Contents

List of Figures

List of Tables

Typographical conventions

- The default font for the text is Arial (Helvetica) 12pt.
- The file names are set in monospace italic font e.g. *`rushlarsen.c`*.
- Name of software packages are in roman font e.g. BeatBox.
- Short code listings within the text are shown in monospace font e.g. `euler()`
- Long code listing are shown in monospace font on gray background e.g.

```
/* stimulation parameters */
def real Period 1.0e3;
def real Ampl 80.0;
```

Chapter 1

Introduction

Bioelectricity is one of the essential regulation mechanisms of the body. It allows the processing of sensory impulses, the transmission of information in nervous system, and also triggers contractions of the muscles. A dysfunction of correct function of the electrophysiological mechanisms can lead to a severe damage of the organism.

In the heart, those mechanisms allow a constant pumping of the blood into the body and so facilitate the circulation of blood, which supply the oxygen and nutrients throughout the organism. Heart diseases are one of the most prevalent causes of death. The effective treatment of the cardiac failure should be based on deep understanding of the mechanisms that govern it.

1.1 Hodgkin-Huxley Model

1.1.1 Model of a Cell

The electrophysiology is based on the study of excitable cells. Anatomically, the cells are bound by a cellular membrane that separates the intra and extracellular environments. Those environments contains an electrolyte, which is a water solution of electrically charged particles, such as ions of sodium and potassium. The concentrations differs inside and outside of the cell which causes the polarisation of the membrane.

Under a certain conditions the membrane becomes permeable for certain types of ions, which then diffuse between the environments driven by the concentration and electric gradient. The permeability of specific ions is determined by a complex processes in the cells which cause an electric excitation called action potential. The flow of ions can be expressed as electric current.

A breakthrough discovery was done in 1952, when Hodgkin and Huxley developed the theory for the electric processes in cells[4]. They expressed the cellular membrane in a form of a diagram as shown on the Figure 1.1. The cellular membrane is a lipid layer that is not conductive but due to electrical forces acts as a capacitor. The permeability of the membrane is expressed as the conductance which is specific for every kind of ions. Finally, there are mechanisms which uses the chemical processes to recover the concentration difference between the environments.

The membrane currents have to obey a Kirshoff's law for current which states that that all the currents entering the membrane will exit it on the other side. This law is expressed as

$$\sum_s I_s = 0, \tag{1.1}$$

where I_s represent the currents corresponding to the capacitor I_C, sodium I_{Na} potassium I_{K} and leakage current I_l.

The relation of the capacitive current correspond to the equation of the charging of capacitor obtained from the equation of the capacitor

$$I_C(t) = C\frac{\mathrm{dV_m}}{\mathrm{d}t} \tag{1.2}$$

where the $\mathrm{V_m}$ is a membrane voltage which corresponds to the difference of the electric potentials between the intracellular and extracellular environment. The currents for sodium, potassium and leakage current 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

$$I_{\mathrm{K}}(t) = g_{\mathrm{K}}(t)(\mathrm{V_m} - E_{\mathrm{K}}), \tag{1.3a}$$

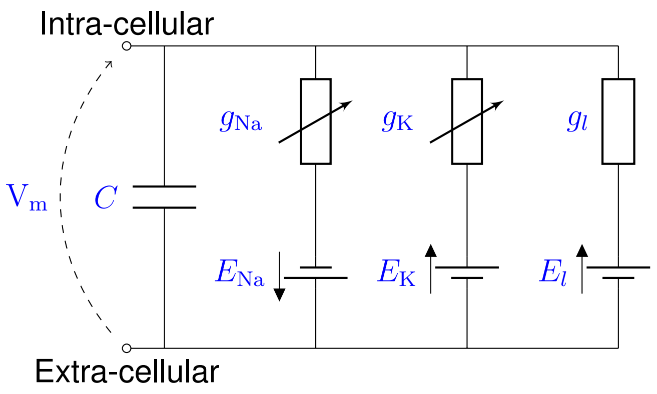


Figure 1.1: Electric diagram of cellular membrane[4]. Each branch of the diagram represent a specific process on the cell. From the left to right it is the capacitance of the membrane, sodium, potassium and leakage current.

$$I_{\mathrm{Na}}(t) = g_{\mathrm{Na}}(t)(\mathrm{V_m} - E_{\mathrm{Na}}), \tag{1.3b}$$
$$I_l(t) = g_l(\mathrm{V_m} - E_l) \tag{1.3c}$$

where g_{K} and g_{Na} and g_l quantify the permeability of the membrane and E_{K}, E_{Na} and E_l is electrochemical energy dependent on the concentrations of particular ions which are considered constant.

The equation for the membrane then gets the form

$$\frac{\mathrm{dV_m}}{\mathrm{d}t} = -\frac{1}{C}\left(I_{\mathrm{Na}} + I_{\mathrm{K}} + I_l\right). \tag{1.4}$$

1.1.2 Ionic Currents

The electrical conductance of the ions could not be derived from the first principles. Instead a theoretical model aims to describe the observed phenomena in the neural cell. In their experiments they measured ionic currents for sodium and potassium ions on giant axon of a squid.

The conductance of sodium and potassium ions changes in time as in the equations (1.3b) and (1.3a). The conductance of the potassium is described by

$$g_{\mathrm{K}} = \bar{g_{\mathrm{K}}} n^4, \tag{1.5}$$

where $\bar{g_{\mathrm{K}}}$ is the maximum conductance. The permeability of ionic membrane for currents is due to large protein molecules residing in the cellular membrane, however this has not been known when the Hodgkin-Huxley developed that model. However, as it simplifies the description of a cellular model and is more realistic, we will use a notion of ionic channel in what is now known as a gate model.

The gate model assumes that the conformation of the channel molecule can be expressed by a finite number gates, that can be in open or closed position. When all those gates are open, the channel allows the ions pass through. The variable then n corresponds to the proportion of open gates. The value of n is described by a differential equation

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(1-n) - \beta_n n, \tag{1.6}$$

where α_n corresponds to the probability of an closed gate to open and β_n corresponds to the probability of an open gate to close per unit of time. This approach assumes an infinite number of ionic channels such that the value of n is a real number, which in the case of a large number of channels is a reasonable approximation.

In the Hodgkin-Huxley model the transition rates were found from experimentally using so called voltage-clamp procedure. In this procedure two electrodes are submerged in a solution together with the measured axon, such that one of the electrodes sense the intracellular, while the other the extracellular environment. Then an external voltage of a constant value is applied to the cell and the time evolution of the current is recorded until a steady current is reached. The experiment is then repeated for different values of testing membrane voltages.

The measurements are repeated while blocking different type of ionic currents which allows to determine a steady-state value and time constant for each type of gate. The experimental results allow to estimate the transition rates, using the initial value determined from the equation (1.6) by setting $\mathrm{d}n/\mathrm{d}t = 0$ so we get the the initial value of open probability as

$$n_0 = \frac{\alpha_{n0}}{\alpha_{n0} + \beta_{n0}}. \tag{1.7}$$

When the testing potential is applied at $t = 0$, the values of $\alpha_n(\mathrm{V_m})$ and $\beta_n(\mathrm{V_m})$ correspond change to the value appropriate for the testing voltage $\mathrm{V_{m test}}$. The solution of (1.6) satisfies the boundary condition that $n = n_0$ when $t = 0$ is

$$n = n_\infty - (n_\infty - n_0) \exp\left(-\frac{t}{\tau_n}\right). \tag{1.8}$$

The n_∞ is a steady state solution for the testing voltage obtained analogically to (1.7). The variable τ_n is the time constant

$$\tau_n = \frac{1}{\alpha_n + \beta_n}$$

To fit the experimental results with the theoretical equation (1.8) the equation (1.5) can be written in the form

$$g_\mathrm{K} = \left[(g_{\mathrm{K}\infty})^{\frac{1}{4}} - \left((g_{\mathrm{K}\infty})^{\frac{1}{4}} - (g_{\mathrm{K}0})^{\frac{1}{4}}\right) \exp\left(-\frac{t}{\tau_n}\right)\right]^4, \tag{1.9}$$

where $g_{\mathrm{K}0}$ corresponds to the initial value of conductance and $g_{\mathrm{K}\infty}$ and steady-state conductance. Then the τ_n is chosen in a way that it fits the experimentally measured g_K. Finally, the transition rates are found as

$$\alpha_n = \frac{n_\infty}{\tau_n}, \tag{1.10a}$$

$$\beta_n = \frac{1 - n_\infty}{\tau_n}. \tag{1.10b}$$

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

$$\alpha_n = \frac{0.01(-\mathrm{V_m} + 10)}{\exp\left(\frac{-\mathrm{V_m}+10}{10} - 1\right)}, \tag{1.11a}$$

$$\beta_n = 0.125 \exp\left(\frac{-\mathrm{V_m}}{80}\right). \tag{1.11b}$$

The situation is a bit more complicated in the case of the sodium current (1.3b). The conductance for sodium is described by the equation

$$g_{\mathrm{Na}} = \bar{g_{\mathrm{Na}}} m^3 h, \tag{1.12}$$

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

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(1-m) - \beta_m m, \tag{1.13a}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(1-h) - \beta_h h, \tag{1.13b}$$

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

The solution of this equations which satisfy the boundary conditions at $m = m_0$ and $h = h_0$ at $t = 0$ are:

$$m = m_\infty - (m_\infty - m_0) \exp\left(-\frac{t}{\tau_m}\right), \tag{1.14a}$$

$$h = h_\infty - (h_\infty - h_0) \exp\left(-\frac{t}{\tau_h}\right), \tag{1.14b}$$

where the initial state and time constants are given as

$$m_0 = \frac{\alpha_{m0}}{\alpha_{m0} + \beta_{m0}}, \tag{1.15a}$$

$$\tau_m = \frac{1}{\alpha_m + \beta_m}, \tag{1.15b}$$

$$h_0 = \frac{\alpha_{h0}}{\alpha_{h0} + \beta_{h0}}, \tag{1.15c}$$

$$\tau_h = \frac{1}{\alpha_h + \beta_h}. \tag{1.15d}$$

Unlike the potassium model there are two different type of gates, which need to be fitted at the same time. However, in this model there are two kind of gates with different behaviour. The link to the experimental data is done using the assumption that the sodium conductance of activation gate is very small for values under $V_m = 30$ mV and so the initial state m_0 can be neglected. For values above the $V_m = -30$ mV the inactivation gate is fully inactivated and so the h_∞ is ignored then the (1.14) get the form

$$m = m_\infty \left[1 - \exp\left(-\frac{t}{\tau_m}\right)\right], \tag{1.16a}$$

$$h = h_0 \exp\left(-\frac{t}{\tau_h}\right). \tag{1.16b}$$

Using this approximation, the link with the experimental results is done substitution into the equation (1.12) as

$$g_{\mathrm{Na}} = \bar{g_{\mathrm{Na}}} m_\infty^3 h_0 \left[1 - \exp\left(-\frac{t}{\tau_m}\right)\right]^3 \exp\left(-\frac{t}{\tau_h}\right) \tag{1.17}$$

and fitting the time constants to the voltage-clamp traces.

Hogking-Huxley found the values of the transition rates from the voltage clamp experiments as

$$\alpha_m = \frac{0.1(-V_m + 25)}{\exp\left(\frac{-V_m+25}{10} - 1\right)}, \tag{1.18}$$

$$\beta_m = 4\exp\left(\frac{-V_m}{18}\right), \tag{1.19}$$

$$\alpha_h = 0.07\exp\left(\frac{-V_m}{20}\right), \tag{1.20}$$

$$\beta_h = \frac{1}{\exp\left(\frac{-V_m+30}{10} + 1\right)}. \tag{1.21}$$

1.1.3 Summary

Table 1.1.3 show the summary of equations describing giant squid axon in the Hodgkin-Huxley model. The implementation of this model C is listed in the appendix B.

The reproduction of the action potential and ionic currents using the Hodgkin-Huxley squid axon model is shown on the Figure 1.2. The first panel shows the membrane voltage and sodium and potassium current (leakage current not shown). The following panel shows the activation gate m and inactivation gate h during the same action potential and the corresponding current I_{Na} and the last panel shows the activation gate n and the corresponding current I_{K}.

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 V_m above the threshold value. When this happens the activation gate m

Table 1.1: Dynamical states of the Hodgkin and Huxley model

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

Table 1.2: Definitions of currents.

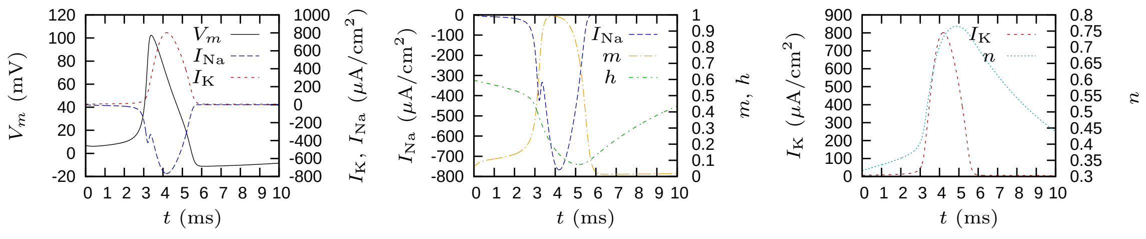
variable	description	units	definition
I_{Na}	sodium current	μA/cm^2	$I_{\mathrm{Na}} = \bar{g}_{\mathrm{Na}} m^3 h(\mathrm{V_m} - E_{\mathrm{Na}})$
I_{K}	potassium current	μA/cm^2	$I_{\mathrm{K}} = \bar{g}_{\mathrm{K}} n^4(\mathrm{V_m} - E_{\mathrm{K}})$
I_l	leakage current	μA/cm^2	$I_l = g_l(\mathrm{V_m}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

Table 1.3: Opening (α's) and closing (β's) 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 1.4: 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



(a) (b) (c)

Figure 1.2: Hodgkin and Huxley action potential and currents (a), I_{Na} current and gates m, h (b), I_{K} currents and n-gate (c).

opens and the influx of sodium ions further increases the membrane voltage. At the same time, the increase in the membrane voltage causes closing of inactivation gate h, which was initially open, which stops the sodium influx. At this point the membrane voltage is high. The high value of membrane voltage activates the gate n and the potassium current exiting the cell restores the resting potential.

1.2 Cardiac Excitation Models

The Hodgkin and Huxley model was introduced as a first electrophysiological model. The same ideas the are to a certain degree repeated and improved in the modern biophysically detailed models of cardiac cells. Those model can contain a large number of dynamical equations even over one hundred like[5]. In this subsection we describe the main differences between modern cardiac models with the Hodgkin-Huxley model.

Starting from anatomical view, the Hodgkin-Huxley model has only one intracellular compartment with a fixed value electrochemical equilibrium potential for each type of cell. This assumes that the value of intracellular ionic concentration is fixed. The modern models introduce further dynamical equations for the concentrations of Na^+, K^+, Ca^{2+} which are based on the laws for the conservation of mass between the compartments. Using the values of intra and extracellular concentrations, the electrochemical equilibrium is determined from the Nerst equation. For instance the electrochemical equilibrium potential of potassium is given as

$$E_{\mathrm{K}} = \frac{RT}{F} \ln\left(\frac{[\mathrm{K}^+]_o}{[\mathrm{K}^+]_i}\right) \tag{1.22}$$

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 intra-cellular and extra-cellular concentration.

Modern model use 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, one of the most interesting one is the sarcoplasmic reticulum which contributes to the calcium dynamics. The sarcoplasmic reticulum acts as a store of the calcium ions. The release of the calcium from the sarcoplasmic reticulum into the intracellular environment causes biochemical processes that trigger the mechanical contraction of the cell. The Figure 1.3 shows a simplified diagram of calcium dynamics.

However, it was discovered that the conductance is based on the permeability of large proteins incorporated in the cellular membrane[6] which can change their conformation to allow the ions pass through. The ionic channels which determine the conductance of the cell are divided into different groups for each particular type of ions and a new type of channels were added to the model. Those

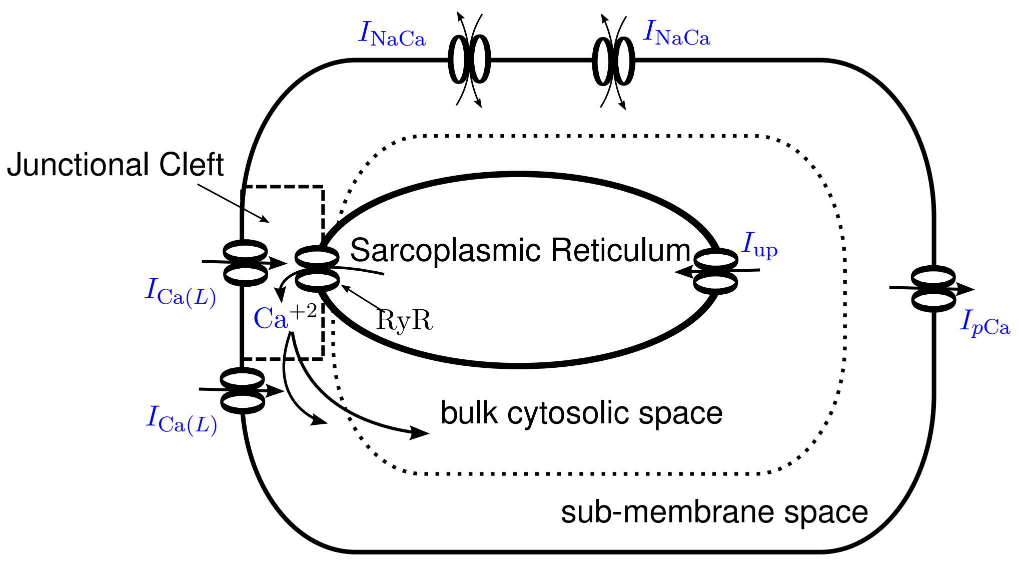


Figure 1.3: Diagram of a cardiac cell that illustrates the calcium dynamics.

are mechanisms which create the concentration and potential difference so it is “charged” for the excitation. For instance sodium-potassium pumps use the chemical energy to pump the sodium outside of the cell and potassium inside. Other example is the sodium-calcium exchanger that use allows three ions of sodium to flow down its gradient to remove the one calcium ion from the inside of the cell.

