *b,real ht) {
  /* first a=alp, b=bet */
  (*b)+=(*a);      /* b=alp+bet         */
  (*a)/=(*b);      /* a=alp/(alp+bet) */
  (*b)*=ht;        /* b=(alp+bet)*ht) */
  (*b)=exp(-(*b)); /* b=exp(-(alp+bet)*ht) */
  (*a)*=(1-(*b));  /* a=alp/(alp+bet)*(1-exp(-(alp+bet)*ht)) */
}

int
memcpy_gsl_matrix_complex_to_real (real *dest, gsl_matrix_complex *src, int n)
```

```
{
  /*
     Convert the gsl format of eigenvector matrices to the format used
     in Beatbox.

     As the eigenvectors come from MC, the complex part should be zero
     (asserted in the code) and can be ignored.

     This function copies the entries of array "src" into array
     "dest", both arrays should be of size n*n as we operate on square
     matrices of dimension n.
  */
  /* assert the gsl_matrix "src" is square of dimension n */
  ASSERT(src->size1 == n);
  ASSERT(src->size2 == n);
  /* loop to copy real entries of "src" into "dest" */
  int unsigned i, j;
  for (i=0; i < n; i++)
    {
      for (j=0; j < n; j++)
        {
          /* assert imaginary part is zero */
          /* the imaginary part has some tolerance */
          if ( fabs(src->data[2 * (src->tda * i + j)+1]) != 0)
            ABORT("Error:␣Imaginary␣part␣of␣eigenvector␣matrix␣is␣not␣zero␣"
                 "(it␣is␣%g).\n", fabs(src->data[2 * (src->tda * i + j)+1]));
          /* copy real part of "src" to "dest"  matrix */
          dest[n * i + j] = (real) src->data[2 * (src->tda * i + j)];
        }
    }
  return 1;
}

int
memcpy_gsl_vector_complex_to_real (real *dest, gsl_vector_complex *src, int n)
{
  /*
     Convert the gsl format of eigenvalue vector to the format used
     in Beatbox.

     As the eigenvalues come from MC, the complex part should be zero
     (asserted in the code) and can be ignored.

     This function copies the entries of array "src" into array
     "dest", both arrays should be of size n (number of eigenvectors).
  */
  /* assert the gsl vector "src" is square of dimension n */
  ASSERT(src->size == n);
  /* loop to copy real entries of "src" into "dest" */
  int unsigned i;
  for (i=0; i < n; i++)
    {
          /* assert imaginary part is zero */
          /* the imaginary part has some tolerance */
      if ( fabs(src->data[2 * src->stride * i + 1]) != 0)
        ABORT("Error:␣Imaginary␣part␣of␣eigenvalue␣vector␣is␣not␣zero␣"
              "(it␣is␣%g).\n", src->data[2 * src->stride * i + 1]);
          /* copy real part of "src" to "dest"  matrix */
          dest[i] = (real) src->data[2 * src->stride * i];
    }
  return 1;
}

/* function to obtain matrix Rush-Larsen by eigenvalue decomposition
   and exponentiation */
/* it returns the mrl in Beatbox format -- real */
int
get_matrix_rush_larsen (real *markov_rates, real ht, real *trans_rates, int n)
{
  unsigned int i, j, k;          /* loop counters */
  double factor;                 /* factor for eigenvector normalization */
  /* ******************************************************* */
  /* create and allocate memory in gsl format structure */
  /* right eigenvalues complex  */
  gsl_vector_complex *eval_right_gsl = gsl_vector_complex_alloc ((size_t) n);
  /* left eigenvalues complex */
```

```
gsl_vector_complex *eval_left_gsl = gsl_vector_complex_alloc ((size_t) n);
/* right eigenvector matrix */
gsl_matrix_complex *evec_right_gsl =
  gsl_matrix_complex_alloc ((size_t) n, (size_t) n);
/* left eigenvector matrix */
gsl_matrix_complex *evec_left_gsl =
  gsl_matrix_complex_alloc ((size_t) n, (size_t) n);
/* workspace for nonsymetric eigenvalue problem */
gsl_eigen_nonsymmv_workspace *workspace =
  gsl_eigen_nonsymmv_alloc ((size_t) n);
/* transposed transition rates matrix of INa */
real *tr, *tr_transp;
/* to eigenvalues and eigenvectors in simple format */
real *evec_right, *evec_left, *evals;
/* auxilary arrays for MRL computations */
real *exp_evals_ht, *evec_right_eval;

/* allocate space to eigenvalues and eigenvectors in Beatbox format */
CALLOC(tr_transp, n*n, sizeof(real));
CALLOC(tr, n*n, sizeof(real));
CALLOC(evec_right, n*n, sizeof(real));
CALLOC(evec_left, n*n, sizeof(real));
CALLOC(evals, n, sizeof(real));
CALLOC(exp_evals_ht, n, sizeof(real));
CALLOC(evec_right_eval, n*n, sizeof(real));


/* view to gsl structures */
gsl_matrix_view rates_matrix =
  gsl_matrix_view_array (tr, (size_t) n, (size_t) n);
gsl_matrix_view rates_matrix_transp =
  gsl_matrix_view_array (tr_transp, (size_t) n, (size_t) n);

/* copy tr and transposed tr matrix (seen by gsl_view rates_matrix_transp )*/
for (i = 0; i < n; i++)
  {
    for (j = 0; j < n; j++)
      {
        tr[n * i + j] = trans_rates[n * i + j];
        tr_transp[n * i + j] = trans_rates[n * j + i];
      }
  }

/* compute eigenvalues and eigenvectors */
/* get the RIGHT evals and evecs */
gsl_eigen_nonsymmv (&rates_matrix.matrix, eval_right_gsl, evec_right_gsl,
                    workspace);
/* get the LEFT evals and evecs */
gsl_eigen_nonsymmv (&rates_matrix_transp.matrix, eval_left_gsl, evec_left_gsl,
                    workspace);
/* ****************************** */
/* process eval and evec */
/* sort the eigenvalues and eigenvectors in descending order */
gsl_eigen_nonsymmv_sort (eval_right_gsl, evec_right_gsl, GSL_EIGEN_SORT_ABS_DESC);
gsl_eigen_nonsymmv_sort (eval_left_gsl, evec_left_gsl, GSL_EIGEN_SORT_ABS_DESC);
/* convert gsl arrays to real */
memcpy_gsl_matrix_complex_to_real (evec_left, evec_left_gsl, n);
memcpy_gsl_matrix_complex_to_real (evec_right, evec_right_gsl, n);
memcpy_gsl_vector_complex_to_real (evals, eval_left_gsl, n);

/* reorder eigenvectors in case of multiple (so far only double) eigenvalues */
#define M 2
real  eta[M*M];
real tmp;
for (i = 0; i < (n - 1); i++)
  {
    /* the "identical" eigenvalues are computed with some tolerance
       if (evals[i] = evals[i + 1]) */
    if (fabs(evals[i] - evals[i + 1]) < 1e-13 )
      {
        /* check for triple eigenvalues -- this has not been resolved */
        if ((i < n - 2) ? (evals[i] == evals[i + 2]) : (0))
          {
            URGENT_MESSAGE ("There␣are␣three␣identical␣eigenvalues.\n");
            ABORT("");
          }
    /* reset the eta's */
    for (j = 0; j < n; j++)
     eta[j] = 0.0;
    for (j = 0; j < n; j++)
     {
      eta[0] += evec_right[n * j + i] * evec_left[n * j + i];
      eta[1] += evec_right[n * j + i + 1] * evec_left[n * j + i];
      eta[2] += evec_right[n * j + i] * evec_left[n * j + i + 1];
      eta[3] += evec_right[n * j + i + 1] * evec_left[n * j + i + 1];
     }
    /* if the eigenvector are in reversed order swap right eigenvectors */
    if (eta[0] == 0 && eta[3] == 0)
     {
      for (j = 0; j < n; j++)
       {
        tmp = evec_right[n * j + i];
        evec_right[n * j + i] = evec_right[n * j + i + 1];
        evec_right[n * j + i + 1] = tmp;
       }
     }
    for (j=0;j<M*M; j++) eta[j] = 0.0;
    for (j = 0; j < n; j++)
     {
      eta[0] += evec_right[n * j + i] * evec_left[n * j + i];
      eta[1] += evec_right[n * j + i + 1] * evec_left[n * j + i];
      eta[2] += evec_right[n * j + i] * evec_left[n * j + i + 1];
      eta[3] += evec_right[n * j + i + 1] * evec_left[n * j + i + 1];
     }
    if (eta[0] == 0 || eta[3] == 0 || eta[1] != 0 || eta[2] != 0 )
     {
      URGENT_MESSAGE ("The␣eigenvalues␣are␣not␣biorthogonal.\n");
      ABORT ("");
     }
   }
 }

/* scale left eigenvectors such that the multiplication
   with right eigenvectors gives identity */
for (i = 0; i < n; i++)
 {
  factor = 0.0;
  for (j = 0; j < n; j++)
   {
    factor += evec_right[n * j + i] * evec_left[n * j + i];
   }
  factor = 1/factor;
  for (j = 0; j < n; j++)
   {
    evec_right[n * j + i] *= factor;
   }
 }
/* exponentiate eigenvalue and time step */
for (i = 0; i < n; i++)
 exp_evals_ht[i] = exp (evals[i] * ht);
/* multiply left eigenvalue and eigenvectors and place to auxilary array */
for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
   {
    evec_right_eval[n * i + j] = evec_right[n * i + j] * exp_evals_ht[j];
   }
 }
/* multiply auxilary array with right eigenvectors so we get the MRL */
for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
   {
    markov_rates[n * i + j] = 0.0;
    for (k = 0; k < n; k++)
     {
      markov_rates[n * i + j] +=
       evec_right_eval[n * i + k] * evec_left[n * j + k];
     }
   }
 }
```

```
  /* free gsl memory */
  FREE(tr_transp);
  FREE(tr);
  FREE(evec_right);
  FREE(evec_left);
  FREE(evals);
  FREE(exp_evals_ht);
  FREE(evec_right_eval);

  gsl_vector_complex_free (eval_right_gsl);
  gsl_vector_complex_free (eval_left_gsl);
  gsl_matrix_complex_free (evec_right_gsl);
  gsl_matrix_complex_free (evec_left_gsl);
  gsl_eigen_nonsymmv_free (workspace);

  return 1;
}

/* print matrix -- for debugging purposes*/
void
print_matrix (real *m, int n)
{
  /* algorithm to print square matrix m of dimension n */
  int i, j;
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
        {
          printf ("%10.2g", m[n * i + j]);
        }
      printf ("\n");
    }
  printf ("\n");
}

/* print matrix -- for debugging purposes*/
void
print_complex_matrix (real *m, int n)
{
  /* algorithm to print square complex matrix m of dimension n */
  int i, j;
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < 2*n; j=j+2)
        {
            printf ("(%0.2g", m[2*n * i + j]);
            printf ("%+0.2gi)\t", m[2*n * i + j + 1 ]);
        }
      printf ("\n");
    }
  printf ("\n");
}

/* Substeps of the algorithm can be done in different order */

/***********************/
/* TABULATED FUNCTIONS */
/* check voltage */
#define DOTABLES \
  V=u[V_index];                                                         \
  iV=floor((V-Vmin)*one_o_dV);                                          \
  /* if outside limits or tabulation step dV is zero calculate */       \
  /* tabulated values by formulas */                                    \
  if (iV<0 || iV>=nV || S->dV == 0.0 ) {                                \
    if (S->dV != 0.0) printf("V=%g␣outside␣[%g,%g]␣at␣t=%ld␣at␣point␣%d,%d,%d\n",\
                             V,Vmin,Vmax,t,x,y,z);                      \
    values=adhoc;                                                       \
    if (!ftab(V,values,ntab))                                   \
      ABORT("error␣calculating␣ftab(%s)␣at␣t=%ld␣point␣%d␣%d␣%d:␣V=%g\n",ionic,\
            t,x,y,z,V);                                                 \
    for (it=0;it<nt;it++)                                               \
      calcab(values+it,values+nt+it,ht);                                \
    /* TODO: report the value of V, to extend the table in the future */ \
  } else {                                                              \
    /* otherwise get them from the table */                             \
    values=tab+iV*ntab;                                         \
  } /* if iV .. else */

/************************/
/* tabulated GATES by Rush-Larsen */
#define DOTGATES \
  a=values;                                                             \
  b=values+nt;                                                          \
  for (it=0;it<nt;it++) {                                               \
    gateold=tgate[it];                                                  \
    tgate[it]=a[it]+b[it]*gateold; /* this is the RL step */            \
    if (!isfinite(tgate[it])) {                                         \
      URGENT_MESSAGE("\nNAN␣(not-a-number)␣at␣t=%ld␣x=%d␣y=%d␣z=%d␣tab␣gate␣%d\n",\
                     t,x,y,z,it);                                       \
      URGENT_MESSAGE("This␣happened␣while␣multiplying␣%g␣by␣%g␣and␣adding␣%g\n",\
                     gateold,b[it],a[it]);                              \
      ABORT("");                                                        \
    } /* if !isfinite */                                                \
  } /* for it */

/************************/
/* equilibrating tabulated GATES */
#define DOTGATESEQ \
  a=values;                                                             \
  b=values+nt;                                                          \
  for (it=0;it<nt;it++) {                                               \
    gateold=tgate[it];                                                  \
    tgate[it]=a[it]/(1-b[it]); /* this is the equilibrium value */      \
    if (!isfinite(tgate[it])) {                                         \
      URGENT_MESSAGE("\nNAN␣(not-a-number)␣equilibrating␣tab␣gate␣%d:␣a=%g,␣b=%g\n"\
                     ,it,a[it],b[it]);                                  \
      ABORT("");                                                        \
    } /* if !isfinite */                                                \
  } /* for it */


/*************************************************************/
/* OTHER VARIABLES by forward Euler */
#define DOOTHER \
  if (!fddt(u,nv,values,ntab,p,var,du,no,nalp,nbet,nn)) {               \
    URGENT_MESSAGE("error␣calculating␣fddt(%s)␣at␣t=%ld␣point␣%d␣%d␣%d:␣u=",\
                   ionic,t,x,y,z);                                      \
    for(iv=0;iv<nv;iv++) URGENT_MESSAGE("␣%lg",u[iv]);                  \
    ABORT("\n");                                                        \
  } /*  if !fddt... */                                                  \
  for (io=0;io<no;io++) {                                               \
    u[io] += ht*du[io]; /* this is the FE step */                       \
    if (u[io]!=u[io]) {                                                 \
      URGENT_MESSAGE("\nNAN␣(not-a-number)␣detected␣at␣t=%ld␣x=%d␣y=%d␣z=%d␣v=%d\n"\
                     ,t,x,y,z,io);                                      \
      URGENT_MESSAGE("This␣happened␣while␣incrementing␣");              \
      for(iv=0;iv<nv;iv++) URGENT_MESSAGE("%c%lg",iv?',':'(',u[iv]);    \
      URGENT_MESSAGE(")␣by␣%lg*",ht);                                   \
      for(jo=0;jo<no;jo++) URGENT_MESSAGE("%c%lg",jo?',':'(',du[jo]);   \
      URGENT_MESSAGE(")\n");                                            \
      ABORT("");                                                        \
    } /* if NaN */                                                      \
  } /* for io */

/*************************************************************/
/* NONTAB GATES by Rush-Larsen                               */
#define DONGATESRL \
  for (in=0;in<nn;in++) {                                               \
    a=nalp+in;                                                          \
    b=nbet+in;                                                          \
    calcab(a,b,ht);                                                     \
    gateold=ngate[in];                                                  \
    ngate[in]=(*a)+(*b)*gateold; /* this is the RL step */              \
    if (!isfinite(ngate[in])) {                                         \
      URGENT_MESSAGE(\
                "\nNAN␣(not-a-number)␣at␣t=%ld␣x=%d␣y=%d␣z=%d␣nontab␣gate␣%d\n" \
                ,t,x,y,z,in);                                           \
      URGENT_MESSAGE("This␣happened␣while␣multiplying␣%g␣by␣%g␣and␣adding␣%g\n",\
                     gateold,*b,*a);                                    \
      ABORT("");                                                        \
    } /* if !isfinite */                                                \
  } /* for in */
```

```
/*************************************************************/
/* equilibrating NONTAB GATES                                */
#define DONGATESEQ \
  for (in=0;in<nn;in++) {                                               \
    a=nalp+in;                                                          \
    b=nbet+in;                                                          \
    ngate[in]=(*a)/((*a)+(*b)); /* this is the equilibrium value */     \
    if (!isfinite(ngate[in])) {                                         \
      URGENT_MESSAGE(\
        "\nNAN␣(not-a-number)␣equilibrating␣nontab␣gate␣%d:␣alp=%g,␣bet=%g\n",\
        in,*a,*b);                                                      \
      ABORT("");                                                        \
    } /* if !isfinite */                                                \
  } /* for in */

/*************************************************************/
/* NONTAB GATES by forward Euler                             */
#define DONGATESFE \
  for (in=0;in<nn;in++) {                                               \
    a=nalp+in;                                                          \
    b=nbet+in;                                                          \
    gateold=ngate[in];                                                  \
    ngate[in] = gateold+ht*((*a)-((*a)+(*b))*gateold); /* this is the FE step */ \
    if (!isfinite(ngate[in])) {                                         \
      URGENT_MESSAGE(\
            "\nNAN␣(not-a-number)␣at␣t=%ld␣x=%d␣y=%d␣z=%d␣nontab␣gate␣%d\n",\
            t,x,y,z,in);                                                \
      URGENT_MESSAGE("This␣happened␣at␣gateold=%g␣ht=%g␣alpha=%g␣beta=%g\n",\
                     gateold,ht,*b,*a);                                 \
      ABORT("");                                                        \
    } /* if !isfinite */                                                \
  } /* for in */


#define DOMARKOV        \
  for (im =0; im < nmc; im++) {
\
    subchain = &(channel[im].subchain[0]);\
    dimension = channel[im].dimension;                   \
                                                         \
    CALLOC(trm, dimension*dimension, sizeof(real));      \
    CALLOC(mrl, dimension*dimension, sizeof(real));      \
    CALLOC(hchain, dimension, sizeof(real));             \
                                                         \
    for (jm = 0; jm < channel[im].num_sub; jm++)                        \
      {                                                                 \
        /* limits of the tabulation */                                  \
        dvar = subchain[jm].tincr;                                      \
        tmax = subchain[jm].tmax;                                       \
        tmin = subchain[jm].tmin;                                       \
                                                                        \
        if ( subchain[jm].i_control >= 0 ) {                            \
          /* find out the scale of the trans_rates_mat function */      \
          if (subchain[jm].scale == 0){                                 \
            /* linear scale */                                          \
            valscale[0] = u[subchain[jm].i_control];                    \
          }                                                             \
          else if (subchain[jm].scale == 1){                            \
            /* logarithmic scale */                                     \
            valscale[0] = log(u[subchain[jm].i_control]);               \
          }                                                             \
          else{                                                         \
            EXPECTED_ERROR("unknown␣tabulation␣scale␣%d", subchain[jm].scale); \
          }                                                             \
          ch_exp_mc = which_exp_mc;                             \
                                                                        \
          val=&(valscale[0]);                                           \
          it=floor((val[0]-tmin)/dvar);                                 \
          if ( dvar > 0 )                                               \
            {                                                           \
              nT=ceil( (subchain[jm].tmax - subchain[jm].tmin) / dvar ); \
            }                                                           \
          else                                                          \
            {                                                           \
              nT = 0;                                                    \
            }                                                            \
        }                                                                \
        else {                                                           \
          /* for channels without unique controling variable */          \
          /* the values will be computed on fly */                       \
          val = u;                                                       \
          nT=0;                                                          \
          it = -1;      /* places the table index out of range */        \
          ch_exp_mc = 0;        /* computes using forward Euler */       \
        }                                                                \
                                                                         \
        /* if the index is outside of precomputed range  */              \
        /* find the solution on fly */                                   \
        if ( ( it < 0 || it >= nT ) || ch_exp_mc == ntabmrl) {           \
          on_fly = 1;                                                    \
          /* compute transition rates matrix */                          \
          if (!(channel[im].subchain[jm].trans_rates_mat)( val, trm))    \
            ABORT("error␣calculating␣mtab(%s)␣table␣for␣V=%g\n",S->ionic,V); \
          /* asign matrix for forward Euler calculations */              \
          markov_adhoc = trm;                                            \
          /* exponential integrator -- matrix_rush_larsen */             \
          if (ch_exp_mc){                                        \
            if ( !get_matrix_rush_larsen (mrl, ht, trm, dimension) )     \
              URGENT_MESSAGE("error␣calculating␣mrl(%s)␣table␣for␣V=%g\n",\
                             S->ionic,V); /* mention the index of mc */ \
            markov_adhoc = mrl;                                          \
          }                                                              \
        }                                                                \
        else {                                                           \
          on_fly = 0;                                                    \
          /* pointer to precomputed and tabulated matrix */              \
          markov_adhoc = &(curr_chain[it*dimension*dimension]);          \
        }                                                                \
                                                                         \
        /* integration method */                                         \
        mat_vec_mult(hchain, markov_adhoc, markov, dimension);           \
        if (ch_exp_mc) {                                         \
          /* MRL step */                                                 \
          for (ii = 0; ii < dimension; ii++) {                           \
            markov[ii] = hchain[ii];                                     \
          }                                                              \
        }                                                                \
        else if (ch_exp_mc == mcfe) {                                    \
          /* FE step */                                                  \
          for (ii = 0; ii < dimension; ii++) {                           \
            markov[ii] += ht * hchain[ii];                               \
          }                                                              \
        }                                                                \
        else {                                                           \
          EXPECTED_ERROR("The␣which_exp_mc␣==␣%d␣is␣not␣supported",which_exp_mc); \
        }                                                                \
                                                                         \
        /* check if the results are finite and within range */           \
        sum = 0.0;                                                       \
        for (ii = 0; ii < dimension; ii++) {                             \
          sum += markov[ii];                                             \
          if (markov[ii]!=markov[ii]) {                                  \
            URGENT_MESSAGE(\
              "\nNAN␣(not-a-number)␣detected␣at␣t=%ld␣x=%d␣y=%d␣z=%d␣v=%d\n",\
              t,x,y,z,io);                                               \
            URGENT_MESSAGE("This␣happened␣while␣incrementing␣");         \
            for(iv=0;iv<nv;iv++) URGENT_MESSAGE("%c%lg",iv?',':'(',u[iv]); \
            URGENT_MESSAGE(")␣by␣%lg*",ht);                              \
            for(jo=0;jo<no;jo++) URGENT_MESSAGE("%c%lg",jo?',':'(',du[jo]); \
            URGENT_MESSAGE(")\n");                                       \
            ABORT("");                                                   \
          } /* if NaN */                                                 \
        }                                                                \
                                                                         \
        /* increment channel pointers */                                 \
        curr_chain += dimension*dimension*nT;                            \
        /* reset auxilary arrays */                                      \
        if (on_fly)                                                      \
          {                                                              \
            for (ii=0;ii< dimension*dimension;ii++){                     \
              trm[ii]=0.0;                                              \
              mrl[ii]=0.0;                                              \
            }                                                           \
        }                                                               \
        /* reset vector with solution increment */                      \
        for (ii=0;ii< dimension;ii++) hchain[ii] = 0.0;                 \
      }                                                                 \
    /* increment markov variable pointer */                             \
    markov += dimension;                                                \
    /* free auxilary arrays */                                          \
    FREE(trm);                                                          \
    FREE(mrl);                                                          \
    FREE(hchain);                                                       \
  }


/* TODO: check if the new u is finite at the end of the time step,
   rather than in each subunit calculation */

enum {
  tgo,
  tog,
  totg,
  numorders
} ordertype;

enum {
  mcfe,
  tabmrl,
  ntabmrl
} mrltype;

/* do matrix vector multiplication of dimension N as dest = mat * vect  */
static inline void
mat_vec_mult(real *dest, real * mat, real * vec, int N)
{
  int im, jm;
  for (im = 0; im < N; im++)
    {
      dest[im] = 0.0;           /* reset destination vector */
      for (jm = 0; jm < N; jm++)
        {
          dest[im] += mat[im * N + jm] * vec[jm];
        }
    }
}

/* compute the rushlarsen step for one cell of the mesh */
/* nv is not used */
static inline int rushlarsen_step(real *u,int nv,STR *S,int x,int y,int z)
{
  IONIC_CONST(Par,p);                   /* set of ionic cell parameters */
  IONIC_CONST(Var,var);                 /* description of variable parameters */
  IONIC_CONST(IonicFddt *,fddt);        /* the rhs functions except gates */
  IONIC_CONST(IonicFtab *,ftab);        /* the tabulated functions calculator */
  IONIC_CONST(int,no);  /* number of non-gate variables in the state vector */
  IONIC_CONST(int,nn);  /* num of nontabulated gate variables in the state vector */
  IONIC_CONST(int,nt);  /* num of tabulated gate variables in the state vector */
  IONIC_CONST(int,ntab);                /* number of tabulated functions */
  IONIC_CONST(int,nmc);                 /* number of Markov chain models */
  IONIC_CONST(int,nmv);                 /* number of Markov chain model
                                           variables */
  IONIC_ARRAY(channel_str,channel);     /* markov chain structures */
  IONIC_CONST(int,V_index);             /* index of V in state vector */
  DEVICE_ARRAY(char,ionic);             /* name of the ionic cell model */
  DEVICE_CONST(real,ht);                /* the time step */
  DEVICE_CONST(int,whichorder);         /* numeric code of the order of execution */
  DEVICE_CONST(int,which_exp_mc);       /* if nonzero, markov chains
                                           are stepped by MRL,
                                           otherwise FE */
  DEVICE_CONST(int,nV);                 /* number of 'rows' in the table */
  DEVICE_CONST(real,Vmin);              /* minimal value of V in the table */
  DEVICE_CONST(real,Vmax);              /* maximal value of V in the table */
  DEVICE_CONST(real,one_o_dV);          /* inverse of increment of V in the table */
  DEVICE_ARRAY(real,du/*[no]*/);        /* array of right-hand sides for FE part */
  DEVICE_ARRAY(real,tab/*[ntab*nV]*/);  /* the table of function values */
  DEVICE_ARRAY(real,nalp/*[nn]*/);      /* array of alphas for the nontab RL part */
  DEVICE_ARRAY(real,nbet/*[nn]*/);      /* array of betas for the nontab RL part */
  DEVICE_ARRAY(real,adhoc/*[ntab]*/);   /* freshly calculated values
                                           of functions and matrix rush larsen */
  int io, jo, in, it, it1, iv;          /* vector components counters */
  int iV;                               /* table row counter */
  real V;                               /* transmembrane voltage value */
  real *ngate;                          /* subvector of nontab gate values */
  real *tgate;                          /* subvector of tab gate values */
  real *values;                         /* vector of values of tabulated
                                           functions */
  real *a, *b;                          /* pointers to subvectors of
                                           Rush-Larsen step coefficients */
  real gateold;                         /* aux variable */
  real *markov,*markov_cur;             /* subvector of MC values */
  real *dmarkov,*dmarkov_cur;           /* subvector of MC values increments */
  real markov_entry;                    /* auxilary markov state */
  real *mrl;                            /* matrix for rush-larsen computations */
  int im,jm,km;                         /* vector components counters */
  int ii;                               /* MC counter */
  real *curr_chain;                     /* pointer to current chain matrix */
  subchain_str * subchain;
  int dimension, nT;
  int on_fly;                   /* flag to specify if the markov_adhoc was done
                                   on the fly or obtained from tabulated data */
  real * dchain, *hchain, *trm, * markov_adhoc, * val ;
  real valscale[1];
  real dvar, tmax, tmin;
  real sum;                     /* variable to check the sum of states */
  int ch_exp_mc;