As mentioned before, the description of membrane currents are not based on first principles, but only aim to reproduce the experimental data. The gate model assumes the independence of the conformation changes of all its gates, however, this assumption is not always reasonable, because the change of conformation in one part of the channel molecule influences the rest of the molecule.

The following part describes ion channel models as the traditional Hodgkin-Huxley gate model and a Markov chain model.

1.3 Ion Channel Models

1.3.1 Exposing the Limitations of Gate Model

The sodium current I_{Na} is responsible for the initiation of action potential. The model used by Hodgkin and Huxley contain four activation gates m and one inactivation gate h which are assumed to be independent (as described in subsection 1.1.2). However, Armstrong and Bezanilla (1977)[7, 8] have found that both activation and inactivation is delayed after a hyperpolarization.

In their patch-clamp experiments they used two-pulse patch clamp protocol (Figure 1.4 inset) in which the cells were held on the potential of -80 mV before the conditioning pulse (CP) was applied for a variable duration between 0.2 and

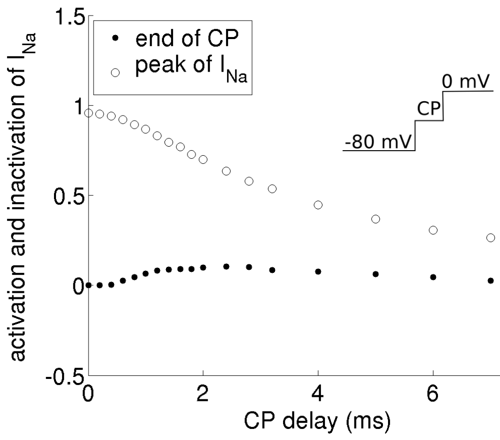


Figure 1.4: Experimental data of I_{Na} channel activation and inactivation (data scanned from article[7]) as a function of conditioning pulse (CP) duration. The conditioning pulse (CP) duration range between 0.2 and 8 ms. In the case shown in the figure the CP potential is -35 mV. Dots show the I_{Na} current at the end of the CP. Circles show the peak I_{Na} current.

8 ms (horizontal axis). The current at the end of the conditioning pulse (shown with dots) approximates the fraction of activated channels m.

After the conditioning pulse finished a second pulse to a potential 0 mV was applied. The peak current of I_{Na} normalised to the peak current without the conditioning pulse (shown with circles) corresponds to the proportion of channels able to activate $(1-h)$ where h describes the inactivation.

According to the formulation of the gate model the inactivation should follow an exponential function, but as implied by the experimental results, the shape of inactivation has sigmoid shape. Further, the inactivation is faster when the activation is well developed. This confirms that the inactivation occurs more likely when the channel is open than when closed which contradicts the main assumption of the gate model, where the gates are assumed to be independent.

Hence, an alternative approaches to address the limitations of gate independence were suggested using Markov chain models.

1.3.2 Conversion between Markov Chain and Gate Models

The ionic current is depends on the conductance of a particular group of ionic channels which is determined from the ratio between open and closed channels of that group as

$$g_s = \bar{g}_s P_{\mathrm{open}}(t) \tag{1.23}$$

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

$$P_{\mathrm{open}}(t) = \prod_{i=1}^{N} s_i(t) \tag{1.24}$$

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

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \alpha_s(1-s) - \beta_s s \tag{1.25}$$

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

An alternative approach Markov chain model is a generic solution for a systems which undergoes transitions from one state to another in a chain-like manner. 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 $P_{\mathrm{open}} = s$. The model assumes that the channel is closed, when is not open, so $P_{\mathrm{closed}} = 1 - s$. Then the transition between the states can be expressed in simple a diagram

$$(1-s) \underset{\beta_s}{\overset{\alpha_s}{\rightleftharpoons}} s. \tag{1.26}$$

Considering s and $(1-s)$ as two states of Markov chain model say conductive (open) state $O = s$ and non-conductive (closed) state $C = 1 - s$ then the diagram reads as

$$C \underset{\beta_s}{\overset{\alpha_s}{\rightleftharpoons}} O \tag{1.27}$$

which can be described by the system

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_s C - \beta_s O, \tag{1.28}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = \beta_s O - \alpha_s C \tag{1.29}$$

which are obtained by replacing the open and closed state in the equation (1.25).

The conversion between the Markov chain to gate model can be done only in special occasions, when the Markov chain has a special symmetry. An example of such Markov chain is the model shown on Figure 1.5 with four states and identical transitions in the horizontal and vertical directions on the diagram.

Here we demonstrate the conversion from this model into the gate model. The probabilities that the model resides in closed C, inactivated I, closed inactivated

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

with β_f on the downward arrows from IC to C and from I to O.

Figure 1.5: 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.

IC and open O states are computed using the system of equations

$$\frac{\mathrm{d}C}{\mathrm{d}t} = \beta_f \mathrm{IC} + \beta_d O - (\alpha_d + \alpha_f)C, \tag{1.30a}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \beta_f I + \alpha_d C - (\alpha_f + \beta_d)O, \tag{1.30b}$$

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \alpha_d \mathrm{IC} + \alpha_f O - (\beta_d + \beta_f)I, \tag{1.30c}$$

$$\frac{\mathrm{d}\,\mathrm{IC}}{\mathrm{d}t} = \beta_d I + \alpha_f C - (\alpha_d + \beta_f)\mathrm{IC} \tag{1.30d}$$

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

The equivalent gate model is obtained by assuming an 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 this new variables we can add (1.30a) to (1.30b) to get

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \alpha_f(1 - f) - \beta_f f \tag{1.31}$$

and adding (1.30b) to (1.30c) we obtain

$$\frac{\mathrm{d}d}{\mathrm{d}t} = \alpha_d(1 - d) - \beta_d d \tag{1.32}$$

which has a familiar form of a gate model. In particular the it describes an ionic channel with one activation gate d and one inactivation f which corresponds to slow inward current I_s[9].

The Markov chain is more generic as it allows to describe behaviour which would not be possible to model in a gate model. For instance the transition rates between the states IC and C on Figure 1.5 could be replaced by different values and such model could not be reduced into a gate model. However, this comes to the expense of higher computational cost due to larger number of dynamical variables.

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

$$\alpha = \alpha_0 \exp\left(z_\alpha \frac{\mathrm{V_m} F}{RT}\right), \tag{1.33}$$

$$\beta = \beta_0 \exp\left(-z_\beta \frac{\mathrm{V_m} F}{RT}\right) \tag{1.34}$$

where α_0 and β_0 (ms) are the transition rates at membrane potential $\mathrm{V_m} = 0$ 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.

In general, the Markov chain model is described in terms of a linear system of differential equations

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}(t)\vec{u} \tag{1.35}$$

where $\vec{u}$ represents the states of the Markov chain and $\boldsymbol{A}$ is matrix filled with the corresponding transition rates. The open probability corresponds to the sum of all open channels.

For instance, the (1.30) can be expressed as

$$\boldsymbol{A} = \begin{bmatrix} -(\alpha_d + \alpha_f) & \beta_d & 0 & \beta_f \\ \alpha_d & -(\alpha_f + \beta_d) & \beta_f & 0 \\ 0 & \alpha_f & -(\beta_d + \beta_f) & \alpha_d \\ \alpha_f & 0 & \beta_d & -(\alpha_d + \beta_f) \end{bmatrix}, \quad \vec{u} = \begin{bmatrix} C \\ O \\ I \\ \mathrm{IC} \end{bmatrix}. \tag{1.36}$$

The sum of the rates in the columns of the transition matrix $\boldsymbol{A}$ is zero, which implies the state conservation, i.e. the sum of the states is constant (equal to one). This also means, that at least one of the eigenvalues of the matrix is zero.

1.3.3 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 equivalent Markov chain. A combination of states for each channel correspond to one state of resulting Markov chain model.

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

One possible approach to convert this model to a Markov chain model is the following. If the system depends on the order in which the particular gates close, then each channel state is one of possible *variations* of the two gate states (e.g. channel state $n, n, n, (1-n)$ is different from $(1-n), n, n, n$). So there is be $2^4 = 16$ channel states. Each channel states is connected with vortexes with other four channel states in a four-dimensional hypercube. The transition rates at each connection correspond 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 need not 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 *combinations* of gate states. So the number of channel states is 5 and the Markov chain can be visualised in a simple linear structure as

$$C_4 \underset{\beta_n}{\overset{4\alpha_n}{\rightleftharpoons}} C_3 \underset{2\beta_n}{\overset{3\alpha_n}{\rightleftharpoons}} C_2 \underset{3\beta_n}{\overset{2\alpha_n}{\rightleftharpoons}} C_1 \underset{4\beta_n}{\overset{\alpha_n}{\rightleftharpoons}} O_{\mathrm{K}} \tag{1.37}$$

where $O_{\mathrm{K}} = C_0$ and C_i is a channel state corresponding to i gates in closed $(1-n)$ gate state.

Unlike in the "variation" case (where the order of variable 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 so the channel resides in C_4, the transition α_n is multiplied by 4 as it is four times more likely, that any of the four gates opens.

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

The initial conditions of the channel states are determined from the initial conditions n_0 of gates. This initial condition of the gate is multiplied by coefficients found from binomial expansion. The cases when all the channel 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 ($C_3(t_0) = 4(1-n_0)^4 n_0$, and $C_1(t_0) = 4n_0^3(1-n_0)$ respectively). The state with two open and two closed gates is found as $C_2 = 6n^2(1-n_0)^2$.

The open probability of I_{Na} model depends on three activation gate m and 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 similar Markov chain to (1.37). In the case of I_{Na} the activation part has only four states

$$C_{3/2\mathrm{Na}} \underset{\beta_m}{\overset{3\alpha_m}{\rightleftharpoons}} C_{2/2\mathrm{Na}} \underset{2\beta_m}{\overset{2\alpha_m}{\rightleftharpoons}} C_{1/2\mathrm{Na}} \underset{3\beta_m}{\overset{\alpha_m}{\rightleftharpoons}} O_{1/2\mathrm{Na}} \tag{1.38}$$

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

Figure 1.6: Diagram of I_{Na} Markov chain model [1]

where $C_{i/2\mathrm{Na}}$ corresponds to state with i gates closed and $C_{0/2\mathrm{Na}} = O_{1/2\mathrm{Na}}$. Transition rates α_m, β_m are related to the opening and closing of the m gates. For more details on the derivation of the model, refer to description of I_{K} Markov chain above.

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

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

where $C_{i\mathrm{Na}}$ states correspond to the states of (1.38) with open inactivated gate h and states $\mathrm{IC}_{i\mathrm{Na}}$ correspond to the same with closed inactivation gate h.

1.4 Popular Markov Chain Ion Channel Models

1.4.1 I_{Na}

Figure 1.6 shows Clancy and Rudy (2002) I_{Na} model. This model improves previously published model by the same authors (1999), and was widely cited in other literature (131 and 220 respectively by March 2013). 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 system is described by the following ODEs:

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{13}C_1 + \beta_2\mathrm{IF} - (\beta_{13} + \alpha_2)O, \tag{1.39a}$$

$$\frac{\mathrm{d}C_1}{\mathrm{d}t} = \alpha_{12}C_2 + \beta_{13}O + \alpha_3\mathrm{IF} - (\beta_{12} + \beta_3 + \alpha_{13})C_1, \tag{1.39b}$$

$$\frac{\mathrm{d}C_2}{\mathrm{d}t} = \alpha_{11}C_3 + \alpha_3\mathrm{IC}_2 + \beta_{12}C_1 - (\beta_{11} + \alpha_{12} + \beta_3)C_2, \tag{1.39c}$$

$$\frac{dC_3}{dt} = \beta_{11}C_2 + \alpha_3 \mathrm{IC}_3 - (\alpha_{11} + \beta_3)C_3, \tag{1.39d}$$

$$\frac{d\,\mathrm{IC}_3}{dt} = \beta_3 C_3 + \beta_{11}\mathrm{IC}_2 - (\alpha_{11} + \alpha_3)\mathrm{IC}_3, \tag{1.39e}$$

$$\frac{d\,\mathrm{IC}_2}{dt} = \alpha_{11}\mathrm{IC}_3 + \beta_3 C_2 + \beta_{12}\mathrm{IF} - (\beta_{11} + \alpha_3 + \alpha_{12})\mathrm{IC}_2, \tag{1.39f}$$

$$\frac{d\,\mathrm{IF}}{dt} = \alpha_{12}\mathrm{IC}_2 + \beta_4\mathrm{IM}_1 + \beta_3 C_1 + \alpha_2 O - (\beta_{12} + \alpha_4 + \beta_2 + \alpha_3)\mathrm{IF}, \tag{1.39g}$$

$$\frac{d\,\mathrm{IM}_1}{dt} = \alpha_4\mathrm{IF} + \beta_5\mathrm{IM}_2 - (\beta_4 + \alpha_5)\mathrm{IM}_1, \tag{1.39h}$$

$$\frac{d\,\mathrm{IM}_2}{dt} = \alpha_5\mathrm{IM}_1 - \beta_5\mathrm{IM}_2 \tag{1.39i}$$

where the transition rates are

$$C_3 \to C_2 \qquad \alpha_{11} = \frac{3.802}{0.1027e^{-\mathrm{V_m}/17.0} + 0.20\exp\left(-\mathrm{V_m}/150\right)}, \tag{1.40a}$$

$$C_2 \to C_1 \qquad \alpha_{12} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/15.0\right) + 0.23\exp\left(-\mathrm{V_m}/150\right)}, \tag{1.40b}$$

$$C_1 \to O \qquad \alpha_{13} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/12.0\right) + 0.25\exp\left(-\mathrm{V_m}/150\right)}, \tag{1.40c}$$

$$\mathrm{IC}_3 \to \mathrm{IC}_2 \qquad \alpha_{11} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/17.0\right) + 0.20\exp\left(-\mathrm{V_m}/150\right)}, \tag{1.40d}$$

$$\mathrm{IC}_2 \to \mathrm{IF} \qquad \alpha_{12} = \frac{3.802}{0.1027\exp\left(-\mathrm{V_m}/15.0\right) + 0.23\exp\left(-\mathrm{V_m}/150\right)}, \tag{1.40e}$$

$$C_2 \to C_3 \qquad \beta_{11} = 0.1917\exp\left(-\mathrm{V_m}/20.3\right), \tag{1.40f}$$

$$C_1 \to C_2 \qquad \beta_{12} = 0.20\exp\left(-(\mathrm{V_m} - 5)/20.3\right), \tag{1.40g}$$

$$O \to C_1 \qquad \beta_{13} = 0.22\exp\left(-(\mathrm{V_m} - 10)/20.3\right), \tag{1.40h}$$

$$C_2 \to C_3 \qquad \beta_{11} = 0.1917\exp\left(-\mathrm{V_m}/20.3\right), \tag{1.40i}$$

$$C_1 \to C_2 \qquad \beta_{12} = 0.20\exp\left(-(\mathrm{V_m} - 5)/20.3\right), \tag{1.40j}$$

$$\mathrm{IF} \to C_1 \qquad \alpha_3 = 3.7933 \cdot 10^{-7}\exp\left(-\mathrm{V_m}/7.7\right), \tag{1.40k}$$

$$\mathrm{IC}_2 \to C_2 \qquad \alpha_3 = 3.7933 \cdot 10^{-7}\exp\left(-\mathrm{V_m}/7.7\right), \tag{1.40l}$$

$$\mathrm{IC}_3 \to C_3 \qquad \alpha_3 = 3.7933 \cdot 10^{-7}\exp\left(-\mathrm{V_m}/7.7\right), \tag{1.40m}$$

$$C_1 \to \mathrm{IF} \qquad \beta_3 = 8.4 \cdot 10^{-3} + 2 \cdot 10^{-5}\mathrm{V_m}, \tag{1.40n}$$

$$C_2 \to \mathrm{IC}_2 \qquad \beta_3 = 8.4 \cdot 10^{-3} + 2 \cdot 10^{-5}\mathrm{V_m}, \tag{1.40o}$$

$$C_3 \to \mathrm{IC}_3 \qquad \beta_3 = 8.4 \cdot 10^{-3} + 2 \cdot 10^{-5}\mathrm{V_m}, \tag{1.40p}$$

$$O \to \mathrm{IF} \qquad \alpha_2 = 9.178 \exp\left(\mathrm{V_m}/29.68\right), \tag{1.40q}$$

$$\mathrm{IF} \to O \qquad \beta_2 = \frac{\alpha_{13}\alpha_2\alpha_3}{\beta_{13}\beta_3}, \tag{1.40r}$$

$$\mathrm{IF} \to \mathrm{IM}_1 \qquad \alpha_4 = \alpha_2/100, \tag{1.40s}$$

$$\mathrm{IM}_1 \to \mathrm{IF} \qquad \beta_4 = \alpha_3, \tag{1.40t}$$

$$\mathrm{IM}_1 \to \mathrm{IM}_2 \qquad \alpha_5 = \alpha_2/(9.5 \cdot 10^4), \tag{1.40u}$$

$$\mathrm{IM}_2 \to \mathrm{IM}_1 \qquad \beta_5 = \alpha_3/50. \tag{1.40v}$$

1.4.2 $I_{\mathrm{Ca}(L)}$

Calcium current $I_{\mathrm{Ca}(L)}$ plays a crucial role in forming the AP morphology and duration. Its importance is enhanced by providing negative feedback to control intra-cellular Ca^{2+} concentration[10]. The channel is opened by 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 observed in the presence of calcium ions Ca^{2+}. Calcium binding does not alter movement of the voltage sensor, however no current can flow through the channel. The experimental discovery of this inactivation was done using barium ions Ba^{2+}, which otherwise behave similarly to calcium, however, as they do not bind to the channel, the inactivation is not observed as in the case of calcium.