  CALLOC(dmarkov,nmv,sizeof(real));

  ngate=u+no;
  tgate=u+no+nn;
  markov=u+no+nn+nt;
  curr_chain=&(S->chains[0]);
  switch (whichorder) {
  case tgo:
    DOTABLES;
    DOTGATES;
    DOOTHER;
    DOMARKOV;
    DONGATESFE;
    break;
  case tog:
    DOTABLES;
    DOOTHER;
    DOMARKOV;
    DONGATESFE;
    DOTGATES;
    break;
  case totg:
    DOTABLES;
    DOOTHER;
    DOMARKOV;
    DONGATESFE;
    DOTABLES;
    DOTGATES;
    break;
  case tgo+numorders:
    DOTABLES;
    DOTGATES;
    DOOTHER;
    DOMARKOV;
    DONGATESRL;
    break;
  case tog+numorders:
    DOTABLES;
    DOOTHER;
    DOMARKOV;
    DONGATESRL;
    DOTGATES;
    break;
  case totg+numorders:
    DOTABLES;
    DOOTHER;
    DOTABLES;
    DOMARKOV;
    DONGATESRL;
    DOTABLES;
    DOTGATES;
    break;
  default:
    EXPECTED_ERROR("unknown␣order␣of␣execution␣code␣%d\n",whichorder);
  } /* swicth whichorder */
  FREE(dmarkov);
  return 1;
}

static inline int equilibration_step(real *u,int nv,STR *S)
{
  IONIC_CONST(Par,p);                   /* set of ionic cell parameters */
  IONIC_CONST(Var,var);                 /* description of variable parameters */
  IONIC_CONST(IonicFddt *,fddt);        /* the rhs functions except gates */
  IONIC_CONST(IonicFtab *,ftab);        /* the tabulated functions calculator */
  IONIC_CONST(int,no);  /* number of non-gate variables in the state vector */
  IONIC_CONST(int,nn);  /* num of nontabulated gate variables in the state vector */
  IONIC_CONST(int,nt);  /* num of tabulated gate variables in the state vector */
  IONIC_CONST(int,ntab);                /* number of tabulated functions */
  IONIC_CONST(int,V_index);             /* index of V in state vector */
  DEVICE_ARRAY(char,ionic);             /* name of the ionic cell model */
  DEVICE_CONST(real,ht);                /* the time step */
  DEVICE_CONST(int,whichorder);         /* numeric code of the order of execution */
  DEVICE_CONST(int,nV);                 /* number of 'rows' in the table */
  DEVICE_CONST(real,Vmin);              /* minimal value of V in the table */
  DEVICE_CONST(real,Vmax);              /* maximal value of V in the table */
  DEVICE_CONST(real,one_o_dV);          /* inverse of increment of V in the table */
  DEVICE_ARRAY(real,du/*[no]*/);        /* array of right-hand sides for FE part */
  DEVICE_ARRAY(real,tab/*[ntab*nV]*/);  /* the table of function values */
  DEVICE_ARRAY(real,nalp/*[nn]*/);      /* array of alphas for the nontab RL part */
  DEVICE_ARRAY(real,nbet/*[nn]*/);      /* array of betas for the nontab RL part */
  DEVICE_ARRAY(real,adhoc/*[ntab]*/);   /* freshly calculated values of functions */
  int io, jo, in, it, it1, iv;          /* vector components counters */
  int iV;                               /* table row counter */
  real V;                               /* transmembrane voltage value */
  real *ngate;                          /* subvector of nontab gate values */
  real *tgate;                          /* subvector of tab gate values */
  real *values;                         /* vector of values of tab. functions */
  real *a, *b;                          /* pointers to subvectors of Rush-Larsen
                                           step coefficients */
  real gateold;                         /* aux variable */
  int x=-1;                             /* these are */
  int y=-1;                             /*   required */
  int z=-1;                             /*   by DOTABLES */

  ngate=u+no;
  tgate=u+no+nn;

  DOTABLES;
  DOOTHER;
  DONGATESEQ;
  DOTGATESEQ;

  return 1;
}

/****************/
RUN_HEAD(rushlarsen) {
  int nv;                               /* size of the state vector */
  int x, y, z;                          /* space grid counters */
  real V;                               /* transmembrane voltage value */
  real *u;                              /* state vector at this point */

  /* The number of layers given to this device */
  nv=s.v1-s.v0+1;

  for(x=s.x0;x<=s.x1;x++) {
    for(y=s.y0;y<=s.y1;y++) {
      for(z=s.z0;z<=s.z1;z++) {
        if(isTissue(x,y,z)){
          u = (real *)(New + ind(x,y,z,s.v0));
          if NOT(rushlarsen_step(u,nv,S,x,y,z)) return 0;
        } /*  if isTissue */
      } /*  for z */
    } /*  for y */
  } /*  for x */
}
RUN_TAIL(rushlarsen)

/**********************/
DESTROY_HEAD(rushlarsen) {
  FREE(S->du);
  FREE(S->nalp);
  FREE(S->nbet);
  FREE(S->adhoc);
  FREE(S->tab);
  FREE(S->u);
  if (S->I.var.n) {
    FREE(S->I.var.src);
    FREE(S->I.var.dst);
  }
  FREE(S->I.p);
} DESTROY_TAIL(rushlarsen)

/* Declare all available ionic models */
#define D(a) IonicFtab ftab_##a;
#include "ioniclist.h"
#undef D
#define D(a) IonicFddt fddt_##a;
#include "ioniclist.h"
#undef D
#define D(a) IonicCreate create_##a;
#include "ioniclist.h"
#undef D

/****************************************/
CREATE_HEAD(rushlarsen)
{
  int nv=dev->s.v1-dev->s.v0+1;
  int no, nn, nt, ntab, nV, nmc, nmv;
  channel_str * channel;
  int size_tr;
  /* int *nm; */
  int step, iv, ix, iy, iz;
  int in, it, io, jo, iV;
  real *ufull, *u, *values;
  real *tr, *mrl, *adhoc;
  real V, alp, bet;
  int im,jm,ii;

  /* Create tables for Markov chain models */
  /* goes to the top */
  real dvar, tmax, tmin, one_o_dvar;
  real * trm, * markov_adhoc;                   /* pointer to the tr_tab */
  subchain_str  * subchain;
  int dimension, nT;
  real var[1];

  /* Accept the time step */
  ACCEPTR(ht,RNONE,0.,RNONE);
  if (ht==0) MESSAGE(
                "/*␣WARNING:␣ht=0␣is␣formally␣allowed␣but␣hardly␣makes␣sense␣*/");

  /* Accept the execution order */
  ACCEPTS(order,"totg");
  STRSWITCH(order);
  STRCASE("tgo")  S->whichorder=tgo;
  STRCASE("tog")  S->whichorder=tog;
  STRCASE("totg") S->whichorder=totg;
  STRDEFAULT EXPECTED_ERROR("\nrushlarsen:␣unknown␣execution␣order␣'%s'\n",order);
  STRENDSW

  ACCEPTI(exp_ngate,0,0,1);
  if (exp_ngate) S->whichorder+=numorders;

  /* Accept the MC integration method */
```

```
ACCEPTS(exp_mc,"tabmrl");
STRSWITCH(exp_mc);
STRCASE("mcfe")  S->which_exp_mc=mcfe;
STRCASE("tabmrl")  S->which_exp_mc=tabmrl;
STRCASE("ntabmrl") S->which_exp_mc=ntabmrl;
STRDEFAULT EXPECTED_ERROR(
            "\nrushlarsen:␣unknown␣MC␣integration␣method␣exp_mc␣'%s'\n",exp_mc);
STRENDSW

/* Accept the ionic cell model */
ACCEPTS(ionic,NULL);
{
  char *pars;
  MALLOC(pars,(long)MAXSTRLEN);
  BEGINBLOCK("par=",pars);
  S->u=NULL;
  #define D(a)                                                         \
  if (0==stricmp(S->ionic,#a)) {                                       \
    if NOT(create_##a(&(S->I),pars,&(S->u),dev->s.v0))                 \
      EXPECTED_ERROR("reading␣parameters␣for␣%s␣in␣\"%s\"",S->ionic,pars); \
    no=S->I.no; nn=S->I.nn; nt=S->I.nt;                                \
    nmc=S->I.nmc;                                                      \
    channel=&(S->I.channel[0]);                                        \
    nmv=S->I.nmv;                                                      \
    ntab=S->I.ntab;                                                    \
    if (no+nn+nt+nmv!=nv)                                              \
      EXPECTED_ERROR("no+nn+nt+nmv=%d+%d+%d+%d␣!=␣nv=%d␣for␣%s\n"      \
                     "Please␣correct␣the␣number␣of␣layers␣in␣the"      \
                     "input␣file␣(BBScript).\n",no,nn,nt,nmv,nv,S->ionic); \
    S->I.ftab=ftab_##a;                                                \
    S->I.fddt=fddt_##a;                                                \
  } else
  #include "ioniclist.h"
  #undef D
  EXPECTED_ERROR("unknown␣ionic␣model␣%s",S->ionic);
  ENDBLOCK;
  FREE(pars);
}

/* compute required sizes for MC */
for (im = 0; im < nmc; im++)
  {
    /* the table is only created for subchains which are tabulated
       i.e. subchains with specified i_control variable (only single
       variable dependent markov chains which are suitable for
       tabulation). */
    /* the corresponding size for each subchain is the
       ntab_entries*dimension^2 */

    subchain = &(S->I.channel[im].subchain[0]);

    for (jm = 0; jm < S->I.channel[im].num_sub; jm++)
      {
        dvar = subchain[jm].tincr;
        dimension = channel[im].dimension;
        tmax = subchain[jm].tmax;
        tmin = subchain[jm].tmin;
        if (subchain[jm].i_control >= 0 && dvar > 0)
          {
            nT = ceil ((tmax - tmin) / dvar);
            size_tr += dimension * dimension * nT;
          }
      }
  }

/* Allocated working arrays */
CALLOC(S->du, no, sizeof (real));
CALLOC(S->nalp, nn, sizeof (real));
CALLOC(S->nbet, nn, sizeof (real));
CALLOC(S->adhoc, ntab, sizeof (real));
CALLOC(S->chains, size_tr, sizeof (real));

/* tabulate Markov chain transition rates matrices */
markov_adhoc = &(S->chains[0]);
for (im = 0; im < nmc; im++)
  {
    subchain = &(S->I.channel[im].subchain[0]);
    dimension = S->I.channel[im].dimension;

    CALLOC (trm, dimension * dimension, sizeof (real));

    for (jm = 0; jm < channel[im].num_sub; jm++)
      {
        dvar = subchain[jm].tincr;
        tmax = subchain[jm].tmax;
        tmin = subchain[jm].tmin;
        if (subchain[jm].i_control >= 0 && dvar > 0)
          {
            nT = ceil ((subchain[jm].tmax - subchain[jm].tmin) / dvar);
            one_o_dvar = 1.0 / dvar;

            for (it = 0; it < nT; it++)
              {
                /* tabulate transition rates of gate and Markov chain and other
                   variables */
                var[0] = tmin + (it + 0.5) * dvar;
                if (!(S->I.channel[im].subchain[jm].trans_rates_mat) (var, trm))
                  ABORT ("error␣calculating␣mtab(%s)␣table␣for␣V=%g\n",\
                         S->ionic, V);
                /* get exponential integrator -- matrix_rush_larsen */
                if (S->which_exp_mc == tabmrl)
                  {
                    if (!get_matrix_rush_larsen (
                            &(markov_adhoc[it * dimension * dimension]),
                            ht, trm, dimension))
                      URGENT_MESSAGE ("error␣calculating␣mrl(%s)␣table␣for␣V=%g\n"
                                      ,S->ionic, V);
                  }
                else if (S->which_exp_mc == mcfe)
                  {
                    memcpy (&(markov_adhoc[it * dimension * dimension]), trm,
                            dimension * dimension * sizeof (real));
                  }

                for (ii = 0; ii < dimension * dimension; ii++)
                  {
                    trm[ii] = 0.0;
                    if (markov_adhoc[it * dimension * dimension + ii] !=
                        markov_adhoc[it * dimension * dimension + ii])
                      {
                        URGENT_MESSAGE ("\nNaN␣in␣calculation␣of␣"
                                        "markov_adhoc(%s)␣table␣for␣var=%.15g\n",
                                        S->ionic, var[0]);
                        ABORT ("");
                      }
                  }
              }
            markov_adhoc += dimension * dimension * nT;
          }
      }
    FREE (trm);
  }


/* In any case, do the tabulation */
ACCEPTR(Vmin,-200,RNONE,RNONE);
ACCEPTR(Vmax,+200,RNONE,RNONE);
ACCEPTR(dV,0.01,0.,RNONE);
if ( dV == 0.0)
  {
    MESSAGE("/*␣NOTE:␣dV=0␣means␣the␣transition␣rates␣are␣calculated␣on␣the␣fly"
            "␣and␣not␣tabulated.␣*/");
    CALLOC(S->tab,1,sizeof(real)); /* for compatibility reasons,
                                      otherwise there are problems
                                      with destruction of the
                                      device */
  }
else
  {

    S->nV=nV=ceil((S->Vmax-S->Vmin)/S->dV);
    S->one_o_dV=1.0/S->dV;
```

```
      CALLOC(S->tab,ntab*nV,sizeof(real));

      for (iV=0;iV<nV;iV++) {
        /* tabulate transition rates of gate and Markov chain and other variables */
        V=Vmin+(iV+0.5)*dV;
        var[0]=V;
        values=&(S->tab[ntab * iV]);
        if (!(S->I.ftab)(V,values,ntab))
          ABORT("error␣calculating␣ftab(%s)␣table␣for␣V=%g\n",S->ionic,V);
        for (it=0;it<nt;it++)
          calcab(values+it,values+nt+it,ht);
      }
    }

  ACCEPTI(rest,0,0,INONE);
  if (S->rest) {
    /* Need full vector, in case variable parameters use extra layers.   */
    /* NB extra layers may be above as well as below this device's space */
    CALLOC(ufull,vmax,sizeof(real));
    u=&(ufull[dev->s.v0]);

    if (S->u) {
      MESSAGE("\n/*␣NOTICE:␣finding␣resting␣state␣while␣already␣"
              "defined␣by␣the␣model␣*/");
      /* use that as the initial condition */
      for (iv=0;iv<nv;iv++) u[iv]=S->u[iv];
    } else {
      CALLOC(S->u,nv,sizeof(real));
    }

    if (S->I.var.n) MESSAGE("\n/*␣NOTICE:␣finding␣resting␣state␣"
                            "while␣parameters␣are␣variable␣*/");

    for (step=0;step<S->rest;step++)
      /* if NOT(rushlarsen_step(u,nv,S,-1,-1,-1)) return 0; */
      if NOT(equilibration_step(u,nv,S)) return 0;

    /* copy what is needed */
    for (iv=0;iv<nv;iv++) S->u[iv]=u[iv];

    /* don't need this one any more */
    FREE(ufull);
  } /* if S->rest */

  if (S->u) {
    MESSAGE0("\n/*␣Resting␣state:␣");
    for(iv=0;iv<nv;iv++) MESSAGE1("%lg␣",(S->u)[iv]); MESSAGE0("*/");
    /* Fill up the whole of the grid with the resting state */
    #if MPI
    if (dev->s.runHere) {
    #endif
      for (ix=dev->s.x0;ix<=dev->s.x1;ix++) {
        for (iy=dev->s.y0;iy<=dev->s.y1;iy++) {
          for (iz=dev->s.z0;iz<=dev->s.z1;iz++) {
            for (iv=dev->s.v0;iv<=dev->s.v1;iv++) {
              New[ind(ix,iy,iz,iv)]=(S->u)[iv-dev->s.v0];
            } /* for iv */
          } /* for iz */
        } /* for iy */
      } /* for ix */
    #if MPI
    } /* if runHere */
    #endif
  } else {
    /* No resting state was defined */
    MESSAGE("\n/*␣Notice:␣no␣standard␣state␣defined␣for␣ionic␣model␣'%s'␣*/\n",\
            ionic);
    CALLOC(S->u,nv,sizeof(real));
  }
}
CREATE_TAIL(rushlarsen,1)
```

Appendix D

Implementation of Cellular Models

D.1 Standalone Code

D.1.1 Hodgkin-Huxley Squid Model

The aim of this appendix is to demonstrate the implementation of minimalist cellular models. The implementation is inspired by the Hodgkin-Huxley squid giant axon description [3], which is the first electrophysiological model. The original description contains only four dynamical variables, one describing the membrane voltage, two variables describing the gating variables of I_{Na} and one variable corresponding to the gating variable of I_{K}.

The model is implemented as standalone code and as BeatBox modules. The BeatBox modules include the implementation as a `rhs` module and two `ionic` modules. The first `ionic` module is identical to the original model description, the second contains ion channels defined as Markov chains. These Markov chains are equivalent to the corresponding gating variables and were developed for the sake of demonstrating the time integration on Markov chains in BeatBox. The description of the conversion of both I_{Na} and I_{K} channels into equivalent Markov chain models can be found in Section 2.3.4.

Listings of Hodgkin-Huxley Standalone Code

The implementation of the giant squid axon model in standalone C is listed in the source code below:

Listing D.1: Code of standalone Hodgkin-Huxley model in file *squid_driver.c*.