Markovian model including calcium-dependent inactivation was proposed by Imredy and Yue [11]. This model was improved to reflect the molecular structure by Jafri et al.[12](180 citation). They define normal mode, and Ca mode when the channel is inactivated by Ca^{+2}. Each mode contains five closed states with possible Ca^{+2} dependent transitions between modes; and one open state. The open transition rate in Ca mode was assumed to be very slow.

In the model proposed by Faber et al.[2] (232 citations) substitute open state by inactivated in Ca mode (Figure 1.7). According to the experimental results [13] the $I_{\mathrm{Ca}(L)}$ contains to additional inactivation with fast and slow kinetics.

The $I_{\mathrm{Ca}(L)}$ can operate in two regimes: the lower tier corresponds to state of the channel under Ca^{2+} dependent inactivation, the upper tier corresponds to states without Ca^{2+} inactivation, The only conductive state is the state O. The states of the model are described by the following system of ODEs

$$\frac{\mathrm{d}C_0}{\mathrm{d}t} = \theta C_{0\mathrm{Ca}} + \beta_0 C_1 - (\delta + \alpha_0)C_0, \tag{1.41a}$$

$$\frac{\mathrm{d}C_1}{\mathrm{d}t} = \theta C_{1\mathrm{Ca}} + \beta_1 C_2 + \alpha_0 C_0 - (\delta + \beta_0 + \alpha_1)C_1, \tag{1.41b}$$

$$\frac{\mathrm{d}C_2}{\mathrm{d}t} = \theta C_{2\mathrm{Ca}} + \beta_2 C_3 + \alpha_1 C_1 - (\delta + \beta_1 + \alpha_2)C_2, \tag{1.41c}$$

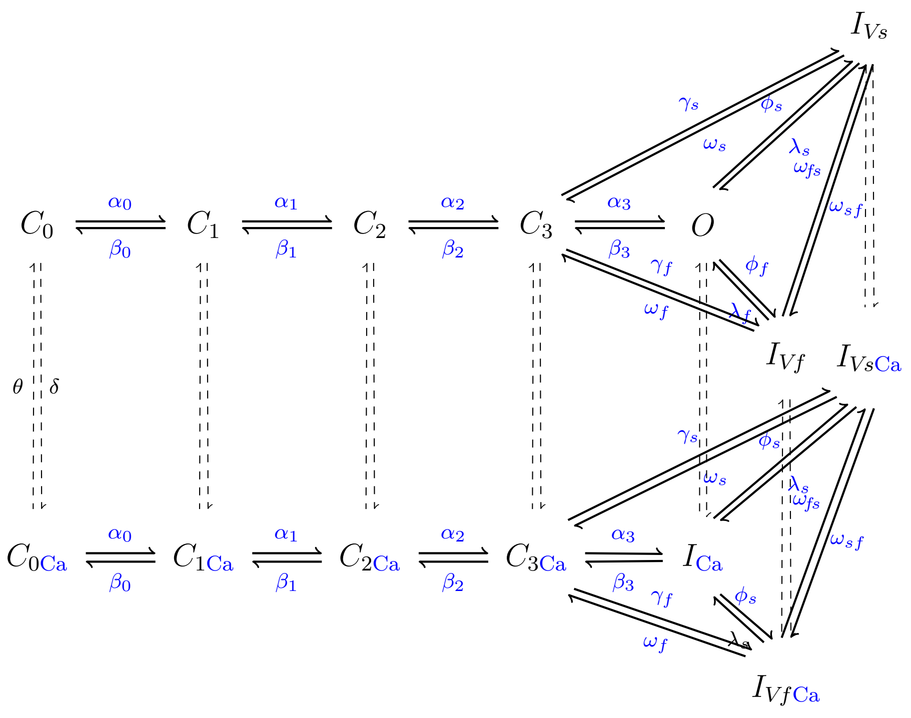


Figure 1.7: Markov chain model of $I_{\mathrm{Ca}(L)}$ [2]

$$\frac{\mathrm{d}C_3}{\mathrm{d}t} = \theta C_{3\mathrm{Ca}} + \beta_3 O + \alpha_2 C_2 + \omega_f I_{Vf} + \omega_s I_{Vs} - (\delta + \beta_2 + \alpha_3 + \gamma_f + \gamma_s) C_3, \tag{1.41d}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \theta I_{\mathrm{Ca}} + \alpha_3 C_3 + \lambda_f I_{Vf} + \lambda_s I_{Vs} - (\delta + \beta_3 + \phi_f + \phi_s) O, \tag{1.41e}$$

$$\frac{\mathrm{d}I_{Vf}}{\mathrm{d}t} = \gamma_f C_3 + \phi_f O + \omega_{sf} I_{Vs} + \theta I_{Vf\mathrm{Ca}} - (\omega_f + \omega_{fs} + \lambda_f + \delta) I_{Vf}, \tag{1.41f}$$

$$\frac{\mathrm{d}I_{Vs}}{\mathrm{d}t} = \gamma_s C_3 + \phi_s O + \omega_{fs} I_{Vf} + \theta I_{Vs\mathrm{Ca}} - (\omega_s + \omega_{sf} + \lambda_s + \delta) I_{Vs}, \tag{1.41g}$$

$$\frac{\mathrm{d}C_{0\mathrm{Ca}}}{\mathrm{d}t} = \delta C_0 + \beta_0 C_{1\mathrm{Ca}} - (\theta + \alpha_0) C_{0\mathrm{Ca}}, \tag{1.41h}$$

$$\frac{\mathrm{d}C_{1\mathrm{Ca}}}{\mathrm{d}t} = \delta C_1 + \beta_1 C_{2\mathrm{Ca}} + \alpha_0 C_{0\mathrm{Ca}} - (\theta + \beta_0 + \alpha_1) C_{1\mathrm{Ca}}, \tag{1.41i}$$

$$\frac{\mathrm{d}C_{2\mathrm{Ca}}}{\mathrm{d}t} = \delta C_2 + \beta_2 C_{3\mathrm{Ca}} + \alpha_1 C_{1\mathrm{Ca}} - (\theta + \beta_1 + \alpha_2) C_{2\mathrm{Ca}}, \tag{1.41j}$$

$$\frac{\mathrm{d}C_{3\mathrm{Ca}}}{\mathrm{d}t} = \delta C_3 + \beta_3 I_{\mathrm{Ca}} + \alpha_2 C_{2\mathrm{Ca}} + \omega_f I_{Vf} + \omega_s I_{Vs} - (\theta + \beta_2 + \alpha_3 + \gamma_f + \gamma_s) C_{3\mathrm{Ca}}, \tag{1.41k}$$

$$\frac{\mathrm{d}I_{\mathrm{Ca}}}{\mathrm{d}t} = \delta O + \alpha_3 C_{3\mathrm{Ca}} + \lambda_f I_{Vf\mathrm{Ca}} + \lambda_s I_{Vs\mathrm{Ca}} - (\theta + \beta_3 + \phi_f + \phi_s) I_{\mathrm{Ca}}, \tag{1.41l}$$

$$\frac{\mathrm{d}I_{Vf\mathrm{Ca}}}{\mathrm{d}t} = \gamma_f C_{3\mathrm{Ca}} + \phi_f O + \omega_{sf} I_{Vf\mathrm{Ca}} + \delta I_{Vf} - (\omega_f + \omega_{fs} + \lambda_f + \theta) I_{Vf\mathrm{Ca}}, \tag{1.41m}$$

$$\frac{\mathrm{d}I_{Vs\mathrm{Ca}}}{\mathrm{d}t} = \gamma_s C_{3\mathrm{Ca}} + \phi_s O + \omega_{fs} I_{Vf\mathrm{Ca}} + \delta I_{Vs} - (\omega_s + \omega_{sf} + \lambda_s + \theta) I_{Vs\mathrm{Ca}} \tag{1.41n}$$

where the parameters are

$$\alpha = 0.925 \exp(\mathrm{V_m}/30), \tag{1.42a}$$
$$\beta = 0.39 \exp(-\mathrm{V_m}/40), \tag{1.42b}$$
$$\alpha_0 = 4\alpha, \tag{1.42c}$$
$$\alpha_1 = 3\alpha, \tag{1.42d}$$
$$\alpha_2 = 2\alpha, \tag{1.42e}$$
$$\alpha_3 = \alpha, \tag{1.42f}$$
$$\beta_0 = \beta, \tag{1.42g}$$
$$\beta_1 = 2\beta, \tag{1.42h}$$
$$\beta_2 = 3\beta, \tag{1.42i}$$
$$\beta_3 = 4\beta, \tag{1.42j}$$
$$\gamma_f = 0.245 \exp(\mathrm{V_m}/10), \tag{1.42k}$$
$$\gamma_s = 0.005 \exp(-\mathrm{V_m}/40), \tag{1.42l}$$
$$\phi_f = 0.02 \exp(\mathrm{V_m}/500), \tag{1.42m}$$
$$\phi_s = 0.03 \exp(-\mathrm{V_m}/280), \tag{1.42n}$$
$$\lambda_f = 0.035 \exp(-\mathrm{V_m}/300), \tag{1.42o}$$
$$\lambda_s = 0.0011 \exp(V/500), \tag{1.42p}$$
$$\omega_f = (\beta_3\lambda_f\gamma_f)/(\alpha_3\phi_f), \tag{1.42q}$$
$$\omega_s = (\beta_3\lambda_s\gamma_s)/(\alpha_3\phi_s), \tag{1.42r}$$
$$\omega_{sf} = (\lambda_s\phi_f)/\lambda_f, \tag{1.42s}$$
$$\omega_{fs} = \phi_s, \tag{1.42t}$$
$$\delta = \frac{1}{1 + 1/[\mathrm{Ca^{2+}}]_{\mathrm{ss}}}, \tag{1.42u}$$
$$\theta = 0.01. \tag{1.42v}$$

1.4.3 RyR

The calcium dynamics in the muscle cell controls mechanical activity of the cell. The cell is divided into four compartments (Figure 1.3): junctional cleft – disk space by the cellular membrane and sarcoplasmic reticulum; sub-membrane space – narrow compartment close to the cellular membrane; sarcoplasmic reticulum – intra-cellular organelle; bulk cytosolic space – remaining intra-cellular space.

Sarcoplasmic reticulum acts as a storage of Ca^{2+} ions inside of the cell. Ryanodine receptor (RyR) is an ion channel presented on a membrane sarcoplasmic reticulum. The current of calcium from the sarcoplasmic reticulum into intra-cellular space is triggered by increasing calcium concentration in the junctional cleft or inside the sarcoplasmic reticulum. The release of calcium due to increased con-

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

Figure 1.8: State diagram of RyR Markov chain model

centration in junctional cleft is known as calcium-induced calcium release (CICR). If the intra sarcoplasmic calcium concentration $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ increases over threshold, the calcium is released. This mechanism may cause arrhythmogenic phenomena called early after depolarisation.

The Markov chain model of RyR was proposed by several groups [14, 15, 16, 17, 18, 19]. Stern et al.[18] tested the available models in cellular environment and developed a most appropriate model reproducing the mechanisms inside of the cell.

Figure 1.8 shows the Markov chain diagram of the RyR. 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 of $\mu\mathrm{A}/\mu\mathrm{F}$.

The RyR model is described by the following system of ordinary differential equations (ODEs)

$$\frac{\mathrm{d}C_1}{\mathrm{d}t} = \beta_{1R}C_2 + \delta_{1R}I_1 - (\alpha_{1R} + \gamma_{1R})C_1, \tag{1.43a}$$

$$\frac{\mathrm{d}C_2}{\mathrm{d}t} = \alpha_{1R}C_1 + \beta_{2R}C_3 + \delta_{2R}I_2 - (\beta_{1R} + \alpha_{2R} + \gamma_{2R})C_2, \tag{1.43b}$$

$$\frac{\mathrm{d}C_3}{\mathrm{d}t} = \alpha_{2R}C_2 + \beta_{3R}C_4 + \delta_{3R}I_3 - (\beta_{2R} + \alpha_{3R} + \gamma_{3R})C_3, \tag{1.43c}$$

$$\frac{\mathrm{d}C_4}{\mathrm{d}t} = \beta_{4R}O + \alpha_{3R}C_3 + \delta_{4R}I_4 - (\beta_{3R} + \alpha_{4R} + \gamma_{4R})C_4, \tag{1.43d}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{4R}C_4 + \delta_{5R}I_5 - (\gamma_{5R} + \beta_{4R})O, \tag{1.43e}$$

$$\frac{\mathrm{d}I_1}{\mathrm{d}t} = \gamma_{1R}C_1 + \beta_{1R}I_2 - (\delta_{1R} + \alpha_{1R})I_1, \tag{1.43f}$$

$$\frac{\mathrm{d}I_2}{\mathrm{d}t} = \gamma_{2R}C_2 + \alpha_{1R}I_1 + \beta_{2R}I_3 - (\delta_{2R} + \beta_{1R} + \alpha_{2R})I_2, \tag{1.43g}$$

$$\frac{\mathrm{d}I_3}{\mathrm{d}t} = \gamma_{3R}C_3 + \alpha_{2R}I_2 + \beta_{3R}I_4 - (\delta_{3R} + \beta_{2R} + \alpha_{3R})I_3, \tag{1.43h}$$

$$\frac{\mathrm{d}I_4}{\mathrm{d}t} = \gamma_{4R}C_4 + \alpha_{3R}I_3 + \beta_{4R}I_5 - (\delta_{4R} + \beta_{3R} + \alpha_{4R})I_4, \tag{1.43i}$$

$$\frac{\mathrm{d}I_5}{\mathrm{d}t} = \gamma_{5R}O + \alpha_{4R}I_4 - (\delta_{5R} + \beta_{4R})I_5 \tag{1.43j}$$

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

$$\alpha_{1R} = 1750[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44a}$$

$$\alpha_{2R} = 5600[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44b}$$

$$\alpha_{3R} = 5600[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44c}$$

$$\alpha_{4R} = 5600[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44d}$$

$$\beta_{1R} = 5, \tag{1.44e}$$

$$\beta_{2R} = 2.62, \tag{1.44f}$$

$$\beta_{3R} = 1, \tag{1.44g}$$

$$\beta_{4R} = 6.25, \tag{1.44h}$$

$$\gamma_{1R} = 0.4[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44i}$$

$$\gamma_{2R} = 1.2[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44j}$$

$$\gamma_{3R} = 2.8[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44k}$$

$$\gamma_{4R} = 5.2[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44l}$$

$$\gamma_{5R} = 8.4[\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \tag{1.44m}$$

$$\delta_{1R} = \frac{0.01}{1 + \left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{1.44n}$$

$$\delta_{2R} = \frac{0.001}{1 + \left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{1.44o}$$

$$\delta_{3R} = \frac{0.0001}{1 + \left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{1.44p}$$

$$\delta_{4R} = \frac{0.00001}{1 + \left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9}, \tag{1.44q}$$

$$\delta_{5R} = \frac{0.000001}{1 + \left(\frac{0.75\overline{\mathrm{CSQN}}}{\mathrm{CSQN}}\right)^9} \tag{1.44r}$$

where the constant $\overline{\mathrm{CSQN}} = 10\ \mathrm{mM}$ and $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$ and CSQN are dynamical variable in the cell model.

Chapter 2

Asymptotic and Numerical Methods

2.1 Perturbation Theory

2.1.1 Leading Order Reduction for Linear Systems

Definition of the System

The dimensionality of the Markov chain corresponds to the number of combinations of the gates in a corresponding model, which results in several more dynamical variables than than in the identical gate-type counterparts. The computation of such a large system takes longer than the computation of identical gate model. The resulting model can contain a combination of phenomena observed in fast and slow time scale. Using Tikhonov approach[20] the system of equations can be approximated by a slow system with lower number of equations.