```
#include <stdio.h>
#include <math.h>               /* to include exp */
#include <stdlib.h>             /* calloc, exit, free */
```

```
#define T_END 10.0
#define DT 0.01

enum
{
#define VAR(name, init) var_##name,
#include "squid_var.h"
#undef VAR
  NEQ
};

typedef struct currents
{
  double I_Na;
  double I_K;
  double I_l;
} s;


/* function called by the solver */
int HH_1952 (double t, double *y, double *ydot, s * user_data);

int
main ()
{
  int i, j;                     /* loop counters */
  int N = T_END / DT;           /* number of time steps */

  double t;                     /* time */
  double y[NEQ], ydot[NEQ];     /* states and states increment */
  s user_data;                  /* currents */

#define VAR(name, init) y[var_##name] = init;
#include "squid_var.h"
#undef VAR

  for (i = 0; i < N; i++)
    {
      t = i * DT;               /* current time */
      /* compute states increment and currents */
      HH_1952 (t, y, ydot, &user_data);

      /* fe step */
      for (j = 0; j < NEQ; j++)
        {
          y[j] += DT * ydot[j];
          if (y[j] != y[j])
            {
              fprintf (stderr, "NaN␣detected␣for␣y[%d].\n", j);
              exit (1);
            }
        }

      /* saving data */
      fprintf (stdout, "%.3f", t);      /* time */
      for (j = 0; j < NEQ; j++)
        {
          /* print dynamical states */
          fprintf (stdout, "\t%.6g", y[j]);
        }
      /* currents */
      fprintf (stdout, "\t%.6g", (&user_data)->I_Na);
      fprintf (stdout, "\t%.6g", (&user_data)->I_K);
      fprintf (stdout, "\n");
    }

  return (0);
}

int
HH_1952 (double t, double *y, double *ydot, s * user_data)
{
  double C_m = 1.0;

#define VAR(name, init) double name = y[var_##name];
#include "squid_var.h"
#undef VAR

#define PAR(p, c)            const double       p = c;
  /* maximum conductance of current I_Na (mS/cm^2) */
  PAR (G_Na, 120);
  /* maximum conductance of current I_K (mS/cm^2)*  */
  PAR (G_K, 36);
  /* maximum conductance of leakage current: I_l (mS/cm^2) */
  PAR (G_l, 0.3);
  /* the sign from HH1952 is according new convention */
  /* reversal potential of current I_Na (mV)  */
  PAR (E_Na, 115);
  /* reversal potential of current I_K (mV)  */
  PAR (E_K, -12);
  /* reversal potential of current I_l (mV)  */
  PAR (E_l, 10.613);
#undef PAR
  /* currents */
  double I_K = user_data->I_K = G_K * n * n * n * n * (V_m - E_K);
  double I_Na = user_data->I_Na = G_Na * m * m * m * h * (V_m - E_Na);
  double I_l = user_data->I_l = G_l * (V_m - E_l);

/* gate transition rates */
  double alpha_n = 0.01 * (-V_m + 10.0) / (exp ((-V_m + 10.0) / 10.0) - 1.0);
  double beta_n = 0.125 * exp (-V_m / 80.0);

  double alpha_m = 0.1 * (-V_m + 25.0) / (exp ((-V_m + 25.0) / 10.0) - 1.0);
  double beta_m = 4.0 * exp (-V_m / 18.0);

  double alpha_h = 0.07 * exp (-V_m / 20.0);
  double beta_h = 1.0 / (exp ((-V_m + 30.0) / 10.0) + 1.0);

/* model equations */
  double dot_V_m = -1.0 / C_m * (I_Na + I_K + I_l);
  double dot_n = alpha_n * (1 - n) - beta_n * n;
  double dot_m = alpha_m * (1 - m) - beta_m * m;
  double dot_h = alpha_h * (1 - h) - beta_h * h;

#define VAR(name, init) ydot[var_##name] = dot_##name;
#include "squid_var.h"
#undef VAR

  return (0);
}
```

The initial conditions are listed in another file that is included from *squid_var.h*, which contains function-like macros for each of the dynamical variables. A detailed explanation of function-like macros is given in Subsection D.2.1.

Listing D.2: Variables in Hodgkin-Huxley model in file *squid_var.h*.

```
/* format: VAR(name, initial) */
/* V_m  - membrane potential (mV) */
VAR(V_m, 7.0)
/* n - activation gate of potassium current I_K  */
VAR(n, 0.3177)
/* m - activation gate of sodium current I_Na  */
VAR(m, 0.0530)
/* h - inactivation gate of sodium current I_Na  */
VAR(h, 0.5960)
```

Building of Hodgkin-Huxley Standalone Code

The compilation and linking of the program can be done by gcc using the following command.

Listing D.3: Compilation of standalone code.

```
gcc -c -o squid_driver.o squid_driver.c
gcc -lm squid_driver.o -o squid_driver
```

The simulation results are printed to the standard output, which is redirected into a file by the > operator, e.g. the output is written into squid.dat by the command below.

Listing D.4: Execution of standalone model using standard output redirection.

```
./squid_driver > squid.dat
```

D.2 Implementation as BeatBox Modules

D.2.1 Enumeration of Variables

Function-like macros are a feature of the C preprocessor, which are routinely used for a variety of purposes within BeatBox modules, for example the enumeration of variables, or setting of initial conditions. As these function-like macros are essential for understanding how BeatBox modules are built, we briefly describe the functionality of the preprocessor macros in this subsection.

The function-like macros used in BeatBox are normally in the format _(<code1>,<code2>), where <code1>, and <code2> contain pieces of code. The function-like macros are processed by the C preprocessor (cpp) according to the definition of a particular macro. An example of such a definition is the following code:

```
#define _(variable,initial) var_##variable,
```

During preprocessing, the string var_ is concatenated (through cpp operator ##) together with the first argument called variable. During the preprocessing stage of the compilation, cpp processes the code and each construct in the specific form, such that the source code is converted to the corresponding expression. For example, a source code defining the initial value of membrane voltage _(V,-80.0) is substituted by var_V,. In this instance the value of -80.0 is ignored.

Once the function-like macro is no longer needed, it can be undefined by #undef _ and redefined again when required. The definition in any of the cases is not constrained syntactically, so it allows more than one definitions within the same code to fit a particular purpose. For example, the same macro that defines the initial value of the membrane voltage _(V,-80.0) from the previous paragraph can be used to assign the initial value in another instance of the model.

The full advantage of using this approach lies in the combination of function-like macros together with the include statements. The include statements include the content of any file into the preprocessed code. The syntax of the include statement is:

```
#include "filename.h"
```

which includes the code in the file *filename.h* into the preprocessed code. If the code contains any defined function-like macros, the cpp expands them in the same manner as described above. So, if the file *filename.h* is imported at different locations, the cpp will substitute the function-like macros according to the latest definition, which can be different in each case.

The specific definition of the function-like macro can be changed depending on the context in which the file is included, resulting in a different piece of code. Any modifications of the included file will have an effect on both places of its inclusion. So, any changes need to be done in only one place, which minimises redundant code, helps to avoid possible errors and simplifies the maintenance of the module.

The following code listing illustrates the idea of the expansion of function-like macros included from header files. The following case shows enumeration of all dynamical variables.

Listing D.5: Enumeration of variables using function-like macros.

```
/* Enumerate the dynamical ("state") variables within the vector
   of dynamical variables */
enum {
  #define _(variable,initial) var_##variable,
  #include "<ionic>_other.h"
  #include "<ionic>_ngate.h"
  #include "<ionic>_tgate.h"
  #undef _
  NV                    /* total number of variables */
};
```

Those header files contain a list of variables and their values (or initial value of dynamical variables) in function-like macros e.g. `_(V,-80.0)`. Notice, that in this case we use `<ionic>` as a placeholder for an arbitrary `ionic` module, e.g. `hh52m` (not to be confused with angle brackets used for including from an include path in cpp).

Notice that due to the use of enumeration the `var_V` is in fact a number, which can be used interchangeably with a number corresponding to the position of a specific function-like macro (starting from 0). Also, the value of `NV` corresponds to the total number of dynamical variables.The macro `NV` is required by the `ionic` module. In the same fashion we define other macros such as the number of “other” (`NO`), tabulated (`NT`) and non-tabulated gating variables (`NN`), as well as the number of tabulated transition rates (`NTAB`).

The counting and initialisation of variables can be done through including separate header files named *<ionic>_<type>.h* into the main file (*<ionic>.c*). Here `<type>` stands for `other` (“other” variables), `ngate` (non-tabulated gating variables), `tgate` (tabulated gating variables), `par` (parameters), `fun` (other voltage-dependent tabulated functions). An example of a routinely used file is shown in Table D.1.

Some `ionic` modules also include large parts of the code to minimise the size of the main file `<ionic>.c`. However, those files are included only once,

Table D.1: Examples of included files with function-like macros.

file name	description
<ionic>_fun.h	functions for other tabulated variables (empty in `<ionic>`)
<ionic>_par.h	parameters and their values
<ionic>_other.h	names and initial conditions of “other” variables
<ionic>_ngate.h	names and initial conditions of gates with non-tabulated transition rates (empty in `<ionic>`)
<ionic>_tgate.h	names and initial conditions of gates with tabulated transition rates
<ionic>_ik.h	definition of I_K model as Markov chain
<ionic>_ina.h	definition of I_Na model as Markov chain

and normally do not use function-like macros. Typically, a module includes the file `<ionic>_ftab.h`, which specifies equations of tabulated voltage-dependent transition rates and other functions, the file `<ionic>_fddt.h`, which includes equations for non-tabulated gating variables and “other” dynamical variables, and `<ionic>_const.h` which contains macro definitions for some physical constants used in tabulated functions, such as temperature and the Faraday constant.

D.2.2 Implementation of Markov Chains

We use the included files to preprocess the function-like macros similarly as described with other dynamical variables. The Markov chain variables are defined within `_(variable, value)` where `variable` determines the name of the Markov chain state and `value` specifies the initial conditions of the variable, e.g. `_(ltypeCzero, 0.948)`. Using this approach we can enumerate the variables of a single Markov chain, which can be used to assign the `dimensionality` of the `channel_str`.

Additional function-like macros serve to initialise the `subchain_str`. Transition rates functions are constructed from macro `_VFUN(variable,expression)`, where `variable` is the name of transition rate and `expression` the mathematical expression that defines it.

The transition matrix is constructed from macro
`_RATE(from,to,direct,reverse)`. This macro specifies the transition rates between two states (`from`, `to`) and the transition rates in between (`direct`, `reverse`) as described on page 168.

The macro `_RATE` helps to generate `TransRatesMat` C-functions that construct a function to construct the transition matrix of a subchain. A macro called `CHANNEL_TR_MATRIX` as shown below generates a function computing transition rates within an `ionic` module.

Listing D.6: Function for population of transition matrix.

```
CHANNEL_TR_MATRIX(<ionic>_ical){
  #define v u[0]
  #define _(n,i)
  #define _VFUN(name,expression) real name=expression;
  #define _RATE(from,to,direct,reverse)          \
    TR_MAT(<channel>,from,to,direct,reverse)
  #include "<ionic>_<channel>.h"
  #undef _RATE
  #undef _VFUN
  #undef _
  #undef v
  return 1;
}
```

Here the `_RATE` expands using another function-like macro `TR_MAT`. The `TR_MAT(chan,from,to,direct,reverse)` is defined in *channel.h* as follows.

Listing D.7: Function-like macro for population of transition matrix.

```
#define TR_MAT(chan,from, to, direct, reverse)\
tr_mat[chan##_##to*NM_##chan+chan##_##from]=direct;\
tr_mat[chan##_##from*NM_##chan+chan##_##from]-=direct;\
tr_mat[chan##_##from*NM_##chan+chan##_##to]=reverse;\
tr_mat[chan##_##to*NM_##chan+chan##_##to]-=reverse;
```

D.2.3 Including Modules into BeatBox

The inclusion of cellular modules into BeatBox requires changes to the BeatBox source files and the files of the GNU build system of BeatBox *configure.ac* (the suffix is an abbreviation from Autoconf) and *Makefile.am* (the suffix is an abbreviation from Automake). Those files are used to create configuration *configure* and build script *Makefile*.

The building of a new `rhs` module starts by including its name into the *rhslist.h* in a form `D(<rhs>)`, as in the following example.

Listing D.8: Specification of new `rhs` modules in *src/rhslist.h*.

```
#if HH
  D(hh)
#endif
```

In this listing the `hh` is the name of the module of the Hodgkin-Huxley model. The `rhs` modules are also required to add a macro definition to the file *on.h* and generate a new file *<rhs>.on* with the same macro. This macro reads as:

```
  #define HH 1
```

The building of a new `ionic` module starts by including its name into the *ioniclist.h* in a form `D(<ionic>)`, as in the following example.

Listing D.9: Specification of new `ionic` modules in *src/ioniclist.h*.

```
  D(hh52)
  D(hh52m)
```

In this listing `hh52` is the name of the module, which is the conversion of `rhs` module `hh` into `ionic` format. In `hh52` the ion channels are implemented as gate models. The `hh52m` is the same model, where the ion channels were converted into Markov chain models as described in Section 2.3.4.

The main module file contains a formal definition of the model. It is expected to be in the file *<name>.c*. The time stepping devices require the module to initialise specific structures and functions.

All files used in the implementation of those modules have to be listed within the `common_sources` variable in *src/Makefile.am* in the BeatBox root directory:

Listing D.10: Specification of new `ionic` and `rhs` modules in *src/Makefile.am*.