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

$$\frac{\mathrm{d}\vec{X}}{\mathrm{d}t} = \boldsymbol{A}(t,\varepsilon)\vec{X}(t) + \vec{H}(t), \quad \vec{X}(t), \vec{H}(t) \in \mathbb{R}^n, \boldsymbol{A}(t,\varepsilon) \in \mathbb{R}^{n\times n}. \tag{2.1}$$

Division to Time Scales

To analyse the fast behaviour of this system in vicinity of a time point t_0 we introduce a 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 $\vec{Y}(t, \tau) \in \mathrm{R}^n$.

The system (2.1) is reformulated in terms of parameter ε and using the chain derivative rule as

$$\varepsilon\left(\frac{\partial\vec{Y}}{\partial t}+\frac{1}{\varepsilon}\frac{\partial\vec{Y}}{\partial t}\right)=\varepsilon\boldsymbol{A}(t,\varepsilon)\vec{Y}(t,\tau)+\varepsilon\vec{H}(t) \tag{2.2}$$

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

$$\boldsymbol{A}(t,\varepsilon)=\sum_{\ell}\varepsilon^{\ell-1}\boldsymbol{A}_{\ell}(t)=\frac{1}{\varepsilon}\left(\boldsymbol{A}_0(t)+\varepsilon\boldsymbol{A}_1(t)+\varepsilon^2\boldsymbol{A}_2(t)+\dots\right), \tag{2.3a}$$

$$\vec{Y}(t,\tau)=\sum_{\ell}\varepsilon^{\ell}\vec{Y}_{\ell}(t,\tau)=\vec{Y}_0(t,\tau)+\varepsilon\vec{Y}_1(t,\tau)+\varepsilon^2\vec{Y}_2(t,\tau)+\dots \tag{2.3b}$$

where each matrix $\boldsymbol{A}_{\ell}(t)$ describes the behaviour of the system in a time-scale $\varepsilon^{\ell-1}t$. After substitution of (2.3) into (2.2) we obtain

$$\begin{aligned}\varepsilon\frac{\partial\vec{Y}_0}{\partial t}+\frac{\partial\vec{Y}_0}{\partial t}+\varepsilon\frac{\partial\vec{Y}_1}{\partial t}&=\\ &=\boldsymbol{A}_0(t)\vec{Y}_0(t,\tau)+\varepsilon\boldsymbol{A}_1(t)\vec{Y}_0(t,\tau)+\varepsilon\boldsymbol{A}_0(t)\vec{Y}_1(t,\tau)+\varepsilon\vec{H}(t)+\mathcal{O}(\varepsilon^2).\end{aligned} \tag{2.4}$$

Collecting the terms according to the order of ε gives us

$$\mathcal{O}(\varepsilon^0):\quad \frac{\partial\vec{Y}_0}{\partial t}=\boldsymbol{A}_0(t)\vec{Y}_0(t,\tau), \tag{2.5a}$$

$$\mathcal{O}(\varepsilon^1):\quad \frac{\partial\vec{Y}_0}{\partial t}+\frac{\partial\vec{Y}_1}{\partial t}=\boldsymbol{A}_1(t)\vec{Y}_0(t,\tau)+\boldsymbol{A}_0(t)\vec{Y}_1(t,\tau)+\vec{H}(t), \tag{2.5b}$$

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

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 solve eigenvalue problem of the matrix $\boldsymbol{A}_0$. The eigenvectors and eigenvalues of matrix $\boldsymbol{A}_0$ have to satisfy the condition

$$\boldsymbol{A}_0(t)\vec{u}_k(t)=\lambda_k(t)\vec{u}_k(t),\qquad \boldsymbol{A}_0(t)\in\mathbb{R}^{n\times n},\vec{u}_k(t)\in\mathbb{R}^n,\lambda_k(t)\mathbb{R}. \tag{2.6}$$

A matrix $\boldsymbol{A}_0(t)\in\mathbb{R}^{n\times n}$ has n eigenvalues. The multiplicity m of eigenvalue $\lambda_k(t)$ is denoted as the number of repetition of the same eigenvalues. If some of the eigenvectors $\lambda_k(t)=0$ we denote the matrix $\boldsymbol{A}_0(t)$ degenerate.

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

$$\boldsymbol{\Lambda}(t) = \begin{bmatrix} \lambda_0(t) & 0 & \dots & 0 \\ 0 & \lambda_1(t) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n(t) \end{bmatrix} \tag{2.7}$$

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

$$\tilde{\boldsymbol{P}}(t) = [\vec{u}_0(t), \vec{u}_1(t), \dots, \vec{u}_n(t)] \tag{2.8}$$

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

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

$$\tilde{\boldsymbol{P}}^{-1}(t) = \begin{bmatrix} \vec{\omega}^{\mathrm{T}}{}_0(t) \\ \vec{\omega}^{\mathrm{T}}{}_1(t) \\ \vdots \\ \vec{\omega}^{\mathrm{T}}{}_n(t) \end{bmatrix} \tag{2.9}$$

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

$$\vec{\omega}^{\mathrm{T}}{}_j(t)\vec{u}_k(t) = \delta_{jk} \tag{2.10}$$

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

$$\vec{v}_k(t) = s_k(t)\vec{u}_k(t) \tag{2.11}$$

because any of them satisfies the relation

$$\boldsymbol{A}_0(t)s_k(t)\vec{u}_k(t) = \lambda_k(t)s_k(t)\vec{u}_k(t), \tag{2.12a}$$
$$\boldsymbol{A}_0(t)\vec{v}_k(t) = \lambda_k(t)\vec{v}_k(t). \tag{2.12b}$$

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

Further, it may be convenient to chose the eigenvector for *dynamical orthogonality* this means that that the derivative of the eigenvector is orthogonal to the corresponding dual basis as

$$\vec{w}^{\mathrm{T}}{}_k \frac{\mathrm{d}\vec{v}_k}{\mathrm{d}t} = \xi_{kk} = 0. \tag{2.13}$$

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

$$0 = \vec{w}^{\mathrm{T}}{}_k(t)\frac{\mathrm{d}\vec{v}_k}{\mathrm{d}t} = \frac{s_k(t)}{s_k(t)}\vec{\omega}^{\mathrm{T}}{}_k(t)\frac{\mathrm{d}\vec{u}_k}{\mathrm{d}t} + \frac{\delta_{kk}}{s_k(t)}\frac{\mathrm{d}s_k}{\mathrm{d}t} = \vec{\omega}^{\mathrm{T}}{}_k(t)\frac{\mathrm{d}\vec{u}_k}{\mathrm{d}t} + s_k^{-1}(t)\frac{\mathrm{d}s_k}{\mathrm{d}t}. \tag{2.14}$$

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

$$\int \frac{1}{s_k(t)}\mathrm{d}s_k(t) = -\int \vec{\omega}^{\mathrm{T}}{}_k(t)\mathrm{d}\vec{u}_k(t). \tag{2.15}$$

Then the $s_k(t)$ has a solution

$$s_k(t) = e^{-\int \vec{\omega}^{\mathrm{T}}{}_k(t)\mathrm{d}\vec{u}_k(t)}. \tag{2.16}$$

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

The equation (2.5a) gives the solution in $\mathcal{O}(\varepsilon^0)$ as

$$\vec{Y}_0(t,\tau) = \sum_k a_k(t)\vec{v}_k(t)e^{\lambda_k(t)\tau} \tag{2.17}$$

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 $\boldsymbol{A}_0(t)$.

Due to the conservation law of Markov chains (sum of all dynamical states is always 1), the system has a zero eigenvalue, to which we assign the index $j = 0$, such that $\lambda_0(t) = 0$. Further, we consider that the multiplicity of all eigenvalues $m = 1$. Because $\lambda_0(t) = 0$ and $\mathrm{Re}\{\lambda_k(t)\} < 0$, to find the solution in the slow time scale t we let the slow time $\tau \to \infty$ as

$$\lim_{\tau\to\infty} \vec{Y}_0(t,\tau) = a_0(t)\vec{v}_0(t). \tag{2.18}$$

We rewrite (2.5b) in a non-homogeneous form

$$\frac{\partial \vec{Y}_1}{\partial t} = \boldsymbol{A}_0(t)\vec{Y}_1(t,\tau) + \vec{F}(t,\tau), \tag{2.19}$$

where the term

$$\vec{F}(t,\tau) = \boldsymbol{A}_1(t)\vec{Y}_0(t,\tau) - \frac{\partial \vec{Y}_0}{\partial t} + \vec{H}(t). \tag{2.20}$$

Using the solution for $\vec{Y}_0$ given in the equation (2.17) we expand

$$\vec{F}(t,\tau) = \sum_k \left[\boldsymbol{A}_1(t) a_k(t) \vec{v}_k(t) - \frac{\mathrm{d}a_k}{\mathrm{d}t} \vec{v}_k(t) - a_k(t) \frac{\mathrm{d}\vec{v}_k}{\mathrm{d}t} - \right. \\ \left. - a_k(t) \vec{v}_k(t) \frac{\mathrm{d}\lambda_k}{\mathrm{d}t} \tau \right] e^{\lambda_k(t)\tau} + \vec{H}(t). \tag{2.21}$$

The solution of the system of equations (2.19) is obtained by using the method of diagonalisation by multiplying the system from the left by $\boldsymbol{P}^{-1}$ to get

$$\frac{\partial \vec{Z}}{\partial t} = \boldsymbol{\Lambda}(t) \vec{Z}(t,\tau) + \vec{G}(t,\tau) \tag{2.22}$$

where $\boldsymbol{\Lambda}(t)$ is eigenvector matrix as defined in (2.7) and

$$\vec{Z}(t,\tau) = \boldsymbol{P}^{-1}(t) \vec{Y}_1(t,\tau), \tag{2.23a}$$
$$\vec{G}(t,\tau) = \boldsymbol{P}^{-1}(t) \vec{F}(t,\tau), \tag{2.23b}$$

which can be written by elements as

$$z_j(t,\tau) = \vec{w}^{\mathrm{T}}{}_j(t) \vec{Y}_1(t,\tau), \tag{2.24a}$$
$$g_j(t,\tau) = \vec{w}^{\mathrm{T}}{}_j(t) \vec{F}(t,\tau). \tag{2.24b}$$

Substituting (2.21) into (2.24b) and using the orthogonality (2.10) and dynamical orthogonality (2.13) properties for eigenvectors we get

$$g_j(t,\tau) = \sum_k \left[\vec{w}^{\mathrm{T}}{}_j(t) \boldsymbol{A}_1(t) a_k(t) \vec{v}_k(t) - \xi_{jk} a_k(t) \right] e^{\lambda_k(t)\tau} - \\ - \left[\frac{\mathrm{d}a_j}{\mathrm{d}t} + a_j(t) \frac{\mathrm{d}\lambda_j}{\mathrm{d}t} \tau \right] e^{\lambda_j(t)\tau} + h_j(t) \tag{2.25}$$

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

The system (2.22) is uncoupled, and each individual equation can be written in the form

$$\frac{\partial z_j}{\partial \tau} = \lambda_j(t) z_j(t,\tau) + g_j(t,\tau) \tag{2.26}$$

and using integrating factor method we get a the solution

$$z_j(t,\tau) = e^{\lambda_j(t)\tau} \int e^{-\lambda_j(t)\tau} g_j(t,\tau) \mathrm{d}\tau, \tag{2.27}$$

that, using g_j from the equation (2.25), rewrites as

$$z_j(t,\tau) = e^{\lambda_j(t)\tau} \left\{ \left[\vec{w}^{\mathrm{T}}{}_j(t) \boldsymbol{A}_1(t) a_j(t) \vec{v}_j(t) - \frac{\mathrm{d}a_j}{\mathrm{d}t} \right] \int \mathrm{d}\tau - a_j(t) \frac{\mathrm{d}\lambda_j}{\mathrm{d}t} \int \tau \mathrm{d}\tau + \right. \tag{2.28}$$

$$+ h_j(t) \int e^{-\lambda_j(t)\tau} \mathrm{d}\tau + \sum_{k \neq j} \left[\vec{w}^{\mathrm{T}}{}_j(t) \boldsymbol{A}_1(t) a_k(t) \vec{v}_k(t) - \xi_{jk} a_k(t) \right] \int e^{(\lambda_k(t) - \lambda_j(t))\tau} \mathrm{d}\tau \Bigg\}.$$

For the integration we have to consider the structure of the eigenvalues of our system (i.e. no multiple eigenvalues and a zero eigenvalue at $j = 0$). For zero eigenvalue $j = 0$ we get the result

$$\begin{aligned} z_0(t,\tau) = &\left[\vec{w}^{\mathrm{T}}{}_0(t) \boldsymbol{A}_1(t) a_0(t) \vec{v}_0(t) - \frac{\mathrm{d}a_0}{\mathrm{d}t} + h_0(t) \right] \tau + \\ &+ \sum_{k>0} \left[\vec{w}^{\mathrm{T}}{}_0(t) \boldsymbol{A}_1(t) a_k(t) \vec{v}_k(t) - \xi_{0k} a_k(t) \right] \frac{e^{\lambda_k(t)\tau}}{\lambda_k(t)} + K_0(t), \end{aligned} \tag{2.29}$$

where $K_0(t)$ is an integrating constant, that is be found using given initial values.

Since all the eigenvalues in the exponential are negative, the term in the sum of equation (2.29) decays to 0 as $\tau \to \infty$, as

$$\lim_{\tau \to \infty} z_0(t,\tau) = \left[\vec{w}^{\mathrm{T}}{}_0(t) \boldsymbol{A}_1(t) a_0(t) \vec{v}_0(t) - \frac{\mathrm{d}a_0}{\mathrm{d}t} + h_0(t) \right] \tau + K_0(t). \tag{2.30}$$

We know that the $\vec{X}(t)_0$ is bounded, so the $\vec{Y}_0(t,\tau)$ is bounded as well, and therefore 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, that contains the τ, must be zero

$$0 = \vec{w}^{\mathrm{T}}{}_0(t) \boldsymbol{A}_1(t) a_0(t) \vec{v}_0(t) - \frac{\mathrm{d}a_0}{\mathrm{d}t} + h_0(t), \tag{2.31}$$

which yields the following differential equation

$$\frac{\mathrm{d}a_0}{\mathrm{d}t} = a_0(t) \vec{w}^{\mathrm{T}}{}_0(t) \boldsymbol{A}_1(t) \vec{v}_0(t) + \vec{w}^{\mathrm{T}}{}_0(t) \vec{H}(t). \tag{2.32}$$

This equation defines a steady-state point on invariant manifold of the system (2.5a). The relation with the original variables $\vec{Y}_0(t)$ and the initial conditions are calculated from (2.18) at the limit $\tau \to \infty$ as

$$\vec{Y}_0 = \vec{v}_0(t) a_0(t), \tag{2.33a}$$

$$a_0(t) = \vec{w}^{\mathrm{T}}{}_0(t) \vec{Y}_0(t). \tag{2.33b}$$

2.1.2 Correction Term for General Systems

Definition of the System

Similarly to Tikhonov Fenichel suggest a method which can be used for reduction of the dimensionality of the system[21]. This method improves the approximation by employing a correction term. This subsection describes the method in more detail.

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

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \vec{f}(\vec{u}) + \varepsilon\vec{h}(\vec{u}), \tag{2.34}$$

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

$$\vec{u} = \vec{U}(\vec{a}(t)) + \varepsilon\vec{v}(t), \tag{2.35}$$

where the state of the system is given by a combination of a solution on an invariant attracting manifold $\vec{U}(\vec{a}(t)) \in \mathbb{R}^{n+1}$ and a small perturbation $\vec{v}(t) \in \mathbb{R}^{n+1}$, which for our purposes is called correction term[22]. Vector $\vec{a}(t) \in A \subset \mathbb{R}^{m+1}, m < n+1$ 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 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

$$\boldsymbol{F}(\vec{a}(t))\vec{V}_i(\vec{a}) = \Lambda_i\vec{V}_i(\vec{a}), \tag{2.36a}$$
$$\vec{W^T}_i(\vec{a})\boldsymbol{F}(\vec{a}(t)) = \Lambda_i\vec{W^T}_i(\vec{a}), \tag{2.36b}$$
$$W^T{}_iV_i = \delta_{ji} \tag{2.36c}$$

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

$$\vec{W^T}_k(\vec{a})\vec{v}(t) = 0 \tag{2.37}$$

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

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

$$\frac{\partial\vec{f}(\vec{U}(\vec{a}))}{\partial a_k} = \frac{\partial\vec{f}(\vec{U}(\vec{a}))}{\partial\vec{U}(\vec{a})}\frac{\partial\vec{U}(\vec{a})}{\partial a_k} = \frac{\partial\vec{f}(\vec{U}(\vec{a}))}{\partial\vec{u}}\frac{\partial\vec{U}(\vec{a})}{\partial a_k} = \boldsymbol{F}(\vec{a})\frac{\partial\vec{U}(\vec{a})}{\partial a_k}. \tag{2.38}$$

We choose $\vec{V}_k$ such that

$$\vec{V}_k = \frac{\partial \vec{U}}{\partial a_k} \tag{2.39}$$

for $k = 0, \dots, m$ are the eigenvectors corresponding to zero eigenvalues $\Lambda_k = 0$.

In the subsection 2.1.1 we aim to show that the left and right eigenvectors corresponding to the same eigenvalue can be chosen 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 (2.10), which is a constant and therefore has a derivative

$$0 = \frac{\partial \delta_{ij}}{\partial a_k} = \vec{W^T}_i \frac{\partial \vec{V}_j(\vec{a}(t))}{\partial a_k} + \frac{\partial \vec{W^T}_i}{\partial a_k} \vec{V}_j(\vec{a}(t)). \tag{2.40}$$

We define

$$K_{ijk} = -\vec{W^T}_i \frac{\partial \vec{V}_j(\vec{a}(t))}{\partial a_k} = \frac{\partial \vec{W^T}_i}{\partial a_k} \vec{V}_j(\vec{a}(t)) \tag{2.41}$$

and say that when the $K_{ijk} = 0$ the eigenvector $\vec{W^T}_i$ and $\vec{V}_i$ are dynamically orthogonal with respect to a_k. The variable K_{ijk} is used for $i = 0, \dots, m$ and $j = m+1, \dots, n$ from different sets, while k is only defined for $k = 0, \dots, m$.

Expansion of the Correction Term

We expand the correction term in the basis of eigenvectors as

$$\vec{v}(t) = \sum_{j=m+1}^{n+1} \vec{V}_j(\vec{a}(t)) b_j(t), \tag{2.42}$$

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 notation for the flow on the invariant manifold

$$\frac{\mathrm{d}a_k}{\mathrm{d}t} = \varepsilon \mathcal{A}_k \tag{2.43}$$

for $k = 0, \dots, m$. This is substituted into an expression of the derivative of the correction term

$$\begin{aligned} \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} &= \sum_{j=m+1}^{n+1} \left[\frac{\mathrm{d}\vec{V}_j}{\mathrm{d}t} b_j(t) + \vec{V}_j(\vec{a}(t)) \frac{\mathrm{d}b_j}{\mathrm{d}t} \right] = \\ &= \sum_{j=m+1}^{n+1} \sum_{k=0}^{m} \frac{\partial \vec{V}_j}{\partial a_k} \frac{\mathrm{d}a_k}{\mathrm{d}t} b_j(t) + \sum_{j=m+1}^{n+1} \vec{V}_j(\vec{a}(t)) \frac{\mathrm{d}b_j}{\mathrm{d}t} = \end{aligned}$$

$$= \sum_{j=m+1}^{n+1} \sum_{k=0}^{m} \frac{\partial \vec{V}_j}{\partial a_k} \varepsilon \mathcal{A}_k b_j + \sum_{j=m+1}^{n+1} \vec{V}_j(\vec{a}(t)) \frac{\mathrm{d} b_j}{\mathrm{d}t}. \tag{2.44}$$

Taylor Expansion in Multiple Variables

For convenience we also recall the Taylor series expansion of function $\vec{f}(\vec{u})$ in multiple variables which is done around the point $\vec{U} = [U_0, \ldots, U_{n+1}]$ as

$$\begin{aligned}
&\vec{f}(u_0, \ldots, u_{n+1}) = \\
&= \sum_{i_1=0}^{\infty} \cdots \sum_{i_{n+1}=0}^{\infty} \left(\frac{\partial^{i_1+\ldots+i_{n+1}} \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_0^{i_1} \ldots \partial u_{n+1}^{i_{n+1}}} \right) \frac{(u_0 - U_0)^{i_1} \ldots (u_{n+1} - U_{n+1})^{i_{n+1}}}{i_1! \ldots i_{n+1}!} = \\
&= \vec{f}(U_0, \ldots, U_{n+1}) + \sum_{j=0}^{n+1} \frac{\partial \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_j} (u_j - U_j) + \\
&\quad + \frac{1}{2!} \sum_{j=0}^{n+1} \sum_{k=0}^{n+1} \frac{\partial^2 \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_j \partial u_k} (u_j - U_j)(u_k - U_k) + \\
&\quad + \frac{1}{3!} \sum_{j=0}^{n+1} \sum_{k=0}^{n+1} \sum_{l=0}^{n+1} \frac{\partial^3 \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_j \partial u_k \partial u_l} (u_j - U_j)(u_k - U_k)(u_l - U_l) + \cdots = \\
&= \vec{f^U} + \sum_{j=0}^{n+1} \vec{f^U}_j (u_j - U_j) + \sum_{j=0}^{n+1} \sum_{k=0}^{n+1} \vec{f^U}_{jk} (u_j - U_j)(u_k - U_k) + \\
&\quad + \sum_{j=0}^{n+1} \sum_{k=0}^{n+1} \sum_{l=0}^{n+1} \vec{f^U}_{jk\ell} (u_j - U_j)(u_k - U_k)(u_l - U_l) + \ldots,
\end{aligned} \tag{2.45}$$

where the Taylor coefficients of the function $\vec{f}(u_1, \ldots, u_{n+1})$ at point $\vec{U}$ are

$$\vec{f^U} = \vec{f}(U_0, \ldots, U_{n+1}), \tag{2.46a}$$

$$\vec{f^U}_j = \frac{\partial \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_j}, \tag{2.46b}$$

$$\vec{f^U}_{jk} = \frac{1}{2!} \frac{\partial^2 \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_j \partial u_k}, \tag{2.46c}$$

$$\vec{f^U}_{jk\ell} = \frac{1}{3!} \frac{\partial^3 \vec{f}(U_0, \ldots, U_{n+1})}{\partial u_j \partial u_k \partial u_\ell}. \tag{2.46d}$$

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

Expanding the System around the Invariant Manifold

We substitute (2.35) to (2.34) and rewrite the left hand side as

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \frac{\mathrm{d}\vec{U}}{\mathrm{d}t} + \varepsilon \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \sum_{k=0}^{m} \frac{\partial \partial \vec{U}}{\partial a_k} \frac{\mathrm{d} a_k}{\mathrm{d}t} + \varepsilon \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} =$$

$$=\varepsilon\left(\sum_{k=0}^{m}\vec{V}_k(\vec{a}(t))\mathcal{A}_k+\frac{\mathrm{d}\vec{v}}{\mathrm{d}t}\right)=$$
$$=\varepsilon\sum_{k=0}^{m}\vec{V}_k(\vec{a}(t))\mathcal{A}_k+\varepsilon\sum_{j=m+1}^{n+1}\vec{V}_j(\vec{a}(t))\frac{\mathrm{d}b_j}{\mathrm{d}t}+\varepsilon^2\sum_{j=m+1}^{n+1}\sum_{k=0}^{m}\frac{\partial\vec{V}_j}{\partial a_k}\mathcal{A}_k b_j, \tag{2.47}$$

and right hand side by using Taylor expansion (2.45) as

$$\vec{f}(\vec{u})+\varepsilon\vec{h}(\vec{u})=\vec{f}(\vec{U}(\vec{a}(t))+\varepsilon\vec{v}(t))+\varepsilon\vec{h}(\vec{U}(\vec{a}(t))+\varepsilon\vec{v}(t))=$$
$$=\vec{f^U}+\varepsilon\sum_{j=0}^{n+1}\vec{f^U}_j v_j(t)+\varepsilon^2\sum_{j=0}^{n+1}\sum_{k=0}^{n+1}\vec{f^U}_{jk}v_j(t)v_k(t)+\varepsilon\vec{h^U}+\varepsilon^2\sum_{j=0}^{n+1}\vec{h^U}_j v_j(t)+\mathcal{O}(\varepsilon^3), \tag{2.48}$$

where 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 both side (2.47) and (2.48) together we obtain

$$\sum_{k=0}^{m}\vec{V}_k(\vec{a}(t))\mathcal{A}_k+\sum_{j=m+1}^{n+1}\vec{V}_j(\vec{a}(t))\frac{\mathrm{d}b_j}{\mathrm{d}t}+\varepsilon\sum_{j=m+1}^{n+1}\sum_{k=0}^{m}\frac{\partial\vec{V}_j}{\partial a_k}\mathcal{A}_k b_j=$$
$$=\sum_{j=0}^{n+1}\vec{f^U}_j v_j+\varepsilon\sum_{j=0}^{n+1}\sum_{k=0}^{n+1}\vec{f^U}_{jk}v_j(t)v_k(t)+\vec{h^U}+\varepsilon\sum_{j}\vec{h^U}_j v_j(t)+\mathcal{O}(\varepsilon^2). \tag{2.49}$$

We are going to we 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 the first term on the RHS as

$$\vec{W^T}_i\sum_{j=0}^{n+1}\vec{f^U}_j v_j(t)=\vec{W^T}_i\sum_{j=0}^{n+1}\frac{\partial\vec{f}(\vec{U}(a))}{\partial u_j}v_j(t)=\vec{W^T}_i\boldsymbol{F}(\vec{a})\vec{v}(t)=$$
$$=\Lambda_i\vec{W^T}_i\vec{v}(t)=\Lambda_i\vec{W^T}_i\sum_{j=0}^{n+1}\vec{V}_j(\vec{a}(t))b_j(t)=\Lambda_i b_i(t), \tag{2.50}$$

and define

$$H_i(t)=\vec{W^T}_i\vec{h^U}. \tag{2.51}$$

We do the multiplication of (2.49) to get

$$\frac{\mathrm{d}b_i}{\mathrm{d}t}=\Lambda_i b_i(t)+H_i-\mathcal{A}_i+\varepsilon\left\{\sum_{j=0}^{n+1}\vec{W^T}_i\vec{h^U}_j v_j(t)+\sum_{j=m+1}^{n+1}\sum_{k=0}^{n+1}\left[K_{ijk}\mathcal{A}_k b_j(t)+\right.\right.$$
$$\left.\left.+\vec{W^T}_i\vec{f^U}_{jk}v_j(t)v_k(t)\right]\right\}+\mathcal{O}(\varepsilon^2). \tag{2.52}$$

where subscripts of v_k denote the entries of the correction vector (2.42), and K_{ijk} was defined in (2.41).

We write the system as

$$\frac{\mathrm{d}b_i}{\mathrm{d}t} = \Lambda_i b_i(t) + H_i - \mathcal{A}_i + \varepsilon \mathcal{F}_i(\vec{b}, \vec{\mathcal{A}}) + \mathcal{O}(\varepsilon^2), \tag{2.53}$$

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

$$\begin{aligned}\mathcal{F}_i(\vec{b}, \vec{\mathcal{A}}) = & \sum_{j=0}^{n+1} \sum_{k=m+1}^{n+1} \vec{W^T}_i \vec{h^U}_j V_{kj}(\vec{a}) b_k(t) + \sum_{j=m+1}^{n+1} \sum_{k=0}^{m} K_{ijk} \mathcal{A}_k b_j(t) + \\ & + \sum_{j,k=0}^{n+1} \sum_{l,q=m+1}^{n+1} \vec{W^T}_i \vec{f^U}_{jk} V_{lj}(\vec{a}) b_l(t) V_{qk}(\vec{a}) b_q(t).\end{aligned} \tag{2.54}$$

Solution of the Reduced General System

To solve the system we impose the condition of orthogonality (2.37) which says that $b_j = 0$ for $j = 0, \ldots, m$. We also recall that the flow $\mathcal{A}_j$ is only defined for for $j = 0, 1, \ldots m$. Then we split the equations into two sets for $i < m + 1$ as

$$\mathcal{A}_i = H_i + \varepsilon \mathcal{F}_i(\vec{b}, \vec{\mathcal{A}}) + \mathcal{O}(\varepsilon^2) \qquad \text{for } i = 0, 1, \ldots, m, \text{ and} \tag{2.55a}$$

$$\frac{\mathrm{d}b_i}{\mathrm{d}t} = \Lambda_i b_i(t) + H_i + \mathcal{O}(\varepsilon) \qquad \text{for } i = m + 1, m + 2, \ldots, n + 1. \tag{2.55b}$$

To solve the $\mathcal{A}_i$ 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 the equation (2.55a) is only present in terms which already contain the coefficient ε.

To find of b_i we use the integrating factor

$$\exp\left(-\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right) \tag{2.56}$$

that multiplies the equation (2.55b) to get

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\exp\left(-\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right) b_i(t)\right] = \exp\left(-\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right) H_i(t) + \mathcal{O}(\varepsilon). \tag{2.57}$$

This formula is multiplied by $\mathrm{d}t$ and the dependent variable t is substituted by τ which yields definite integral with limits s_i, t

$$\exp\left(-\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right) b_i(t) = \int_{s_i}^t \exp\left(-\int_r^\tau \Lambda_i(\xi)\mathrm{d}\xi\right) H_i(\tau)\mathrm{d}\tau + C_i + \mathcal{O}(\varepsilon), \tag{2.58}$$

and where C_i is an arbitrary constant to be chosen. Finally, we divide by the exponential function to get

$$b_i(t) = \int_{s_i}^t \exp\left(\int_\tau^t \Lambda_i(\xi)\mathrm{d}\xi\right) H_i(\tau)\mathrm{d}\tau + D_i + \mathcal{O}(\varepsilon), \tag{2.59}$$

where the arbitrary constant

$$D_i(t) = C_i \exp\left(\int_r^t \Lambda_i(\xi)\mathrm{d}\xi\right). \tag{2.60}$$

The equation (2.59) is integrated after expanding the terms according to the Taylor series around a reference point t^* as

$$\begin{aligned} \Lambda_i(\xi) =& \Lambda_i(t^*) + \mathcal{O}(\varepsilon), && \text{(2.61a)} \\ H_i(\tau) =& H_i(t^*) + \mathcal{O}(\varepsilon). && \text{(2.61b)} \end{aligned}$$

which is substituted into (2.59) and yields

$$b_i(t) = H_i(t^*) \int_{s_i}^t \exp\left[\Lambda_i(t^*)(t-\tau)\right] \mathrm{d}\tau + D_i(t) + \mathcal{O}(\varepsilon), \tag{2.62}$$

which yields the formula

$$b_i(t) = -\frac{H_i(t^*)}{\Lambda_i(t^*)}\left\{1 - \exp\left[\Lambda_i(t^*)(t-s_i)\right]\right\} + D_i(t) + \mathcal{O}(\varepsilon). \tag{2.63}$$

The arbitrary constant $D_i(t)$ is conveniently chosen as

$$D_i(t) = -\frac{H_i(t^*)}{\Lambda_i(t^*)} \exp\left[\Lambda_i(t^*)(t-s_i)\right], \tag{2.64}$$

so that the exponentials cancel out. Then the final equation for the coordinates across the invariant manifold gets the form

$$b_i(t) = -\frac{H_i(t)}{\Lambda_i(t)} + \mathcal{O}(\varepsilon) = -\frac{\vec{W^T}_i\vec{h^U}(t)}{\Lambda_i(t)} + \mathcal{O}(\varepsilon). \tag{2.65}$$

To find the solution on the invariant manifold we put the result (2.65) into (2.55a), and using the $\mathrm{d}a_i/\mathrm{d}t = \varepsilon\mathcal{A}_i$ we obtain

$$\begin{aligned} &\frac{\mathrm{d}a_i}{\mathrm{d}t} = \varepsilon \vec{W^T}_i\vec{h^U} + \\ &\varepsilon^2\left\{-\sum_{j=0}^{n+1}\sum_{k=m+1}^{n+1} \vec{W^T}_i\vec{h^U}_j V_{kj}(\vec{a})\frac{\vec{W^T}_k\vec{h^U}(t)}{\Lambda_k(t)} - \sum_{j=m+1}^{n+1}\sum_{k=0}^{m} K_{ijk}\vec{W^T}_k\vec{h^U}(t)\frac{\vec{W^T}_j\vec{h^U}(t)}{\Lambda_j(t)} + \right. \\ &\left. + \sum_{j,k=0}^{n+1}\sum_{\ell,q=m+1}^{n+1} \vec{W^T}_i\vec{f^U}_{jk}V_{lj}(\vec{a})\frac{\vec{W^T}_\ell\vec{h^U}(t)}{\Lambda_\ell(t)}V_{qk}(\vec{a})\frac{\vec{W^T}_q\vec{h^U}(t)}{\Lambda_q(t)}\right\} + \mathcal{O}(\varepsilon^3). \end{aligned} \tag{2.66}$$

2.1.3 Second Order Correction Term for Markov Chains

Autonomisation of Markov Chains

Here we apply the dimensionality reduction using correction term for a Markov chain model, i.e. a system of linear ordinary differential equations. The Markov chain models of ionic channels are normally non-autonomous. As the theory was developed for autonomous system of ODEs, we extend the Markov chain into an autonomous system using an additional variable σ as

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \left[\frac{1}{\varepsilon}\boldsymbol{A}_0(\sigma) + \boldsymbol{A}_1(\sigma)\right]\vec{x} + \vec{H}(\sigma), \tag{2.67a}$$

$$\frac{\mathrm{d}\sigma}{\mathrm{d}t} = 1, \tag{2.67b}$$

where vectors $\vec{x}, \vec{H} \in \mathbb{R}^n$; matrices $\boldsymbol{A}_0(\sigma), \boldsymbol{A}_1(\sigma) \in \mathbb{R}^{n\times n}$; parameters $\sigma, \varepsilon \in \mathbb{R}$; where $\varepsilon \to 0$ is a small number.