```
common_sources = \
  hh52.c hh52_other.h hh52_tgate.h hh52_par.h hh52_ftab.h\
  hh52m.c hh52m_ik.h hh52m_ina.h\
  hh.c \
  HH.on \
  hh.h \
```

this file maintains the list of the source files. The Automake takes care of generating the *Makefile*s specific for compilation on a particular computer architecture, for which the BeatBox executable is targeted. This allows portability of the code to a large variety of different machines.

The template files *Makefile.am* and *configure.ac* (not modified in this example) are processed to generate and install corresponding *configure* and *Makefile* in the appropriate directories. This is done by a command

```
  autoreconf -fi
```

in the root directory of BeatBox (`-f` for force, `-i` for install).

D.3 Hodgkin-Huxley Squid Model

D.3.1 Code Listings of Hodgkin-Huxley Model as `rhs` Module

The listing below shows the main file *hh.c* of the model `hh` which is a minimalist implementation of the original Hodgkin-Huxley code in BeatBox in the `rhs` format.

Listing D.11: Main file *hh.c* of `rhs` module `hh`.

```
/* Hodkin & Huxley model (J Physiol 117:500-544,1952),
 */
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "system.h"
#include "beatbox.h"
#include "HH.on"

#if HH

#include "device.h"
#include "state.h"
#include "bikt.h"
#include "rhs.h"
#include "qpp.h"

/* number of layers of dynamical variables */
#define N 4

static double cub(double x) {return x*x*x;}
static double qrt(double x) {double q=x*x; return q*q;}

typedef struct {
  #define _(n,v) real n;
  #include "hh.h"
  #undef _
  real I; /* current/area = mV/ms*capacitance-trmbr current */
  real IV;/* mV/ms - intercellular current */
} STR;

/* RHS_HEAD expands to full right hand side function for the
   model (including all the dynamical equations):

   int hh(real *u, real *du, Par par, Var var, int ln)

   INPUT ARGUMENTS
   ---------------
   real *u      | pointer to array of states variables
   real *du     | pointer to array of increments of *u
   Par par      | parameter structure
   Var var      | variable structure
   int ln       | number of variables
*/
RHS_HEAD(hh,N) {
  /* DEVICE_CONST copies elements of device structure S */
  #define _(n,v) DEVICE_CONST(real,n)
  #include "hh.h"
  #undef _
  DEVICE_CONST(real,I)
  DEVICE_CONST(real,IV)
  real V = u[0];
  real h = u[1];
  real n = u[2];
  real m = u[3];
  real *dV = du+0;
  real *dh = du+1;
  real *dn = du+2;
  real *dm = du+3;
  real INa, IK, Il;

  real alpm = 0.1 * (-V + 25.0) /
    (exp ((-V + 25.0) / 10.0) - 1.0);
  real betm = 4.0 * exp (-V / 18.0);
  real alph = 0.07 * exp (-V / 20.0);
  real beth = 1.0 / (exp ((-V + 30.0) / 10.0) + 1.0);
  real alpn = 0.01 * (-V + 10.0) /
    (exp ((-V + 10.0) / 10.0) - 1.0);
  real betn = 0.125 * exp (-V / 80.0);

  IK = gK*qrt(n)*(V-VK);
  INa = gNa*cub(m)*h*(V-VNa);
  Il = gl*(V-Vl);

  *dV = -1./C*( IK + INa + Il + I ) + IV;
  *dh = alph*(1-h)-beth*h;
  *dn = alpn*(1-n)-betn*n;
  *dm = alpm*(1-m)-betm*m;
} RHS_TAIL(hh)

/* RHS_CREATE_HEAD expands to intitialise the model. This
   includes assigning values for all model parameters,
   i.e. reading from the script or keeping the default
   values otherwise; and creating the vector of initial
   (steady-state) values of dynamic variables, which will
   be used by time-stepping device to initialise the whole
   medium.

   int
   create_hh(Par *par, Var *var, char *w, real **u, int v0)

   INPUT ARGUMENTS
   ---------------
   Par par      | parameter structure
   Var var      | variable structure
   char *w      | parameters to be assigned from script
   real **u     | pointer to array of states variables
   int v0        | number of entries in states array
*/
RHS_CREATE_HEAD(hh) {
  /* ACCEPTP reads value of named parameter from BBS script */
  #define _(n,v) ACCEPTP(n,v,RNONE,RNONE);
  #include "hh.h"
  #undef _
  ACCEPTP(I,0,RNONE,RNONE);
  ACCEPTP(IV,0,RNONE,RNONE);
  /* allocate an array for results */
  MALLOC(*u,N*sizeof(real));
  /* V_m  - membrane potential (mV) */
  (*u)[0] = 7.0;
  /* h - inactivation gate of sodium current I_Na  */
  (*u)[1] =0.5960;
  /* n - activation gate of potassium current I_K  */
  (*u)[2] =0.3177;
  /* m - activation gate of sodium current I_Na  */
  (*u)[3] =0.0530;
} RHS_CREATE_TAIL(hh,N)

#endif
```

The file *hh.h* included in several places contains a list of variables used in the model, as follows.

Listing D.12: Variables in `hh` module file *hh.h*.

```
/* List of pars  */
/* Name    Value */
_(C,        1.   )
/* maximum conductance of current I_Na (mS/cm^2) */
_(gNa, 120);
/* maximum conductance of current I_K (mS/cm^2)*  */
_(gK, 36);
/* maximum conductance of leakage current: I_l (mS/cm^2) */
_(gl, 0.3);
/* reversal potential of current I_Na (mV)  */
_(VNa, 115);
/* reversal potential of current I_K (mV)  */
_(VK, -12);
/* reversal potential of current I_l (mV)  */
_(Vl, 10.613);
```

The `rhs` module `hh` can be used from the following `bbs` script passed as a command line argument to BeatBox, i.e. `Beatbox hh.bbs`.

Listing D.13: `bbs` script for `rhs` module `hh` from file *hh.bbs*.

```
/*
 * Driver for Hodgkin-Huxley 1952 minimalistic rhs model
 */
def int neqn 4; /* number of layers of state variables */
def real dt 0.01;           /* time step */

/* declare schedule variables */
def real begin;
def real end;
def real T;

/* configuration of the dimensions */
state xmax=1 ymax=1 zmax=1 vmax=neqn+1;

/* Schedule */
k_func name=schedule nowhere=1 pgm={
  T = t*dt;                      /* simulation time */
  begin =eq(T, 0);               /* start of simulation [ms] */
  end   =ge(T, 10.);             /* end of simulation [ms] */
};

/* Reaction substep */
euler v0=0 v1=neqn-1 ht=dt ode=hh par={IV=@4;};

/* define output variable */
```

```
def real v;
sample x0=0 v0=0 result=v;
def real h;
sample x0=0 v0=1 result=h;
def real n;
sample x0=0 v0=2 result=n;
def real m;
sample x0=0 v0=3 result=m;

/* write output to a file */
k_print nowhere=1 when=always file="hh.vtg" append=0
  valuesep="\t" list={T;v;n*n*n*n;m*m*m*h;};

/* end simulation */
stop when=end;
end;
```

D.3.2 Code Listings of Hodgkin-Huxley Model as `ionic` Module with Gate Models of Ion Channels

The `rhs` module can be converted to an ionic module. For that purpose, we need to change the template macros to the `ionic` equivalents. In the first instance, we use all dynamical variables, including gate ion channel models, as “other” variables, calculated using forward Euler. This allows us to compare the solution of an `ionic` module using `rushlarsen` device with its `rhs` counterpart solved by `euler` device. This comparison proved the identity of both results (not shown).

Consequently, we have implemented a C-function calculating voltage-dependent functions, such as the transition rates of the gating variables, and performed necessary changes to specify that the gating variables are now considered to be tabulated gates. This implementation allows us to employ the Rush-Larsen technique.

The listing below shows the main file *hh52.c* of the model `hh52`, which is a minimalist implementation of the original Hodgkin-Huxley code in BeatBox in the `ionic` format.

Listing D.14: `ionic` module `hh52` with gate variables defined in file *hh52.c*.

```
/**
 * IONIC description of the Hodgkin-Huxley 1952 model.
*/

#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "system.h"
#include "beatbox.h"

#include "device.h"
#include "state.h"
#include "bikt.h"
#include "ionic.h"
#include "qpp.h"

/* number of Markov chain models */
#define NMC 0

/* possition of membrane voltage in state vector */
static int V_index = 0;
```

```
/* Enumerate all dynamic variables */
enum
{
#define _(n,i) var_##n,
#include "hh52_other.h"
#include "hh52_tgate.h"
#undef _
  NV    /* total number of variables */
};

/* Enumerate the other (non-gate) variables */
enum
{
#define _(n,i) other_##n,
#include "hh52_other.h"
#undef _
  NO    /* total number of other variables */
};

/* there are none of non-tabulated gate variables */
#define NN 0

/* Enumerate the gates */
enum
{
#define _(n,i) gate_##n,
#include "hh52_tgate.h"
#undef _
  NT    /* total number of tabulated gate variables */
};

/* Enumerate the tabulated transition rates */
enum
{
#define _(n,i) _alp_##n,
#include "hh52_tgate.h"
#undef _
#define _(n,i) _bet_##n,
#include "hh52_tgate.h"
#undef _
/* there are no other tabulated functions in HH52 */
  NTAB  /* total number of tabulated functions */
};

/* The structure containing the parameter values
   for this instance of the model */
typedef struct
{
  /* First go the canonical cell parameters */
#define _(name,default) real name;
#include "hh52_par.h"
#undef _
  /* Then the external current. */
  real IV;
} STR;

/* IONIC_FTAB_HEAD expands to a function defining voltage dependent
   transition rates for tabulation:
   int ftab_hh52(real V, real *values, int ntab)

   INPUT
   -----
   real V       | membrane voltage
   real *values | pointer to array to be filled with Transition
                | rates
   int ntab     | number of tabulated variables

   Returns 1 if succeeds.
*/
IONIC_FTAB_HEAD (hh52)
{
#include "hh52_ftab.h"
  /* Copy the results into the output array values[]. */
#define _(n,i) values[_alp_##n]=alpha_##n;
#include "hh52_tgate.h"
#undef _
#define _(n,i) values[_bet_##n]=beta_##n;
#include "hh52_tgate.h"
#undef _
} IONIC_FTAB_TAIL (hh52);


/* IONIC_FDDT_HEAD expands to a function of right hand sides for the
   computation of increment of other and non-tabulated gates.

   int fddt_hh52(real *u,int nv,real *values,int ntab,Par par,\
         Var var,real *du,int no,real *nalp,real *nbet,int nn)

   INPUT ARGUMENTS
   ---------------
   real *u      | pointer to array of states variables
   int nv       | total number of variables
   real *values | pointer to array of tab. transition rates
   int ntab     | number of tab. transition rates
   Par par      | parameter structure
   Var var      | variable structure
   real *du     | pointer to array of increments of *u
   int no       | number of other variables
   real *nalp   | pointer to array of non-tabulated alphas
   real *nbet   | pointer to array of non-tabulated betas
   int nn       | number of non-tab. gates
*/
IONIC_FDDT_HEAD (hh52, NV, NTAB, NO, NN)
{
  /* Declare the const pars and take their values from struct
     S==par (a formal parameter) */
#define _(name,default) DEVICE_CONST(real,name);
#include "hh52_par.h"
#undef _
  DEVICE_CONST (real, IV);
  /* Declare and assign local variables for dynamic variables
     from state vector */
  /* ..., first for non-gate variables */
#define _(name,initial) real name=u[var_##name];
#include "hh52_other.h"
#undef _
  /* ..., and then for tabulated gate variables */
#define _(name,i) real name=u[var_##name];
#include "hh52_tgate.h"
#undef _
  /* Calculate the rates of non-gate variables */
  real O_K = n * n * n * n;
  real O_Na = m * m * m * h;
  /* currents */
  real I_K = G_K * O_K * (V - E_K);
  real I_Na = G_Na * O_Na * (V - E_Na);
  real I_l = G_l * (V - E_l);

  /* model equations */
  real dot_V = -1.0 / C_m * (I_Na + I_K + I_l);

  /* Copy the calculated rates into the output array du[].  */
  /* Care is taken that all, and only, non-gating variables
     are attended here */
#define _(name,initial) du[other_##name]=dot_##name;
#include "hh52_other.h"
#undef _
  /* Finally add the "external current" parameter values */
  du[V_index] += IV;
} IONIC_FDDT_TAIL (hh52);

/* IONIC_CREATE_HEAD expands to a function which initialises\
   an instance of the model.

   int create_hh52(ionic_str *I,char *w,real **u,int v0)

   INPUT ARGUMENTS
   ---------------
   ionic_str *I | pointer to ionic structure to be initialised
   char *w      | parameters to be assigned from script
   real **u     | pointer to array of states variables
   int v0        | number of entries in states array
*/
IONIC_CREATE_HEAD (hh52)
{
  /* Here we assign the parameter values to the structure
     AND to namesake local variable */
  #define _(name,default) ACCEPTP(name,default,0,RNONE);
  #include "hh52_par.h"
  #undef _

  /* Assign the initial values as given in the *.h files */
  #define _(name,initial) (*u)[var_##name]=initial;
  #include "hh52_other.h"
  #include "hh52_tgate.h"
  #undef _
} IONIC_CREATE_TAIL (hh52, NV);
```

The functions of transition rates are needed in several places in the code and therefore are included from a file *hh52_ftab.h*.

Listing D.15: Transition rates in file *hh52_ftab.h*.

```
/* gate transition rates */
real alpha_n = 0.01 * (-V + 10.0) /
  (exp ((-V + 10.0) / 10.0) - 1.0);
real beta_n = 0.125 * exp (-V / 80.0);

real alpha_m = 0.1 * (-V + 25.0) /
  (exp ((-V + 25.0) / 10.0) - 1.0);
real beta_m = 4.0 * exp (-V / 18.0);

real alpha_h = 0.07 * exp (-V / 20.0);
real beta_h = 1.0 / (exp ((-V + 30.0) / 10.0) + 1.0);
```

Dynamical variables are defined in two separate files since this particular model does not contain non-tabulated gating variables. The first file contains the “other” variables.

Listing D.16: Other variables *hh52_other.h*.

```
/* format: _(name, initial) */
/* V  - membrane potential (mV) */
_(V, 7.0)
```

The second file contains the tabulated gating variables.

Listing D.17: Tabulated gating variables *hh52_tgate.h*.