The time is re-scaled as analogously to (2.2) by introducing slow-time τ.

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}\tau} = [\boldsymbol{A}_0(\sigma) + \varepsilon\boldsymbol{A}_1(\sigma)]\vec{x} + \varepsilon\vec{H}(\sigma), \tag{2.68a}$$

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\tau} = \varepsilon. \tag{2.68b}$$

We group the terms according to their order of ε. This allows to rewrite the system to the form of (2.34), in which

$$\vec{f}(\vec{u}) = \begin{bmatrix} \boldsymbol{A}_0(\sigma)\vec{x} \\ 0 \end{bmatrix}, \qquad \vec{h} = \begin{bmatrix} \boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \\ 1 \end{bmatrix}, \text{and} \qquad \vec{u} = \begin{bmatrix} \vec{x} \\ \sigma \end{bmatrix}. \tag{2.69}$$

The solution in the leading order lies on the invariant m-dimensional manifold $\vec{U}$ such that $\vec{f}(\vec{U}) = 0$ where

$$\vec{U} = \begin{cases} \vec{x} \in \ker(\boldsymbol{A}_0(\sigma)) \\ \sigma \in \mathbb{R} \end{cases}. \tag{2.70}$$

We recall that the kernel is a subspace which is projected to a null vector by the matrix $\boldsymbol{A}_0(\sigma)$. This condition is satisfied for the linear combination of the eigenvectors of $\boldsymbol{A}_0(\sigma)$ that corresponds to zero eigenvalues $\lambda_j(\sigma) = 0$.

We have ordered those eigenvalues as $\lambda_k(\sigma) = 0$ for $k = 1, \ldots, m$, and $\lambda_j(\sigma) \neq 0$ for $j = m+1, \ldots, n$, where m denotes the multiplicity of the zero eigenvalue of the matrix $\boldsymbol{A}_0$. The solution can be written in a form analogical to the equation (2.17) that gives the transformation of the old coordinates $\vec{x}$ in terms of the coordinates on the invariant manifold $\vec{a}$ as

$$\vec{x} = \sum_{k=1}^{m} a_k \vec{v}_k(\sigma). \tag{2.71}$$

Then the $\vec{U}$ can be written as

$$\vec{U} = \begin{bmatrix} \sum_{j=1}^{m} a_j \vec{v}_j(\sigma) \\ \sigma \end{bmatrix}. \tag{2.72}$$

The Jacobian $\boldsymbol{F}(\vec{u})$ of this system is

$$\boldsymbol{F}(\vec{u}) = \frac{\partial \vec{f}(\vec{u})}{\partial u} = \begin{bmatrix} \frac{\partial}{\partial x}(\boldsymbol{A}_0(\sigma)\vec{x}) & \frac{\partial}{\partial \sigma}(\boldsymbol{A}_0(\sigma)\vec{x}) \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_0(\sigma) & \frac{\partial \boldsymbol{A}_0(\sigma)}{\partial \sigma}\vec{x} \\ 0 & 0 \end{bmatrix}. \tag{2.73}$$

Eigenvalues and Eigenvectors

We find the eigenvalues and eigenvectors according to the equation (2.36a) which gives the relation

$$\vec{V}_i = \begin{bmatrix} \vec{\kappa}_i \\ \mu_i \end{bmatrix}, \tag{2.74}$$

where $\vec{\kappa} \in \mathbb{R}^n$ 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

$$\boldsymbol{A}_0(\sigma)\vec{\kappa}_i + \frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\vec{x}\mu_i = \Lambda_i \vec{\kappa}_i, \tag{2.75a}$$

$$0\vec{\kappa}_i + 0\mu_i = \Lambda_i \mu_i. \tag{2.75b}$$

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

$$\vec{V}_i = \begin{bmatrix} \vec{v}_i(\sigma) \\ 0 \end{bmatrix}, \qquad \Lambda_i = \lambda_i, \tag{2.76}$$

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

So we have to find one more linearly independent solution, which corresponds to the extended space when the $\mu \neq 0$. Then the equation (2.75b) implies $\Lambda_k = 0$ (we let this index for this solution to be $k = 0$). For $\Lambda_0 = 0$ the equation (2.75a) gets the form

$$\boldsymbol{A}_0(\sigma)\vec{\kappa}_0 + \frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\vec{x}\mu_0 = 0 \tag{2.77}$$

and expanding $\vec{x}$ according to its definition (2.71) we get

$$\boldsymbol{A}_0(\sigma)\vec{\kappa}_0 + \mu_0 \frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma} \sum_{k=1}^{m} a_k \vec{v}_k = 0. \tag{2.78}$$

The relation $\boldsymbol{A}_0(\sigma)\vec{v}_k(\sigma) = 0$ is satisfied because of the zero eigenvalues $\lambda_k = 0$ for $k = 1, \ldots, m$. The derivative $\frac{\mathrm{d}}{\mathrm{d}\sigma}(\boldsymbol{A}_0(\sigma)\vec{v}_k) = 0$ gives the following relation according product rule

$$\frac{\mathrm{d}\boldsymbol{A}_0(\sigma)}{\mathrm{d}\sigma}\vec{v}_k(\sigma) = -\boldsymbol{A}_0(\sigma)\frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma}. \tag{2.79}$$

Then the (2.78) yields

$$\boldsymbol{A}_0(\sigma)\left(\vec{\kappa}_0 - \mu_0 \sum_{k=1}^{m} a_k \frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma}\right) = 0. \tag{2.80}$$

which gives us the eigenvector

$$\vec{\kappa}_0 = \mu_0 \sum_{k=1}^{m} a_k \frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma}. \tag{2.81}$$

As we can choose any linear combination of that vector, we choose $\mu_0 = 1$, to get the remaining eigenvector of the Jacobian $\boldsymbol{F}(\sigma)$ as

$$\vec{V}_0 = \begin{bmatrix} \sum_{k=1}^{m} a_k \frac{\mathrm{d}\vec{v}_k(\sigma)}{\mathrm{d}\sigma} \\ 1 \end{bmatrix} \tag{2.82}$$

for eigenvalue $\Lambda_0 = 0$.

The adjoint vectors are defined according the equation (2.36b) and satisfy the Kronecker delta function as $\vec{W^T}_j \vec{V}_i = \delta_{ji}$. So, we find the adjoint vectors for the eigenvectors (2.76) and (2.82) as

$$\vec{W^T}_0 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \tag{2.83a}$$

$$\vec{W^T}_i = \begin{bmatrix} \vec{w}_i & -\sum_{j=1}^{m} a_j \vec{w}_i \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}\sigma} \end{bmatrix}, \tag{2.83b}$$

where $i = 1, \ldots, n$.

Term K_{ijk} defined by (2.41) is found substituting the adjoint and eigenvectors. The possible combinations are then $K_{0jk} = K_{0j0} = 0$ (this is because $\vec{W^T}_0$ is a constant vector), and

$$K_{ij0} = -\vec{W^T}_i \frac{\partial \vec{V}_j}{\partial a_0} = \frac{\partial \vec{W^T}_i}{\partial a_0}\vec{V}_j, \tag{2.84a}$$

$$K_{ijk} = -\vec{W^T}_i \frac{\partial \vec{V}_j}{\partial a_k} = \frac{\partial \vec{W^T}_i}{\partial a_k}\vec{V}_j, \tag{2.84b}$$

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

Taylor Coefficients

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

$$h^{\vec{U}}{}_j = \frac{\partial h^{\vec{U}}}{\partial u_j} = \frac{\partial}{\partial u_j}\begin{bmatrix} \boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial u_j}\left(\boldsymbol{A}_1(\sigma)\vec{x}\right) + \frac{\partial \vec{H}}{\partial u_j} \\ 0 \end{bmatrix}, \tag{2.85}$$

$$h^{\vec{U}}{}_{jk} = \frac{\partial^2 h^{\vec{U}}}{\partial u_j \partial u_k} = \frac{\partial^2}{\partial u_j \partial u_k}\begin{bmatrix} \boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\partial^2}{\partial u_j \partial u_k}\left(\boldsymbol{A}_1(\sigma)\vec{x}\right) + \frac{\partial^2 \vec{H}(\sigma)}{\partial u_j \partial u_k} \\ 0 \end{bmatrix}. \tag{2.86}$$

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

$$f^{\vec{U}}{}_j = \frac{\partial f^{\vec{U}}}{\partial u_j} = \begin{bmatrix} \frac{\partial}{\partial u_j}\left(\boldsymbol{A}_0(\sigma)\vec{x}\right) \\ 0 \end{bmatrix}, \tag{2.87a}$$

$$f^{\vec{U}}{}_{jk} = \frac{\partial^2 f^{\vec{U}}}{\partial u_j \partial u_k} = \begin{bmatrix} \frac{\partial^2}{\partial u_j \partial u_k}\left(\boldsymbol{A}_0(\sigma)\vec{x}\right) \\ 0 \end{bmatrix}, \tag{2.87b}$$

$$f^{\vec{U}}{}_{jk\ell} = \frac{\partial^3 f^{\vec{U}}}{\partial u_j \partial u_k \partial u_\ell} = \begin{bmatrix} \frac{\partial^2}{\partial u_j \partial u_k}\left(\boldsymbol{A}_0(\sigma)\vec{x}\right) \\ 0 \end{bmatrix}. \tag{2.87c}$$

Solution of the Reduced Markov Chain

Now we substitute into (2.66) for $i = 0$ as

$$\begin{aligned} \frac{\mathrm{d}a_0}{\mathrm{d}t} &= \vec{W^T}{}_0 h^{\vec{U}} + \\ \varepsilon &\left\{ -\sum_{j=0}^{n}\sum_{k=m+1}^{n} \vec{W^T}{}_0 h^{\vec{U}}{}_j V_{kj}(\vec{a}) \frac{\vec{W^T}{}_k h^{\vec{U}}(t)}{\Lambda_k(t)} - \sum_{j=m+1}^{n}\sum_{k=0}^{m} K_{0jk} \vec{W^T}{}_k h^{\vec{U}}(t) \frac{\vec{W^T}{}_j h^{\vec{U}}(t)}{\Lambda_j(t)} + \right. \\ &\left. + \sum_{j,k=0}^{n}\sum_{\ell,q=m+1}^{n} \vec{W^T}{}_0 f^{\vec{U}}{}_{jk} V_{lj}(\vec{a}) \frac{\vec{W^T}{}_\ell h^{\vec{U}}(t)}{\Lambda_\ell(t)} V_{qk}(\vec{a}) \frac{\vec{W^T}{}_q h^{\vec{U}}(t)}{\Lambda_q(t)} \right\} + \mathcal{O}(\varepsilon^2) = 1 \end{aligned} \tag{2.88}$$

and for $i \neq 0$ as

$$\begin{aligned} \frac{\mathrm{d}a_i}{\mathrm{d}t} =& \vec{w}_i \left(\boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \right) - \sum_{j=1}^{m} a_j \vec{w}_i \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}\sigma} + \\ & + \varepsilon \left\{ -\sum_{j=0}^{n}\sum_{k=m+1}^{n} \vec{w}_i \left(\frac{\partial}{\partial u_j}\left(\boldsymbol{A}_1(\sigma)\vec{x}\right) + \frac{\partial \vec{H}}{\partial u_j} \right) \right. \\ & V_{kj}(\vec{a}) \frac{\vec{w}_k \left(\boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \right) - \sum_{\ell=1}^{m} a_\ell \vec{w}_k \frac{\mathrm{d}\vec{v}_\ell}{\mathrm{d}\sigma}}{\Lambda_k(t)} + \end{aligned}$$

$$- \sum_{j=m+1}^{n} \sum_{k=0}^{m} K_{ijk} \vec{W^T}_k \vec{h^U}(t) \frac{\vec{W^T}_j \vec{h^U}(t)}{\Lambda_j(t)} +$$
$$+ \sum_{j,k=0}^{n} \sum_{\ell,q=m+1}^{n} \left[\vec{w}_i \frac{\partial^2}{\partial u_j \partial u_k} (\boldsymbol{A}_0(\sigma)\vec{x}) \right] \times$$
$$V_{\ell j}(\vec{a}) \frac{\vec{w}_\ell \left(\boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \right) - \sum_{p=1}^{m} a_p \vec{w}_\ell \frac{\mathrm{d}\vec{v}_p}{\mathrm{d}\sigma}}{\Lambda_\ell(t)}$$
$$V_{qk}(\vec{a}) \frac{\vec{w}_q \left(\boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \right) - \sum_{p=1}^{m} a_p \vec{w}_q \frac{\mathrm{d}\vec{v}_p}{\mathrm{d}\sigma}}{\Lambda_q(t)} \Bigg\} + \mathcal{O}(\varepsilon^2). \quad (2.89)$$

Then we substitute for $\vec{x}$ according to (2.71), so that the leading order solution yields

$$\frac{\mathrm{d}a_i}{\mathrm{d}t} = \vec{w}_i \boldsymbol{A}_1(\sigma) \sum_{j=1}^{m} a_j \vec{v}_j + \vec{w}_i \vec{H}(\sigma) - \sum_{j=1}^{m} a_j \vec{w}_i \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}\sigma}. \quad (2.90)$$

Introducing a new variable corresponding to the leading order term as

$$L_i(\sigma) = \vec{w}_i \left(\boldsymbol{A}_1(\sigma)\vec{x} + \vec{H}(\sigma) \right) - \sum_{j=1}^{m} a_j \vec{w}_i \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}\sigma}. \quad (2.91)$$

the equation (2.89) simplifies as

$$\frac{\mathrm{d}a_i}{\mathrm{d}t} = L_i(\sigma) + \varepsilon \Bigg\{ - \sum_{j=0}^{n} \sum_{k=m+1}^{n} \vec{w}_i \left(\frac{\partial}{\partial u_j} (\boldsymbol{A}_1(\sigma)\vec{x}) + \frac{\partial \vec{H}}{\partial u_j} \right) \frac{V_{kj}(\vec{a}) L_k(\sigma)}{\Lambda_k(t)} +$$
$$+ \sum_{j,k=0}^{n} \sum_{\ell,q=m+1}^{n} \left[\vec{w}_i \frac{\partial^2}{\partial u_j \partial u_k} (\boldsymbol{A}_0(\sigma)\vec{x}) \right] \frac{V_{\ell j}(\vec{a}) L_\ell(\sigma)}{\Lambda_\ell(t)} \frac{V_{qk}(\vec{a}) L_q(\sigma)}{\Lambda_q(t)} \Bigg\} + \mathcal{O}(\varepsilon^2). \quad (2.92)$$

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 $\frac{\partial}{\partial u_j} (\boldsymbol{A}_1(\sigma)\vec{x})$ is split for u_j when $j = 0, \ldots, n-1$. In this case the $u_j = x_j$, and for $j = n$ in which case the $u_n = \sigma$. We obtain the relation

$$\frac{\partial}{\partial u_j} (\boldsymbol{A}_1(\sigma)\vec{x} + H(\sigma)) = \vec{A}_1^j(\sigma), \qquad \text{for } j = 0, \ldots, n-1, \quad (2.93a)$$
$$\frac{\partial}{\partial u_n} (\boldsymbol{A}_1(\sigma)\vec{x} + H(\sigma)) = \frac{\partial A_1}{\partial \sigma} \vec{x} + \frac{\partial H}{\partial \sigma}. \quad (2.93b)$$

Accordingly we obtain the relation for the second derivatives as

$$\frac{\partial^2}{\partial u_j \partial u_k} (\boldsymbol{A}_0(\sigma)\vec{x}) = \frac{\partial \vec{A}_0^j}{\partial u_k} = 0, \qquad \text{for } j,k = 0, \ldots, n-1, \quad (2.94a)$$
$$\frac{\partial^2}{\partial u_j \partial u_n} (\boldsymbol{A}_0(\sigma)\vec{x}) = \frac{\partial}{\partial u_j} \left(\frac{\partial A_0}{\partial \sigma} \vec{x} \right) = \frac{\partial \vec{A}_0^j}{\partial \sigma}, \qquad \text{for } j = 0, \ldots, n-1, \quad (2.94b)$$

$$\frac{\partial^2}{\partial {u_n}^2}\left(\boldsymbol{A}_0(\sigma)\vec{x}\right) = \frac{\partial^2 A_0(\sigma)}{\partial \sigma^2}\vec{x}. \tag{2.94c}$$

We substitute into the equation (2.92) as

$$\begin{aligned}\frac{\mathrm{d}a_i}{\mathrm{d}t} =& L_i(\sigma) + \varepsilon\Bigg\{ -\sum_{j=0}^{n-1}\sum_{k=m+1}^{n} \vec{w}_i \vec{A}_1^j(\sigma)\frac{V_{kj}(\vec{a})L_k(\sigma)}{\Lambda_k(t)} - \\ & - \sum_{k=m+1}^{n} \vec{w}_i \vec{A}_1^n(\sigma)\frac{V_{kn}(\vec{a})L_k(\sigma)}{\Lambda_k(t)} - \\ & + \sum_{\ell,q=m+1}^{n} \vec{w}_i \frac{\partial^2 A_0(\sigma)}{\partial\sigma^2}\vec{x}\frac{V_{\ell n}(\vec{a})L_\ell(\sigma)}{\Lambda_\ell(t)}\frac{V_{qn}(\vec{a})L_q(\sigma)}{\Lambda_q(t)} + \\ & + \sum_{k=0}^{n-1}\sum_{\ell,q=m+1}^{n} \vec{w}_i \frac{\partial \vec{A}_0^k}{\partial\sigma}\frac{V_{\ell n}(\vec{a})L_\ell(\sigma)}{\Lambda_\ell(t)}\frac{V_{qk}(\vec{a})L_q(\sigma)}{\Lambda_q(t)} + \\ & + \sum_{j=0}^{n-1}\sum_{\ell,q=m+1}^{n} \vec{w}_i \frac{\partial \vec{A}_0^j}{\partial\sigma}\frac{V_{\ell j}(\vec{a})L_\ell(\sigma)}{\Lambda_\ell(t)}\frac{V_{qn}(\vec{a})L_q(\sigma)}{\Lambda_q(t)}\Bigg\} + \mathcal{O}(\varepsilon^2). \end{aligned} \tag{2.95}$$

According to the equation (2.76) that defines the value of the n-th entry of eigenvector the V_{jk} is $V_{jn} = 0$. The other entries of the eigenvectors correspond to the components of eigenvectors of the A_0 i.e. $V_{kj} = v_{kj}$ for $j = 1, \dots, n-1$. Using those values, we obtain

$$\frac{\mathrm{d}a_i}{\mathrm{d}t} = L_i(\sigma) - \varepsilon\sum_{j=0}^{n-1}\sum_{k=m+1}^{n} \vec{w}_i \vec{A}_1^j(\sigma)\frac{v_{kj}(\vec{a})L_k(\sigma)}{\lambda_k(t)} + \mathcal{O}(\varepsilon^2). \tag{2.96}$$

2.2 Numerical Integration Methods

2.2.1 Order of Approximation

The analytical solution of the systems of ordinary differential equations (ODE) is are always possible, in those cases we can approximate the result using numerical methods.