```
/* format: _(name, initial) */
/* h - inactivation gate of sodium current I_Na  */
_(h, 0.5960)
/* n - activation gate of potassium current I_K  */
_(n, 0.3177)
/* m - activation gate of sodium current I_Na  */
_(m, 0.0530)
```

The default value of parameters are defined in a similar format to dynamical variables. The default value of parameters can be modified from `bbs` script. The file defining the parametes is shown in subsequent listing.

Listing D.18: Parameters of the `hh52` model *hh52_par.h*.

```
/* membrane capacitance */
_(C_m, 1.0)
/* maximum conductance of current I_Na (mS/cm^2) */
_(G_Na, 120);
/* maximum conductance of current I_K (mS/cm^2)*  */
_(G_K, 36);
/* maximum conductance of leakage current: I_l (mS/cm^2) */
_(G_l, 0.3);
```

```
/* the sign from HH1952 is according new convention */
/* reversal potential of current I_Na (mV)  */
_(E_Na, 115);
/* reversal potential of current I_K (mV)  */
_(E_K, -12);
/* reversal potential of current I_l (mV)  */
_(E_l, 10.613);
```

The `ionic` model `hh52` model can be run by using the following `bbs` script passed as command line argument to BeatBox, i.e. `Beatbox hh52.bbs`.

Listing D.19: `bbs` script *hh52.bbs* for `hh52m` `ionic` module with gating variables.

```
/*
 * Driver for Hodgkin-Huxley 1952 minimalistic ionic model
 */
def int neqn 4; /* number of layers of state variables */
def real dt 0.01;           /* time step */

/* declare schedule variables */
def real begin;
def real end;
def real T;

/* configuration of the dimensions */
state xmax=1 ymax=1 zmax=1 vmax=neqn+1;

/* Schedule */
k_func name=schedule nowhere=1 pgm={
  T = t*dt;                     /* simulation time */
  begin =eq(T, 0);              /* start of simulation [ms] */
  end   =ge(T, 10.);            /* end of simulation [ms] */
};

/* Reaction substep */
rushlarsen v0=0 v1=neqn-1 ht=dt ionic=hh52 order=tog
  par={ht=dt};

/* define output variable */
def real v;
sample x0=0 v0=0 result=v;
def real h;
sample x0=0 v0=1 result=h;
def real n;
sample x0=0 v0=2 result=n;
def real m;
sample x0=0 v0=3 result=m;

/* write output to a file */
k_print nowhere=1 when=always file="hh52.vtg" append=0
  valuesep="\t" list={T;v;n*n*n*n;m*m*m*h;};

/* end simulation */
stop when=end;
end;
```

The parameters of the model can be modified using a `par` parameter of the `rushlarsen` device. For instance we could set up different membrane capacitance and sodium conductance by the following call of `rushlarsen` device in `bbs` script.

```
rushlarsen v0=0 v1=neqn-1 ht=dt ionic=hh52 order=tog
  par={ht=dt C_m=2.0 G_Na=130};
```

D.3.3 Code Listings of Hodgkin-Huxley Model as `ionic` Module with Markov Chain Models

To illustrate the functionality of the Matrix Rush-Larsen method we have implemented a minimalist example of an `ionic` module. In this example we have converted the gating variables for both I_{K} and I_{Na} channels from the `hh52` module into Markov chain. The details about the coversion are provided in Subsection 2.3.4. The definition of the Markov chain in the I_{K} model is given by the following listing.

Listing D.20: Definition of states and Matrix of I_{K} Markov chain file *hh52m_ik.h*

```
/* _(state_name, intial_condition) */
_(closed4,(1-n)*(1-n)*(1-n)*(1-n))
_(closed3,4*n*(1-n)*(1-n)*(1-n))
_(closed2,6*n*n*(1-n)*(1-n))
_(closed1,4*n*n*n*(1-n))
_(O_K,n*n*n*n)

/* _RATE(from_state, to_state, direct_TR, reverse_TR) */
_RATE(closed4,closed3,4*alpha_n,beta_n)
_RATE(closed3,closed2,3*alpha_n,2*beta_n)
_RATE(closed2,closed1,2*alpha_n,3*beta_n)
_RATE(closed1,O_K,alpha_n,4*beta_n)
```

The definition of the Markov chain of I_{Na} format is given below.

Listing D.21: Definition of states and Matrix of I_{Na} Markov chain file *hh52m_ina.h*

```
/* _(state_name, intial_condition) */
_(C3,(1-m)*(1-m)*(1-m)*h)
_(C2,3*m*(1-m)*(1-m)*h)
_(C1,3*m*m*(1-m)*h)
_(O_Na,m*m*m*h)

_(I3,(1-m)*(1-m)*(1-m)*(1-h))
_(I2,3*m*(1-m)*(1-m)*(1-h))
_(I1,3*m*m*(1-m)*(1-h))
_(I0,m*m*m*(1-h))

/* _RATE(from_state, to_state, direct_TR, reverse_TR) */
_RATE(C3,C2,3*alpha_m,beta_m)
_RATE(C2,C1,2*alpha_m,2*beta_m)
_RATE(C1,O_Na,alpha_m,3*beta_m)

_RATE(I3,I2,3*alpha_m,beta_m)
_RATE(I2,I1,2*alpha_m,2*beta_m)
_RATE(I1,I0,alpha_m,3*beta_m)

_RATE(I3,C3,alpha_h,beta_h)
_RATE(I2,C2,alpha_h,beta_h)
_RATE(I1,C1,alpha_h,beta_h)
_RATE(I0,O_Na,alpha_h,beta_h)
```

The main file *hh52m.c* which implements the model `hh52m`, is a modification of *hh52.c*.

Listing D.22: `ionic` module `hh52m` with Markov chains in file *hh52m.c*.

```
/**
 * IONIC description of the Hodgkin-Huxley 1952 model.
*/

#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include "system.h"
#include "beatbox.h"

#include "device.h"
#include "state.h"
#include "bikt.h"
#include "ionic.h"
#include "qpp.h"

/* Enumerate the INa Markov chains variables */
enum {
  #define _RATE(n,i,a,b)
  #define _(n,i) ina_##n,
  #include "hh52m_ina.h"
  #undef _RATE
  #undef _
  NM_ina /* total number of INa Markov variables */
};

/* Enumerate the IK Markov chains variables */
enum {
  #define _RATE(n,i,a,b)
  #define _(n,i) ik_##n,
  #include "hh52m_ik.h"
  #undef _RATE
  #undef _
  NM_ik /* total number of IK Markov variables */
};

/* Enumerate Markov chains */
enum {
  MC_ina,
  MC_ik,
  NMC
};

/* possition of membrane voltage in state vector */
static int V_index = 0;

/* Enumerate all dynamic variables */
enum
{
#define _(n,i) var_##n,
#include "hh52_other.h"
#define _RATE(a,b,c,d)
#include "hh52m_ina.h"
#include "hh52m_ik.h"
#undef _RATE
#undef _
  NV    /* total number of variables */
};

/* Enumerate the other (non-gate) variables */
enum
{
#define _(n,i) other_##n,
#include "hh52_other.h"
#undef _
  NO    /* total number of other variables */
};

/* there are none of non-tabulated gate variables */
#define NN 0

#define NT 0
#define NTAB 0

/* The structure containing the parameter values
   for this instance of the model */
typedef struct
{
  /* First go the canonical cell parameters */
#define _(name,default) real name;
#include "hh52_par.h"
#undef _
```

```
  /* Then the external current. */
  real IV;
} STR;

/* IONIC_FTAB_HEAD expands to a function defining voltage
   dependent transition rates for tabulation:
   int ftab_hh52(real V, real *values, int ntab)

   INPUT
   -----
   real V       | membrane voltage
   real *values | pointer to array to be filled with
                |  transition rates
   int ntab     | number of tabulated variables

   Returns 1 if succeeds.
*/
IONIC_FTAB_HEAD (hh52m)
{
} IONIC_FTAB_TAIL (hh52m);

/* CHANNEL_TR_MATRIX expands to a function which fills the
   transition rates matrix:
   int hh52m_{ina,ik}(real *u, real *tr_mat)

   INPUT ARGUMENTS
   ---------------
   real *u      | states vector
   real *tr_mat | matrix to be filled with transtion rates
 */
CHANNEL_TR_MATRIX(hh52m_ina){
  #define V (u[0])
  /* transition rates */
  real alpha_m = 0.1 * (-V + 25.0) /
    (exp ((-V + 25.0) / 10.0) - 1.0);
  real beta_m = 4.0 * exp (-V / 18.0);
  real alpha_h = 0.07 * exp (-V / 20.0);
  real beta_h = 1.0 / (exp ((-V + 30.0) / 10.0) + 1.0);
  #undef V

  #define _(n,i)
  #define _RATE(from,to,direct,reverse)         \
    TR_MAT(ina,from,to,direct,reverse)
  #include "hh52m_ina.h"
  #undef _RATE
  #undef _
  return 1;
}

CHANNEL_TR_MATRIX(hh52m_ik){    /* described above */
  #define V (u[0])
  /* transition rates */
  real alpha_n = 0.01 * (-V + 10.0) /
    (exp ((-V + 10.0) / 10.0) - 1.0);
  real beta_n = 0.125 * exp (-V / 80.0);
  #undef V

  #define _(n,i)
  #define _RATE(from,to,direct,reverse)         \
    TR_MAT(ik,from,to,direct,reverse)
  #include "hh52m_ik.h"
  #undef _RATE
  #undef _
  return 1;
}

/* IONIC_FDDT_HEAD expands to a function of right hand sides
   for the computation of increment of other and non-tabulated
   gates.

   int fddt_hh52(real *u,int nv,real *values,int ntab,Par par,\
         Var var,real *du,int no,real *nalp,real *nbet,int nn)

   INPUT ARGUMENTS
   ---------------
   real *u      | pointer to array of states variables
   int nv       | total number of variables
   real *values | pointer to array of tab. transition rates
   int ntab     | number of tab. transition rates
   Par par      | parameter structure
   Var var      | variable structure
   real *du     | pointer to array of increments of *u
   int no       | number of other variables
   real *nalp   | pointer to array of non-tabulated alphas
   real *nbet   | pointer to array of non-tabulated betas
   int nn       | number of non-tab. gates
*/
IONIC_FDDT_HEAD (hh52m, NV, NTAB, NO, NN)
{
  /* Declare the const pars and take their values from struct
     S==par (a formal parameter) */
#define _(name,default) DEVICE_CONST(real,name);
#include "hh52_par.h"
#undef _
  DEVICE_CONST (real, IV);
  /* Declare and assign local variables for dynamic variables
     from state vector */
  /* ..., first for non-gate variables */
#define _(name,initial) real name=u[var_##name];
#include "hh52_other.h"
#undef _
  /* ..., then the Markov chains. */
#define _RATE(a,b,c,d)
#define _(name,i) real name=u[var_##name];
#include "hh52m_ina.h"
#undef _
#define _(name,i) real name=u[var_##name];
#include "hh52m_ik.h"
#undef _
#undef _RATE
/* currents */
real I_K = G_K * O_K * (V - E_K);
real I_Na = G_Na * O_Na * (V - E_Na);
real I_l = G_l * (V - E_l);

/* model equations */
real dot_V = -1.0 / C_m * (I_Na + I_K + I_l);

  /* Copy the calculated rates into the output array du[].  */
  /* Care is taken that all, and only, non-gating variables
     are attended here */
#define _(name,initial) du[other_##name]=dot_##name;
#include "hh52_other.h"
#undef _
  /* Finally add the "external current" parameter values */
  du[V_index] += IV;
} IONIC_FDDT_TAIL (hh52m);

/* IONIC_CREATE_HEAD expands to a function which
   initialises an instance of the model.

   int create_hh52(ionic_str *I,char *w,real **u,int v0)

   INPUT ARGUMENTS
   ---------------
   ionic_str *I | pointer to ionic structure to be initialised
   char *w      | parameters to be assigned from script
   real **u     | pointer to array of states variables
   int v0       | number of entries in states array
*/
IONIC_CREATE_HEAD (hh52m)
{
  /* Here we assign the parameter values to the structure
     AND to namesake local variable */
  #define _(name,default) ACCEPTP(name,default,0,RNONE);
  #include "hh52_par.h"
  #undef _

  /* Assign the initial values as given in the *.h files */
  /*
     the macro SUBCHAIN(fun_tr, index, min, max, incr, sc) intialises
     the parameters of Markov subchain.
     variable | meaning
     -------------------------------------------------
     fun_tr   | function of transition rates as defined
              | by CHANNEL_TR_MATRIX
     index    | index of control variable for tabulation
              | (negative to avoid tabulation)
     min      | minimal value for tabulation
     max      | maximal value for tabulation
     incr     | increment in the tabulation
     sc       | scale for tabulation
     -------------------------------------------------
  */
  /* intialize INa Markov chain */
  sbch = &(ch->subchain[0]);
  ch->dimension = NM_ina;
  SUBCHAIN(hh52m_ina, -1, -200, 200, 0.01, 0);
  /* intialize IK Markov chains */
  ch += 1;
  sbch = &(ch->subchain[0]);
  ch->dimension = NM_ik;
  SUBCHAIN(hh52m_ik, -1, -200, 200, 0.01, 0);

  /* asign gates for computation of MCs initial conditions */
  #define _(name, initial) real name = initial;
  #include "hh52_tgate.h"
  #undef _

  #define _(name,initial) (*u)[var_##name]=initial;
  #include "hh52_other.h"
  #define _RATE(a,b,c,d)
  #include "hh52m_ina.h"
  #include "hh52m_ik.h"
  #undef _RATE
  #undef _
} IONIC_CREATE_TAIL (hh52m, NV);
```

The *hh52m* shares some included files with the *hh52* model, such as the file for “other” variables *hh52_other.h*, initial values of gating variables used to determine the initial values of Markov chains states *hh52_tgate.h* and parameters *hh52_par.h*.

The bbs script for hh52m module has only a few differences from *hh52.bbs*. The main difference is obviously the name of the ionic module in the rushlarsen device call, which now reads as hh52m. Another difference is the higher number of dynamical variables in hh52m, which results from the conversion of gate variables into corresponding Markov chain models. This is reflected in the parameter neqn, which specifies the number of dynamical variables in bbs script. Now it has to be set to the specific value for hh52m which is neqn=14. Finally, to save the results we sample the open probabilities of both I_{Na} and I_{K}. The corresponding bbs script is shown below.

Listing D.23: bbs script *hh52m.bbs* for hh52m ionic module with Markov chains.