A generic form of ODE can is given as

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{f}(t, x(t)) \tag{2.97}$$

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

$$\vec{f}(t,\vec{x}) = \sum_{j=0}^{\infty}\frac{\vec{f}^{(j)}(t_0,\vec{x}_0)}{j!}\Delta t^j \tag{2.98}$$

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

$$\vec{f}^{(j)}(t_0, \vec{x}_0) = \left.\frac{\mathrm{d}^j \vec{f}}{\mathrm{d}t^j}\right|_{t=t_0}. \tag{2.99}$$

It can be noticed that each consecutive term of the series becomes smaller, due to the division of a large factorial of j. The series can be written as

$$\vec{f}(t, \vec{x}) = \sum_{j=0}^{k-1} \frac{\vec{f}^{(j)}(t_0, \vec{x}_0)}{j!} \Delta t^j + \mathcal{O}(\Delta t^k) \tag{2.100}$$

where $\mathcal{O}(\Delta t^k)$ are terms of an order k. Ignoring the $\mathcal{O}(\Delta t^k)$ we obtain an approximation of order k.

2.2.2 Explicit Methods

Forward Euler

The simplest explicit method is the forward Euler method, which is derived by solving the first order approximation of Taylor series of the system (2.97) which gives an iterative scheme

$$\vec{x}_{n+1} = \vec{x}_n + \vec{f}(t_n, \vec{x}_n)\Delta t, \qquad \vec{x}_0 = x_0(t_0). \tag{2.101}$$

where the numerical solution $\vec{x}_{n+1} \approx \vec{x}(t_{n+1})$ converges to the exact solution as $\Delta t \to 0$. The expense of the higher accuracy resulting from the reduction of the time step size is the higher demand on the computation, which requires time inversely proportional to the chosen time step Δt.

Runge-Kutta Methods

Runge-Kutta have developed a whole family of explicit methods. The most popular is the fourth order method which is given as

$$\vec{x}_{n+1} = \vec{x}_n + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{2.102}$$

where

$$k_1 = \vec{f}(t_n + \tfrac{\Delta t}{2}, \vec{x}_n), \tag{2.103a}$$
$$k_2 = \vec{f}(t_n + \tfrac{\Delta t}{2}, \vec{x}_n + \tfrac{\Delta t}{2} k_1), \tag{2.103b}$$
$$k_3 = \vec{f}(t_n + \tfrac{\Delta t}{2}, \vec{x}_n + \tfrac{\Delta t}{2} k_2), \tag{2.103c}$$
$$k_4 = \vec{f}(t_n + \Delta t, \vec{x}_n + \Delta t k_3). \tag{2.103d}$$

2.2.3 Rush-Larsen Technique for a Gate Model

The dynamical equations describing the evolution of a gate model are often the most stiff parts of the cellular model. For that reason a very small time step is required to conserve numerical stability. The Rush-Larsen technique is a semi-implicit algorithm for efficient solution of the gate model[23]. The gating variables are described by equation in the form

$$\frac{\mathrm{d}w}{\mathrm{d}t} = \alpha(\mathrm{V_m}(t))(1-w) - \beta(\mathrm{V_m}(t))w \tag{2.104}$$

where α, β are transition rates. 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

$$\alpha(\mathrm{V_m}(t)) \approx \alpha(\mathrm{V_m}(t_n)) = \alpha_n, \tag{2.105a}$$

$$\beta(\mathrm{V_m}(t)) \approx \beta(\mathrm{V_m}(t_n)) = \beta_n. \tag{2.105b}$$

Then the equation (2.104) gives a solution

$$w(t) = \frac{\alpha_n}{\alpha_n+\beta_n} - \left(\frac{\alpha_n}{\alpha_n+\beta_n} - w(t_n)\right)\exp\Big(-(t-t_n)(\alpha_n+\beta_n)\Big) \tag{2.106}$$

that is more convenient to write in a form

$$w_{n+1} = w_{\infty,n} - (w_{\infty,n} - w_n)\exp\left(-\frac{\Delta t}{\tau_{w,n}}\right) \tag{2.107}$$

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

$$w_{\infty,n} = \frac{\alpha_n}{\alpha_n+\beta_n} \tag{2.108}$$

is the steady state solution which is obtained from (2.104) by setting the $\mathrm{d}w/\mathrm{d}t = 0$, and

$$\tau_{w,n} = \frac{1}{\alpha_n+\beta_n} \tag{2.109}$$

is the time constant $\tau_{w,n}$.

The Rush-Larsen algorithm[23] adjust the time step size according to the rate of change of the other variables, e.g. $\mathrm{V_m}$. The algorithm monitors the value the slope of the potential $\mathrm{dV_m}/\Delta t$ and set the step size (Δt) to reduce the computational cost. During the stimulus or if $\mathrm{dV_m}/\Delta t > 5$ mV/ms the $\Delta t = 0.01$ ms, otherwise the step $\Delta t = 0.01 \cdot [5/(\mathrm{dV_m}/\Delta t)]$ but never more than 1 ms.

2.3 Numerical Methods for Markov Chain

2.3.1 Order of Approximation

We recall that the Markov chain is described by a system of equations in a form

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}(t)\vec{u}(t), \tag{2.110}$$

where the entries of $\vec{u}$ represent the states occupancies and the $\boldsymbol{A}(t)$ is a transition rates matrix. The exact solution is given as follows

$$u(t_{n+1}) = \exp\left(\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}(t')\mathrm{d}t'\right) u(t_n). \tag{2.111}$$

The truncation error is defined as a difference between the exact and the numerical solution. First, expanding the integrand in the exponential of the the exact solution (2.111) according to the Taylor expansion around t_n we get

$$\boldsymbol{A}(t')\Big|_{t'=t_n} = \boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}t + \mathcal{O}(t^2), \tag{2.112}$$

where $\boldsymbol{A}_n = \boldsymbol{A}(t_n)$ and

$$\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t} = \frac{\mathrm{d}\boldsymbol{A}}{\mathrm{d}t}\Big|_{t'=t_n}. \tag{2.113}$$

Then, we substitute the expanded version back to the expression (2.111) and get

$$\begin{aligned} u(t_{n+1}) &= \exp\left(\int_{t_n}^{t_n+\Delta t} \boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}t' + \mathcal{O}(t^2)\mathrm{d}t'\right) u(t_n) = \\ &= \exp\left(\boldsymbol{A}_n\Delta t + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t^2 + \mathcal{O}(\Delta t^3)\right) u(t_n) \\ &= g(t_n, \Delta t)u(t_n) + \mathcal{O}(\Delta t^3), \end{aligned} \tag{2.114}$$

where we expand the exponential

$$g(t_n, \Delta t) = \exp\left(\boldsymbol{A}_n\Delta t + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t^2\right) \tag{2.115}$$

around the point $\Delta t = 0$

$$g(t_n, \Delta t)\Big|_{\Delta t=0} = g(t_n, 0) + \frac{\mathrm{d}g}{\mathrm{d}\Delta t}\Delta t + \frac{1}{2}\frac{\mathrm{d}^2 g}{\mathrm{d}\Delta t^2}\Delta t^2 + \mathcal{O}(\Delta t^3). \tag{2.116}$$

So, first we find the derivatives

$$\frac{\mathrm{d}g}{\mathrm{d}\Delta t} = \left(\boldsymbol{A}_n + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t\right)\exp\left(\boldsymbol{A}_n\Delta t + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t^2\right), \tag{2.117}$$

$$\frac{\mathrm{d}^2 g}{\mathrm{d}\Delta t^2} = \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\exp\left(\boldsymbol{A}_n\Delta t + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t^2\right) + \boldsymbol{A}_n^2\exp\left(\boldsymbol{A}_n\Delta t + \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\Delta t^2\right) + \mathcal{O}(\Delta t), \tag{2.118}$$

which for $\Delta t = 0$ read as

$$\frac{\mathrm{d}g}{\mathrm{d}\Delta t} = \boldsymbol{A}_n, \tag{2.119}$$

$$\frac{\mathrm{d}^2 g}{\mathrm{d}\Delta t^2} = \frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t} + \boldsymbol{A}_n^2. \tag{2.120}$$

Then the (2.116) can be rewritten as

$$g(t_n) = 1 + \boldsymbol{A}_n\Delta t + \frac{1}{2}\left[\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t} + \boldsymbol{A}_n^2\right]\Delta t^2 + \mathcal{O}(\Delta t^3), \tag{2.121}$$

so the expression for the exact solution (2.114) yields

$$u(t_{n+1}) = \left(1 + \boldsymbol{A}_n\Delta t + \frac{1}{2}\left(\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t} + \boldsymbol{A}_n^2\right)\Delta t^2\right)u(t_n) + \mathcal{O}(\Delta t^3). \tag{2.122}$$

2.3.2 Forward Euler method

The solution of the system (2.110) according to the forward Euler method is

$$u_{\mathrm{FE},n+1} = (1 + \boldsymbol{A}_n\Delta t)u(t_n). \tag{2.123}$$

Comparing this value with (2.122) we obtain a local truncation error of forward Euler as

$$E_{\mathrm{FE}} = \|u(t_{n+1}) - u_{\mathrm{FE},n+1}\| = \frac{1}{2}\Delta t^2\left(\left\|\boldsymbol{A}^2\right\| + \frac{\mathrm{dV_m}}{\mathrm{d}t}\left\|\frac{\mathrm{d}\boldsymbol{A}}{\mathrm{dV_m}}\right\|\right) + \mathcal{O}(\Delta t^3) \tag{2.124}$$

where $\|\cdot\|$ denotes the norm of a matrix.

2.3.3 Matrix Rush-Larsen

The Rush-Larsen method is not directly applicable for the Markov chain models. Here we generalise the same idea for a Markov chain models. For a duration of a time step Δt, the transition rates matrix is approximated by "freezing" the coefficients to the values at the beginning of the time step as $\boldsymbol{A}(t_n)$ as

$$\vec{u}_{\mathrm{MRL},n+1} = \exp\left(\boldsymbol{A}(t_n)\Delta t\right)\vec{u}(t_n), \tag{2.125}$$

which is expanded using Taylor expansion to get

$$u_{\mathrm{MRL},n+1} = \left[1 + \boldsymbol{A}_n \Delta t + \frac{1}{2}\boldsymbol{A}_n^2 \Delta t^2\right] u(t_n) + \mathcal{O}(\Delta t^3). \tag{2.126}$$

Comparing this result with (2.122) we find the truncation error

$$E_{\mathrm{MRL}} = \|u(t_{n+1}) - u_{\mathrm{MRL},n+1}\| = \frac{1}{2}\left\|\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{d}t}\right\|\Delta t^2 + \mathcal{O}(\Delta t^3). \tag{2.127}$$

If $\boldsymbol{A}_n = \boldsymbol{A}(\mathrm{V_m}(t_n))$ the chain rule for the derivation the $\mathrm{dV_m}/\mathrm{d}t$ we can rewrite this expression in a form

$$E_{\mathrm{MRL}} = \frac{1}{2}\Delta t^2 \frac{\mathrm{dV_m}}{\mathrm{d}t}\left\|\frac{\mathrm{d}\boldsymbol{A}_n}{\mathrm{dV_m}}\right\| + \mathcal{O}(\Delta t^3). \tag{2.128}$$

So, the difference in error of forward Euler from the MRL method is

$$E_{\mathrm{FE}} - E_{\mathrm{MRL}} = \frac{1}{2}\Delta t^2 \left\|\boldsymbol{A}^2\right\| + \mathcal{O}(\Delta t^3). \tag{2.129}$$

For practical reasons the (2.125) is written in the form

$$\vec{u}_{\mathrm{MRL},n+1} = \boldsymbol{T}(t_n)\vec{u}(t_n) = \boldsymbol{V}(t_n)\exp\left(\boldsymbol{\Lambda}(t_n)\Delta t\right)\boldsymbol{W}^{\boldsymbol{T}}(t_n)\vec{u}(t_n), \tag{2.130}$$

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 $\boldsymbol{\Lambda}$.

2.3.4 Hybrid Methods

The hybrid methods use the operator splitting technique for the transition rates matrix of the Markov chain as

$$\boldsymbol{A} = \boldsymbol{A}_1 + \boldsymbol{A}_2 + \ldots + \boldsymbol{A}_k, \tag{2.131}$$

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 (2.110) becomes

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}\vec{u} = (\boldsymbol{A}_1 + \boldsymbol{A}_1 + \ldots + \boldsymbol{A}_k)\,\vec{u}, \tag{2.132}$$

which has an exact solution that gives us the following iterative scheme

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t + \boldsymbol{A}_2\Delta t + \ldots + \boldsymbol{A}_k\Delta t\right)\vec{u}_n. \tag{2.133}$$

However, sometimes the exact solution is not convenient, so we split the solution as

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right)\ldots\exp\left(\boldsymbol{A}_k\Delta t\right)\vec{u}_n, \tag{2.134}$$

which rewrites as

$$\vec{u}_{n+1/k} = \exp\left(\boldsymbol{A}_1\Delta t\right)\vec{u}_n, \tag{2.135a}$$
$$\vec{u}_{n+2/k} = \exp\left(\boldsymbol{A}_2\Delta t\right)\vec{u}_{n+1/k}, \tag{2.135b}$$
$$\vdots$$
$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_k\Delta t\right)\vec{u}_{n+(k-1)/k}. \tag{2.135c}$$

This scheme allows us to substitute specific steps by an alternative methods, if the exact solution is too costly. Hence, we call this methods hybrid.

2.3.5 Lie Splitting

Double Splitting

Lie splitting is the simplest method of operator splitting. Here we analyse the error in splitting of the operator matrix into two, so that $k = 1$ in (2.131). This section is inspired by [24].