```
/*
 * Driver for Hodgkin-Huxley 1952 minimalistic ionic model
 */
def int neqn 14; /* number of layers of state variables */
def real dt 0.01;           /* time step */

/* declare schedule variables */
def real begin;
def real end;
def real T;

/* configuration of the dimensions */
```

```
state xmax=1 ymax=1 zmax=1 vmax=neqn+1;

/* Schedule */
k_func name=schedule nowhere=1 pgm={
  T = t*dt;                       /* simulation time */
  begin =eq(T, 0);                /* start of simulation [ms] */
  end   =ge(T, 10.);              /* end of simulation [ms] */
};

/* Reaction substep */
rushlarsen v0=0 v1=neqn-1 ht=dt ionic=hh52m order=tgo
  exp_mc=ntabmrl  par={ht=dt};

/* define output variable */
def real v;
sample x0=0 v0=0 result=v;
/* ik channel */
def real O_Na;
sample x0=0 v0=4 result=O_Na;
/* ina channel */
def real O_K;
sample x0=0 v0=13 result=O_K;

/* write output to a file */
k_print nowhere=1 when=always file="hh52m.vtg" append=0
  valuesep="\t" list={T;v;O_K;O_Na;};

/* end simulation */
stop when=end;
end;
```

D.4 TenTusscher-Panfilov (2006) Model

We have noticed that some published models include the forward Euler in equations for computing calcium dynamics. The numerical algorithm used for integrating the system of differential equations in BeatBox has to allow for the flexibility of using any numerical integration method, i.e. the calcium dynamics have to be implemented as pure dynamic equation.

For this purpose we suggest a method for transforming the equations of calcium dynamics, which include the forward Euler integration, into pure dynamic equations.

This section was developed with important contribution from Vadim Biktashev.

D.4.1 Calcium Dynamics

The equation (2.32) describes the evolution of total calcium dynamics as a combination of free calcium and buffered calcium. The chemical reactions of the buffering are assumed to be fast, hence can be adiabatically eliminated. Equation (2.32) is multiplied by the denominator of the second term on the right hand side and collecting terms with $[\mathrm{Ca}^{+2}]_f$ together yields a quadratic equation

$$[\mathrm{Ca}^{+2}]_f^2 + \Big(k + B_t - [\mathrm{Ca}^{+2}]_t\Big)\,[\mathrm{Ca}^{+2}]_f - k[\mathrm{Ca}^{+2}]_t = 0. \tag{D.1}$$

We find both roots of the quadratic equation and considering the physical constraints of the concentration, which is restricted only to positive values, the only feasible solution is a unique positive root

$$[\mathrm{Ca}^{+2}]_f = \frac{1}{2}\left(-(k + B_t - [\mathrm{Ca}^{+2}]_t) + \sqrt{(k + B_t - [\mathrm{Ca}^{+2}]_t)^2 + 4k[\mathrm{Ca}^{+2}]_t}\right), \tag{D.2}$$

where the total concentration $[\mathrm{Ca}^{+2}]_t$ is described by a differential equation involving the ionic fluxes of Ca^{2+} in and out of the corresponding compartment.

To summarise, we write down the equations used for the computation of the Calcium dynamics. Function $f([\mathrm{Ca}^{+2}], \ldots)$ is a given function defined within the cell model, binding $k = k_{\mathrm{on}}/k_{\mathrm{off}}$, and constants $B_t, k_{\mathrm{on}}, k_{\mathrm{off}} \in \mathbb{R}^+$ so that:

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]_t}{\mathrm{d}t} = f(\ldots), \tag{D.3}$$

$$[\mathrm{Ca}^{+2}]_f = \frac{1}{2}\left(-(k + B_t - [\mathrm{Ca}^{+2}]_t) + \sqrt{(k + B_t - [\mathrm{Ca}^{+2}]_t)^2 + 4k[\mathrm{Ca}^{+2}]_t}\right), \tag{D.4}$$

$$[\mathrm{Ca}^{+2}]_b = \frac{B_t[\mathrm{Ca}^{+2}]_f}{[\mathrm{Ca}^{+2}]_f + k}, \tag{D.5}$$

which satisfies the requirement on BeatBox `ionic` modules, where the “other” variables have to be implemented as a system of differential equations. In the model published by tenTusscher-Panfilov (2006), the computation of calcium dynamics includes the time-stepping algorithm within the code. For the implementation as `ionic` model we have converted the system into dynamic equations as described in the next subsection.

D.4.2 Calcium Computation in the TenTusscher-Panfilov Model

The Ca^{2+} dynamics within the TTP model are calculated for three compartments: sarcoplasmic reticulum (SR); dyadic subspace (SS), which is the compartment in the proximity of the cellular membrane and SR; and bulk intracellular calcium.

The algorithm computing the calcium dynamics in the SR in the TTP code is given as

$$[\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} = \frac{B_{\mathrm{SR}}[\mathrm{Ca}^{+2}]_{\mathrm{SR}}}{\left([\mathrm{Ca}^{+2}]_{\mathrm{SR}} + k_{\mathrm{SR}}\right)}, \tag{D.6}$$

$$\Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} = \Delta t\left(I_{\mathrm{up}} - I_{\mathrm{rel}} - I_{\mathrm{leak}}\right), \tag{D.7}$$

$$b_{\mathrm{SR}} = B_{\mathrm{SR}} - [\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} - \Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} - [\mathrm{Ca}^{+2}]_{\mathrm{SR}} + k_{\mathrm{SR}}, \tag{D.8}$$

$$c_{\mathrm{SR}} = k_{\mathrm{SR}}\left([\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} + \Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} + [\mathrm{Ca}^{+2}]_{\mathrm{SR}}\right), \tag{D.9}$$

$$[\mathrm{Ca}^{+2}]_{\mathrm{SR}} = \frac{\sqrt{b_{\mathrm{SR}}^2 + 4c_{\mathrm{SR}}} - b_{\mathrm{SR}}}{2}. \tag{D.10}$$

where $[\mathrm{Ca}^{+2}]_{\mathrm{CSQN}}$ represents buffered calcium and $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ represents free calcium concentration. I_{rel} is calcium induced calcium release (CICR) current; I_{up} is

SR Ca^{+2} pump current; I_{leak} is SR Ca^{2+} leak current. The coefficients $k_{\mathrm{SR}} = 0.3$, and $B_{\mathrm{SR}} = 10.0$. Time and calcium concentration increments in one time step are denoted by Δt and $\Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ respectively.

This method computes the $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}$ by the forward Euler method, which is implemented into (D.8) and (D.9) as

$$\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}} = [\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} + [\mathrm{Ca}^{+2}]_{\mathrm{SR}} + \Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}}, \tag{D.11}$$

so that the higher order method cannot be applied unless we reformulate the system as pure dynamical equations.

D.4.3 Differential Equations for Calcium

For the purposes of BeatBox `ionic` modules, the calcium concentration for $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}$, $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SS}}$ and $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{i}}$ should be implemented as pure dynamical equations. The particular equations for each compartment are constructed by substituting the subsequent formulas into the equations (D.3)–(D.5). For the sarcoplasmic reticulum $[\mathrm{Ca}^{+2}]_t = \overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}$ we use

$$\begin{aligned}
f(\ldots) = \frac{\mathrm{d}\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}}{\mathrm{d}t} =& I_{\mathrm{up}}([\mathrm{Ca}^{2+}]_i) - \\
& I_{\mathrm{rel}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}}, O([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}})) - \\
& I_{\mathrm{leak}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{2+}]_i), && \text{(D.12)}\\
B_t = B_{\mathrm{SR}} =& 10.0, && \text{(D.13)}\\
k = k_{\mathrm{SR}} =& 0.3, && \text{(D.14)}
\end{aligned}$$

for the subspace $[\mathrm{Ca}^{+2}]_t = \overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SS}}$ we use

$$\begin{aligned}
f(\ldots) = \frac{\mathrm{d}\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SS}}}{\mathrm{d}t} =& -\frac{V_{\mathrm{c}}}{V_{\mathrm{SS}}} I_{\mathrm{xfer}}([\mathrm{Ca}^{+2}]_{\mathrm{ss}}, [\mathrm{Ca}^{2+}]_i) + \\
& \frac{V_{\mathrm{SR}}}{V_{\mathrm{SS}}} I_{\mathrm{rel}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}}, O([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}})) - \\
& \frac{C_m}{2V_{\mathrm{SS}}F} I_{\mathrm{Ca}(L)}(d(\mathrm{V_m}), f_1(\mathrm{V_m}), f_2(\mathrm{V_m}), f_3([\mathrm{Ca}^{+2}]_{\mathrm{ss}})) && \text{(D.15)}\\
B_t = B_{\mathrm{SS}} =& 0.4 && \text{(D.16)}\\
k = k_{\mathrm{SS}} =& 0.00025, && \text{(D.17)}
\end{aligned}$$

and in the bulk intracellular space $[\mathrm{Ca}^{+2}]_t = \overline{[\mathrm{Ca}^{+2}]}_{\mathrm{i}}$ we use

$$\begin{aligned}
f(\ldots) = \frac{\mathrm{d}\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{i}}}{\mathrm{d}t} =& -\frac{C_m}{2V_{\mathrm{c}}F}\Big(I_{b\mathrm{Ca}}(\mathrm{V_m}) + I_{p\mathrm{Ca}}([\mathrm{Ca}^{2+}]_i) - 2I_{\mathrm{NaCa}}(\mathrm{V_m}, [\mathrm{Na}^+]_i, [\mathrm{Ca}^{2+}]_i)\Big) - \\
& \frac{V_{\mathrm{SR}}}{V_{\mathrm{c}}}\Big(I_{\mathrm{up}}([\mathrm{Ca}^{2+}]_i) - I_{\mathrm{leak}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{2+}]_i)\Big) +
\end{aligned}$$

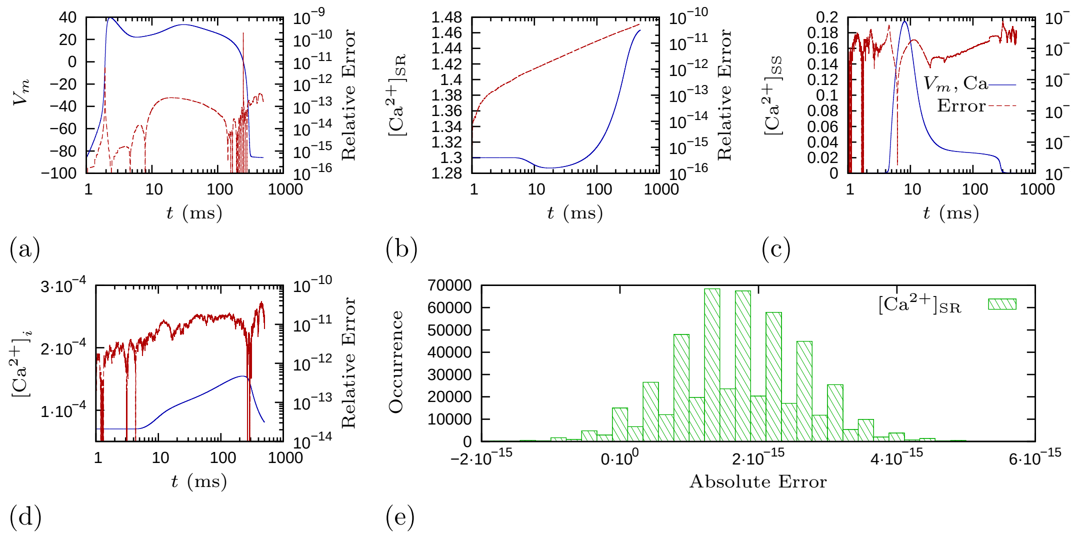


Figure D.1: Comparison of author's and reformulated algorithm for computation of calcium dynamics: (a) Membrane potential $\mathrm{V_m}$, (b) calcium concentration in sarcoplasmic reticulum $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$, (c) subspace $[\mathrm{Ca}^{+2}]_{\mathrm{ss}}$, (d) intracellular calcium $[\mathrm{Ca}^{2+}]_i$ (blue line shows simulated traces – left axis, red line shows the difference between results of the authors' algorithm and reformulated code – right axis), (e) distribution of absolute differences in one time step between the authors' algorithm and reformulated code for $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$. Simulations were done using the forward Euler method with a time step of $\Delta t = 1\ \mu\mathrm{s}$.

$$I_{\mathrm{xfer}}([\mathrm{Ca}^{+2}]_{\mathrm{ss}}, [\mathrm{Ca}^{2+}]_i) \tag{D.18}$$

$$B_t = B_i = 0.2 \tag{D.19}$$

$$k = k_i = 0.001, \tag{D.20}$$

where F is the Faraday constant; V_{c}, V_{SR}, V_{SS} are volumes of intracellular space, sarcoplasmic reticulum and subspace respectively; C_m is the cellular membrane capacitance; ionic currents are denoted by $I_s(\ldots)$ with corresponding subscripts substituted for s and dependent on other dynamical variables given in the model description. The other dynamical variables are: membrane voltage $\mathrm{V_m}$; calcium concentrations $[\mathrm{Ca}^{2+}]_i$, $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$, $[\mathrm{Ca}^{+2}]_{\mathrm{ss}}$; open probability $O([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}})$ of RyR channel; and open probability of gating variables $d(\mathrm{V_m})$, $f_1(\mathrm{V_m})$, $f_2(\mathrm{V_m})$, $f_3([\mathrm{Ca}^{+2}]_{\mathrm{ss}})$ for $I_{\mathrm{Ca}(L)}$ current.

D.4.4 Comparison of Algorithms for Computation of Calcium Dynamics

The Figure D.1 shows the original (TTP) implementation (free calcium as dynamical variables) compared with the reformulated algorithm (total calcium concentration is as dynamical variable).

The error of the computation is normally defined as the deviation from the exact solution. Because both methods only calculate approximate solution we compare them to each other, to demonstrate that both offer similar results.The relative error for our purposes is defined as the relative difference between the two methods and calculated according to the following formula:

$$\epsilon = \left| \frac{A - D}{A} \right|, \tag{D.21}$$

where A is the value from the authors' code and D is the corresponding dynamical variable using the reformulated algorithm.

From the Figure D.1(a-d) we see, that the difference between the TTPs' and updated algorithm is around 10^{-11}. Although this value of error seems rather high, we can show that such a result is consistent.

The iterative solvers introduce a small truncation error in each iteration. $\mathrm{BeatBox}$ uses variables in `double float` precision. This means the number of significant digits is 15. Any consequent digits are lost so the truncation error in each iteration is of the order of 10^{-15}. The error at individual steps can accumulate or compensate the global error for the simulation.

To see the average effect of the error in our simulations we analyse the absolute numerical error in one time step. We implement both the TTPs' and updated algorithm side by side in one code. The calcium concentrations from the updated algorithm D are used for the calculations of all of the cellular compartments. Besides that we obtain the calcium concentrations using the TTPs' algorithm based on the calcium concentrations obtained in the updated algorithm D in the previous time step, and we denote these results by A^*.

The error in this experiment is than defined as the difference between the updated method and the TTPs' method $(D - A^*)$. Figure D.1e shows the distribution of the values of error during the simulation. The average error is skewed towards 10^{-15}.

As seen from the relative error $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ in Figure D.1 panel (c) the error reaches values about 10^{-11} after 100 ms with a time step of $\Delta t = 1\ \mu s$. This means that if the average error is 10^{-15} after 10^5 calculations (100 ms $\times 1000$ steps) we can expect an error around 10^{-10}. We have to bear in mind that this is a rough estimation as the two simulations are not exactly identical. However, it is a clear indication that a plausible source of error is due to the accumulation of truncation errors.

The truncation errors are skewed because the increment of the concentration of calcium is applied to different variable. The TTPs' algorithm increases the free calcium concentration $[\mathrm{Ca}^{+2}]_f$. In the approach used in the updated code, the $\Delta[\mathrm{Ca}^{+2}]$ increases the total calcium concentration. The steady-state approximation of calcium buffering then determines the free calcium concentration. However, as the difference is only to the order of 10^{-10} for $100\ \mathrm{ms}$ the algorithms are equivalent for practical purposes.

D.4.5 Implementation of TenTusscher-Panfilov in `ionic` Format

BeatBox implementation of the TTP model has already been present in form of `rhs` module. To use the `rushlarsen` device for solving gate variables using Rush-Larsen method, we have to convert the `rhs` module to `ionic` format. The specifications of the `ionic` module is described in Subsection 6.4.2. The `ionic` module has to define all the equations as the ODEs, which requires a modification of the equations for the calcium dynamical as has been described in the previous subsection.

The dynamical variables are then divided into three groups: so called (1) tabulated gates, (2) non-tabulated gates, and (3) "other" variables. The transition rates of tabulated gates depend on the voltage. Those variables and their initial conditions are defined in the file *ttp2006rl_tgate.h*. This includes the gates for currents:

- I_{Na} — M, H, J gates;
- $I_{\mathrm{K}r1}$ — Xr1 gate;
- $I_{\mathrm{K}r2}$ — Xr2 gate;
- $I_{\mathrm{K}s}$ — Xs gate;
- I_{to1} — R, S gates;
- I_{Ca} — D, F, F2 gates.

The transition rates of non-tabulated gates depend on variables other than voltage and are computed on-the-fly. Those variables and their initial conditions are defined in the file *ttp2006rl_ngate.h*. This includes the gates for currents:

- I_{Ca} — FCaSS gate;
- I_{rel} (RyR) — RR gate.

Finally, the "other" variables include non-gating variables such as voltage and ionic concentrations. Those variables and their initial conditions are defined in the file *ttp2006rl_others.h*. This includes the following dynamical variables:

- $\mathrm{V_m}$ — voltage;
- $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{i}}$ — intracellular calcium concentration;
- $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}$ — total calcium concentration in sarcoplasmic reticulum;
- $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SS}}$ — total calcium concentration in subspace;
- $[\mathrm{Na}^{+}]_i$ — sodium concentration;
- $[\mathrm{K}^{+}]_i$ — potassium concentration.

The dynamics of calcium refer to the total calcium concentration rather than the free calcium concentration as in the authors' code.

Figure D.2 compares the simulation results by the `rhs` module `ttp06` with the solution of the `ionic` module `ttp2006` and shows the convergence in the time-step and step in the tabulation of the tabulated gates. The results of the figure confirm expected convergence with step size and step in the tabulation grid.

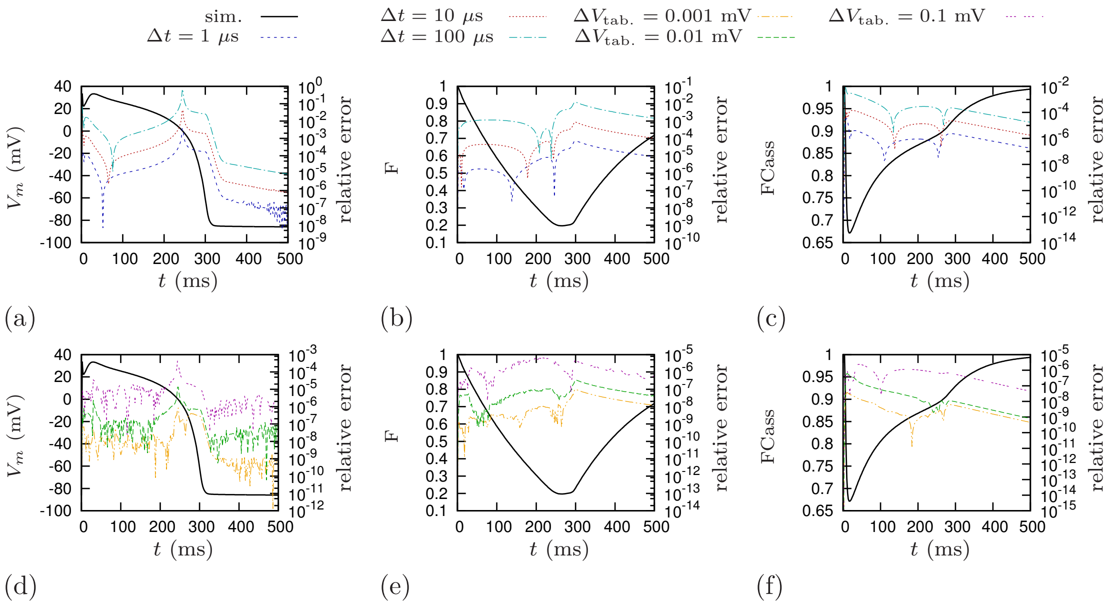


Figure D.2: Convergence of the solution in time-step (top row) and voltage-step in tabulation (bottom row) for tenTusscher-Panfilov (2006) [29] model. The panels (a), (d) show the membrane voltage $\mathrm{V_m}$ computed as “other” variable; the panels (b), (e) show F gate of the I_{Ca} current computed as tabulated gating variable; and the panels (c), (f) show the gate FCaSS of the I_{Ca} computed as non-tabulated gating variables. The convergence in time-step is compared with a reference solution of rhs model solved with the $\Delta t = 0.1\ \mu$s, the convergence in voltage step in the tables is compared with reference solution of the same ionic without tabulation (transition rates computed on-the-fly).

These results also confirm that the difference in the computation of calcium dynamics, as discussed in the previous section, is insignificant compared to the errors due to the increase in step sizes. The difference in the algorithm for the computation of calcium dynamics between rhs module and ionic module only accounts for errors up to a magnitude of 10^{-10}, which is several orders of magnitude lower than the errors observed in the step-size convergence.

Bibliography

[1] Gregory M Faber, Jonathan Silva, Leonid Livshitz, and Yoram Rudy. Kinetic properties of the cardiac L-type Ca^{2+} channel and its role in myocyte electrophysiology: a theoretical investigation. *Biophys J*, 92(5):1522–1543, Mar 2007.

[2] Colleen E Clancy and Yoram Rudy. Na^{+} channel mutation that causes both Brugada and long-QT syndrome phenotypes: a simulation study of mechanism. *Circulation*, 105(10):1208–1213, Mar 2002.

[3] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol*, 117(4):500–544, Aug 1952.

[4] Vladimir E. Bondarenko. A compartmentalized mathematical model of the β1-adrenergic signaling system in mouse ventricular myocytes. *PLoS One*, 9(2):e89113, 2014.

[5] W. G. Wier and D. T. Yue. Intracellular calcium transients underlying the short-term force-interval relationship in ferret ventricular myocardium. *J Physiol*, 376:507–530, Jul 1986.

[6] Bertil Hille. *Ionic Channels of Excitable Membranes*. Sinauer Associates Inc, second edition, 1992.

[7] T. Kispersky and J. A. White. Stochastic models of ion channel gating. *Scholarpedia*, 3(1):1327, 2008. revision #137554.

[8] F. Bezanilla and C. M. Armstrong. Inactivation of the sodium channel. I. Sodium current experiments. *J Gen Physiol*, 70(5):549–566, Nov 1977.

[9] C. M. Armstrong and F. Bezanilla. Inactivation of the sodium channel. II. gating current experiments. *J Gen Physiol*, 70(5):567–590, Nov 1977.

[10] G. W. Beeler and H. Reuter. Reconstruction of the action potential of ventricular myocardial fibres. *J Physiol*, 268(1):177–210, Jun 1977.

[11] R. W. Hadley and W. J. Lederer. Ca^{2+} and voltage inactivate Ca^{2+} channels in guinea-pig ventricular myocytes through independent mechanisms. *J Physiol*, 444:257–268, Dec 1991.

[12] J. P. Imredy and D. T. Yue. Mechanism of Ca^{2+}-sensitive inactivation of L-type Ca^{2+} channels. *Neuron*, 12(6):1301–1318, Jun 1994.

[13] M. S. Jafri, J. J. Rice, and R. L. Winslow. Cardiac Ca^{2+} dynamics: the roles of ryanodine receptor adaptation and sarcoplasmic reticulum load. *Biophys J*, 74(3):1149–1168, Mar 1998.

[14] A. N. Tikhonov. Systems of differential equations containing small parameters in the derivatives. *Mat. Sb. (N.S.)*, 31(73)(3):575–586, 1952.

[15] Neil Fenichel. Geometric singular perturbation theory for ordinary differential equations. *Journal of Differential Equations*, 31(1):53 – 98, 1979.

[16] V.N. Biktashev. Envelope equations for modulated non-conservative waves. *IUTAM Symposium Asymptotics, Singularities and Homogenisation in Problems of Mechanics*, 5(1):11, 2003.

[17] S. Rush and H. Larsen. A practical algorithm for solving dynamic membrane equations. *IEEE Trans Biomed Eng*, 25(4):389–392, Jul 1978.

[18] Dmitrii E Makarov. Some mathematical properties of master equations, 2011. http://makarov.cm.utexas.edu/resources/Lecture-notes,-tutorials-etc./master_equations.pdf.

[19] Rebecca Suckley and Vadim N Biktashev. Comparison of asymptotics of heart and nerve excitability. *Phys Rev E Stat Nonlin Soft Matter Phys*, 68(1 Pt 1):011902, Jul 2003.

[20] W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team, 2014. `http://www.sagemath.org`.

[21] Gregory M Faber, Jonathan Silva, Leonid Livshitz, and Yoram Rudy. *Source code of Faber et al. (2007) cellular model*. http://rudylab.wustl.edu/research/cell/code/Faber_LRd_CaL_2007.zip.

[22] J. Douglas Faires Richard L. Burden. *Numerical Analysis, 9th Edition*. Brooks Cole, 9 edition, 2010.

[23] OpenCourseWare. Numerical methods for partial differential equations, operator splitting. online, spring 2009. http://ocw.mit.edu/courses/mathematics/18-336-numerical-methods-for-partial-differential-equations-spring-2009/lecture-notes/MIT18_336S09_lec20.pdf.

[24] Mauro Perego and Alessandro Veneziani. An Efficient Generalization Of The Rush-Larsen Method For Solving Electro-Physiology Membrane Equations. *Electronic Transactions on Numerical Analysis*, 35:234–256, 2009.

[25] Joakim Sundnes, Robert Artebrant, Ola Skavhaug, and Aslak Tveito. A second-order algorithm for solving dynamic cell membrane equations. *IEEE Trans Biomed Eng*, 56(10):2546–2548, Oct 2009.

[26] BeatBox developers. Source code of beatbox package. online, accessed 2015. http://empslocal.ex.ac.uk/people/staff/vnb262/software/BeatBox/.

[27] R. McFarlane. *High-Performance Computing for Computational Biology of the Heart*. PhD thesis, University of Liverpool, 2010.

[28] BeatBox developers. Beatbox home page. online, accessed 2015. http://empslocal.ex.ac.uk/people/staff/vnb262/software/BeatBox/.

[29] K. H W J ten Tusscher and A. V. Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *Am J Physiol Heart Circ Physiol*, 291(3):H1088–H1100, Sep 2006.

[30] Gregory M Faber and Yoram Rudy. Calsequestrin mutation and catecholaminergic polymorphic ventricular tachycardia: a simulation study of cellular mechanism. *Cardiovasc Res*, 75(1):79–88, Jul 2007.

[31] SM Cox and PC Matthews. New instabilities in two-dimensional rotating convection and magnetoconvection. *Physica D: Nonlinear Phenomena*, 149(3):210–229, 2001.

[32] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[33] Donald M. Bers. Cardiac excitation-contraction coupling. *Nature*, 415(6868):198–205, Jan 2002.

[34] Wei Chen, Mesfin Asfaw, and Yohannes Shiferaw. The statistics of calcium-mediated focal excitations on a one-dimensional cable. *Biophysical Journal*, 102(3):461 – 471, 2012.

[35] Enrique Alvarez-Lacalle, Blas Echebarria, Jon Spalding, and Yohannes Shiferaw. Calcium alternans is due to an order-disorder phase transition in cardiac cells. *Phys Rev Lett*, 114(10):108101, Mar 2015.

[36] C. H. Luo and Y. Rudy. A dynamic-model of the cardiac ventricular action-potential .1. simulations of ionic currents and concentration changes. *Circulation Research*, 74(6):1071–1096, June 1994.

[37] P. C. Viswanathan, R. M. Shaw, and Y. Rudy. Effects of I-Kr and I-Ks heterogeneity on action potential duration and tts rate dependence - A simulation study. *Circulation*, 99(18):Amer Heart Assoc; Hoechst Marion Roussel Inc, Kansas City, May 1999.

[38] J. L. Zeng, K. R. Laurita, D. S. Rosenbaum, and Y. Rudy. Two components of the delayed rectifier K$^+$ current in ventricular myocytes of the guinea-pig type - theoretical formulation and their role in repolarization. *Circulation Research*, 77(1):140–152, July 1995.

[39] William T. Vetterling Brian P. Flannery William H. Press, Saul A. Teukolsky. *Numerical Recipes; The Art of Scientific Computing*. Cambridge University Press, 2007.

[40] C. Wilkinson, H.H.; Reish. *Handbook for Automatic Computation*. Springer, 1971.

[41] Jack Dongarra. Freely available software for linear algebra (may 2013). web page, May 2013. Table on Sparse Eigenvalue Solvers; retrieved 26 January 2015.

[42] M. Galassi et al. *GNU Scientific Library Reference Manual*. 2009.