The exponential in the exact solution in this case is

$$\begin{aligned}\exp\Big((\boldsymbol{A}_1 + \boldsymbol{A}_2)\Delta t\Big) &= 1 + \Delta t\left(\boldsymbol{A}_1 + \boldsymbol{A}_2\right) + \frac{\Delta t^2}{2}\Big(\boldsymbol{A}_1^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_2^2\Big) + \\ &+ \frac{\Delta t^3}{6}({\boldsymbol{A}_1}^3 + {\boldsymbol{A}_1}^2\boldsymbol{A}_2 + \boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_1{\boldsymbol{A}_2}^2 + \boldsymbol{A}_2{\boldsymbol{A}_1}^2 + \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{A}_2 + {\boldsymbol{A}_2}^2\boldsymbol{A}_1 + {\boldsymbol{A}_2}^3) + \\ &+ \mathcal{O}(\Delta t^4).\end{aligned} \tag{2.136}$$

The operator splitting method rewrites as

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right)\vec{u}_n. \tag{2.137}$$

Each exponentials expands according to

$$\exp(\boldsymbol{A}_j\Delta t) = 1 + \Delta t\boldsymbol{A}_j + \frac{\Delta t^2}{2}\boldsymbol{A}_j^2 + \mathcal{O}(\Delta t^3), \tag{2.138}$$

and we multiply two exponentials for $j = 0, 1$ to get

$$\left(1 + \Delta t \boldsymbol{A}_1 + \frac{\Delta t^2}{2}\boldsymbol{A}_1^2\right)\left(1 + \Delta t \boldsymbol{A}_2 + \frac{\Delta t^2}{2}\boldsymbol{A}_2^2\right) + \mathcal{O}(\Delta t^3) =$$
$$=1 + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}(\boldsymbol{A}_1^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2) + \mathcal{O}(\Delta t^3) \tag{2.139}$$

the local truncation error is found after subtracting the (2.139) from (2.136), which by orders of Δt gives

$$\mathcal{O}(\Delta t^0) : 1 - 1 = 0, \tag{2.140a}$$
$$\mathcal{O}(\Delta t^1) : (\boldsymbol{A}_1 + \boldsymbol{A}_2) - (\boldsymbol{A}_1 + \boldsymbol{A}_2) = 0, \tag{2.140b}$$
$$\mathcal{O}(\Delta t^2) : \frac{1}{2}\Big((\boldsymbol{A}_1^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_2^2) - (\boldsymbol{A}_1^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2)\Big) =$$
$$= \frac{1}{2}(\boldsymbol{A}_2\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_2). \tag{2.140c}$$

So, that the error of double splitting is

$$E_{\text{OS},2} = \frac{1}{2}\Delta t^2 \|[\boldsymbol{A}_2, \boldsymbol{A}_1]\| + \mathcal{O}(\Delta t^3) \tag{2.141}$$

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 error in splitting of the operator matrix into three, so that $k = 2$ in (2.131). The exponential in the exact solution in this case is

$$\exp\Big((\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3)\Delta t\Big) = 1 + \Delta t\,(\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) +$$
$$+ \frac{\Delta t^2}{2}\Big(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2 + \boldsymbol{A}_3^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_1\boldsymbol{A}_3 + \boldsymbol{A}_3\boldsymbol{A}_1 + \boldsymbol{A}_2\boldsymbol{A}_3 + \boldsymbol{A}_3\boldsymbol{A}_2\Big)$$
$$+ \mathcal{O}(\Delta t^3). \tag{2.142}$$

The operator splitting method rewrites as

$$\vec{u}_{n+1} = \exp(\boldsymbol{A}_1\Delta t)\exp(\boldsymbol{A}_2\Delta t)\exp(\boldsymbol{A}_3\Delta t)\,\vec{u}_n \tag{2.143}$$

we multiply all three terms expanded as (2.138) and using (2.139) to get

$$\left(1 + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}(\boldsymbol{A}_1^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2^2)\right)\left(1 + \Delta t\boldsymbol{A}_3 + \frac{\Delta t^2}{2}\boldsymbol{A}_3^2\right) + \mathcal{O}(\Delta t^3) =$$
$$= 1 + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) + \Delta t^2(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2 + \boldsymbol{A}_3^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + 2\boldsymbol{A}_1\boldsymbol{A}_3 + 2\boldsymbol{A}_2\boldsymbol{A}_3) \tag{2.144}$$

And the local truncation error is then

$$\mathcal{O}(\Delta t^0) : 1 - 1 = 0 \tag{2.145}$$

$$\mathcal{O}(\Delta t^1) : (\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) - (\boldsymbol{A}_1 + \boldsymbol{A}_2 + \boldsymbol{A}_3) = 0 \tag{2.146}$$

$$\begin{aligned}\mathcal{O}(\Delta t^2) : &\frac{1}{2}\Big[(\boldsymbol{A}_1^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_1\boldsymbol{A}_3 + \boldsymbol{A}_2\boldsymbol{A}_1 + \boldsymbol{A}_2^2 + \boldsymbol{A}_2\boldsymbol{A}_3 + \boldsymbol{A}_1\boldsymbol{A}_3 + \boldsymbol{A}_2\boldsymbol{A}_3 + \boldsymbol{A}_3^2)\\ &-(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2 + \boldsymbol{A}_3^2 + 2\boldsymbol{A}_1\boldsymbol{A}_2 + 2\boldsymbol{A}_1\boldsymbol{A}_3 + 2\boldsymbol{A}_2\boldsymbol{A}_3)\Big] =\\ &= \frac{1}{2}[(\boldsymbol{A}_2\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_2) + (\boldsymbol{A}_3\boldsymbol{A}_1 - \boldsymbol{A}_1\boldsymbol{A}_3) + (\boldsymbol{A}_3\boldsymbol{A}_2 - \boldsymbol{A}_2\boldsymbol{A}_3)].\end{aligned} \tag{2.147}$$

So, we find the error of triple splitting as

$$E_{\mathrm{OS},3} = \frac{1}{2}\Delta t^2 \left(\|[\boldsymbol{A}_2, \boldsymbol{A}_1] + [\boldsymbol{A}_3, \boldsymbol{A}_1] + [\boldsymbol{A}_3, \boldsymbol{A}_2]\|\right) + \mathcal{O}(\Delta t^3). \tag{2.148}$$

2.3.6 Higher Order Accuracy Operator Splitting

Strang splitting

The procedure of strang splitting[24] is

$$\vec{u}_{n+1} = \exp\left(\frac{1}{2}\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right)\exp\left(\frac{1}{2}\boldsymbol{A}_1\Delta t\right)\vec{u}_n, \tag{2.149}$$

where we expanded in power series the first and last term as

$$\exp\left(\frac{1}{2}\boldsymbol{A}_1\Delta t\right) = 1 + \frac{\Delta t}{2}\boldsymbol{A}_1 + \frac{\Delta t^2}{8}{\boldsymbol{A}_1}^2 + \frac{\Delta t^3}{48}{\boldsymbol{A}_1}^3 + \mathcal{O}(\Delta t^4), \tag{2.150}$$

and the expansion of the middle term is identical to (2.138). Then we can multiply all three terms of (2.149)

$$\begin{aligned}&\left[1 + \frac{\Delta t}{2}\boldsymbol{A}_1 + \frac{\Delta t^2}{8}{\boldsymbol{A}_1}^2 + \frac{\Delta t^3}{48}{\boldsymbol{A}_1}^3 + \mathcal{O}(\Delta t^4)\right]\left[1 + \Delta t\boldsymbol{A}_2 + \frac{\Delta t^2}{2}{\boldsymbol{A}_2}^2 + \frac{\Delta t^3}{6}{\boldsymbol{A}_2}^3 + \mathcal{O}(\Delta t^4)\right]\\ &\left[1 + \frac{\Delta t}{2}\boldsymbol{A}_1 + \frac{\Delta t^2}{8}{\boldsymbol{A}_1}^2 + \frac{\Delta t^3}{48}{\boldsymbol{A}_1}^3 + \mathcal{O}(\Delta t^4)\right] =\\ &= 1 + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}({\boldsymbol{A}_1}^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + {\boldsymbol{A}_2}^2) +\\ &+ \frac{\Delta t^3}{24}(4{\boldsymbol{A}_1}^3 + 3{\boldsymbol{A}_1}^2\boldsymbol{A}_2 + 6\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_1 + 6\boldsymbol{A}_1{\boldsymbol{A}_2}^2 + 3\boldsymbol{A}_2{\boldsymbol{A}_1}^2 + 6{\boldsymbol{A}_2}^2\boldsymbol{A}_1 + 4{\boldsymbol{A}_2}^3) + \mathcal{O}(\Delta t^4)\end{aligned} \tag{2.151}$$

and find the local truncation error by subtracting the (2.136) from (2.151) as

$$E_{\mathrm{strang}} = -\frac{1}{24}\Delta t^3\Big(\|[\boldsymbol{A}_1, [\boldsymbol{A}_1, \boldsymbol{A}_2]] - 2[\boldsymbol{A}_2, [\boldsymbol{A}_1, \boldsymbol{A}_2]]\|\Big) + \mathcal{O}(\Delta t^4). \tag{2.152}$$

SWSS splitting

The procedure of SWSS splitting[24] is

$$\vec{u}_{n+1} = \frac{1}{2}\left[\exp\left(\boldsymbol{A}_1\Delta t\right)\exp\left(\boldsymbol{A}_2\Delta t\right) + \exp\left(\boldsymbol{A}_2\Delta t\right)\exp\left(\boldsymbol{A}_1\Delta t\right)\right]\vec{u}_n, \tag{2.153}$$

which is expanded according to (2.138) and multiplied to get

$$\begin{aligned}
&\frac{1}{2}[\exp(\boldsymbol{A}_1\Delta t)\exp(\boldsymbol{A}_2\Delta t) + \exp(\boldsymbol{A}_2\Delta t)\exp(\boldsymbol{A}_1\Delta t)] = \\
=&1 + \Delta t(\boldsymbol{A}_1 + \boldsymbol{A}_2) + \frac{\Delta t^2}{2}({\boldsymbol{A}_1}^2 + \boldsymbol{A}_1\boldsymbol{A}_2 + \boldsymbol{A}_2\boldsymbol{A}_1 + {\boldsymbol{A}_2}^2) + \\
&+ \frac{\Delta t^3}{12}(2{\boldsymbol{A}_1}^3 + 3{\boldsymbol{A}_1}^2\boldsymbol{A}_2 + 3\boldsymbol{A}_1{\boldsymbol{A}_2}^2 + 3{\boldsymbol{A}_2}^2\boldsymbol{A}_1 + 3\boldsymbol{A}_2{\boldsymbol{A}_1}^2 + 2{\boldsymbol{A}_2}^3) + \mathcal{O}(\Delta t^4).
\end{aligned} \tag{2.154}$$

The local truncation error is by comparing (2.154) with (2.136), which gives the error of SWSS operator splitting as

$$E_{\text{SWSS}} = \frac{1}{12}\Delta t^3\Big(\|([\boldsymbol{A}_1, [\boldsymbol{A}_1, \boldsymbol{A}_2]] - [\boldsymbol{A}_2, [\boldsymbol{A}_1, \boldsymbol{A}_2]])\|\Big) + \mathcal{O}(\Delta t^4). \tag{2.155}$$

Chapter 3

Dimensionality Reduction of I_{Na} Markov Chain

3.1 Analysis of I_{Na} Markov Chain

3.1.1 Formulation of I_{Na} Markov Chain

The Markov chain I_{Na} published by Clancy and Rudy[1] was introduced in subsection 1.3.1. For convenience we reformulate the terminology of the states and transition rates. The new states are denoted by single letters O to W. The new state O represent the same state as in the old system. The following states are marked in the alphabetical order in clock-wise direction (Figure 3.1(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 terminology is

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{UO}U - (\alpha_{OP} + \alpha_{OU})O, \tag{3.1a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{UP}U + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PU} + \alpha_{PO})P, \tag{3.1b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{3.1c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{3.1d}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, \tag{3.1e}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, \tag{3.1f}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - (\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV})U, \tag{3.1g}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{3.1h}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.1i}$$

This notation is more convenient, because, we can readily check that

- Positive transition α_{jk} have a corresponding negative transition α_{kj} in the same equation,
- A certain transition appearing in one equation will appear also in another one with negative sign (so the conservation law is satisfied), and
- The transition rates (e.g. α_{jk}) multiplies a state corresponding to the first index (here state j).

The transition rates are redefined according to the new notation as

$$\alpha_{RQ} = \alpha_{11}, \qquad \alpha_{QR} = \beta_{11}, \tag{3.2a}$$
$$\alpha_{QP} = \alpha_{12}, \qquad \alpha_{PQ} = \beta_{12}, \tag{3.2b}$$
$$\alpha_{PO} = \alpha_{13}, \qquad \alpha_{OP} = \beta_{13}, \tag{3.2c}$$

$$\alpha_{ST} = \alpha_{11}, \qquad \alpha_{TS} = \beta_{11}, \tag{3.2d}$$
$$\alpha_{TU} = \alpha_{12}, \qquad \alpha_{UT} = \beta_{12}, \tag{3.2e}$$

$$\alpha_{UP} = \alpha_{3}, \qquad \alpha_{PU} = \beta_{3}, \tag{3.2f}$$
$$\alpha_{TQ} = \alpha_{3}, \qquad \alpha_{QT} = \beta_{3}, \tag{3.2g}$$
$$\alpha_{SR} = \alpha_{3}, \qquad \alpha_{RS} = \beta_{3}, \tag{3.2h}$$

$$\alpha_{OU} = \alpha_{2}, \qquad \alpha_{UO} = \beta_{2}, \tag{3.2i}$$

$$\alpha_{UV} = \alpha_{4}, \qquad \alpha_{VU} = \beta_{4}, \tag{3.2j}$$
$$\alpha_{VW} = \alpha_{5}, \qquad \alpha_{WV} = \beta_{5}. \tag{3.2k}$$

Notice that in the old system some transition rates were present at more than one place. Using the new notation each transition rate presents specific transition between two states. Therefore number of transition rates increases, although some are defined by the same formulas.

The Figure 3.1(a) shows the dependence of the transition rates of the system on the membrane voltage. The transition rates can be divided according to the speeds to fast transition rates (top panel) and slow transition rates (bottom panel).

The sum of transition rates in both directions determines the probability of transition in between two states given that the state occupancy of both states is identical. The Figure 3.1 shows the dependency of the speed of transition rates as

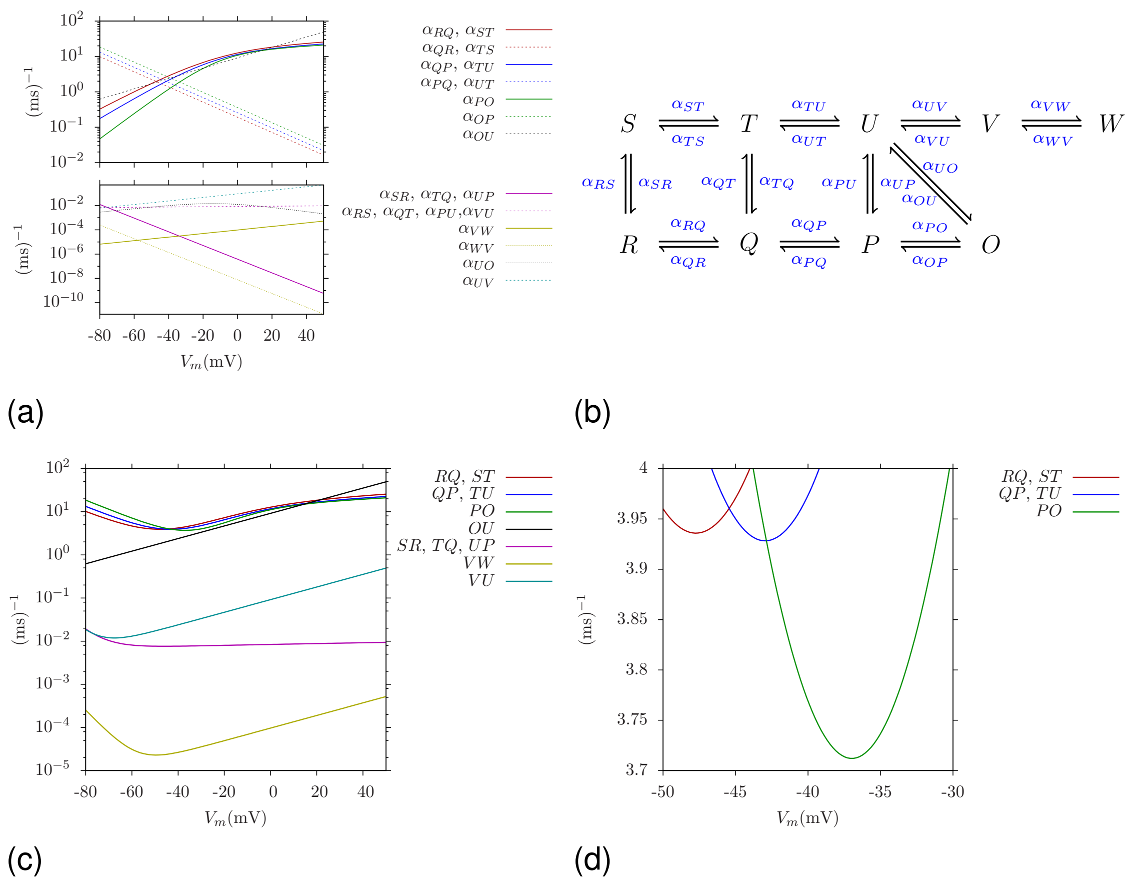


Figure 3.1: Characteristics of I_{Na} Markov chain model: (a) individual transition rates for the range of membrane potentials (identical colours are used to show the pair of transition rates between two states according to the legend on the right from the panels); (d) a diagram of the model (with new terminology); (c,d) sum of pairs of transition rates between two states for the range of membrane potentials (as denoted in the legend on the right from the panels).

a function of membrane voltage – panel (c) shows dependency for the whole range of the membrane potential, panel (d) shows the detail of the fastest pairs of the transition rates in their minimum. In our system the fastest transitions are found between states RQ,ST, QP, TU, PO. This transition rates will be considered for the embedding.

To analyse the dynamic properties of the system we show the eigenvalues and eigenvectors of the transition rates matrix at fixed value of membrane voltage $\mathrm{V_m} = -30$ mV on Figure 3.2. We see, that all the eigenvalues are non-positive with one zero eigenvalue, which confirms that the solution is bounded. The zero eigenvalue appears due to the conservation law, which is observed in Markov chains.

3.1.2 Embeddings of I_{Na} Markov chain

To gain insight into which states are the best candidates for the reduction, we perform a transition rates embedding. Transition rates embedding is obtained by

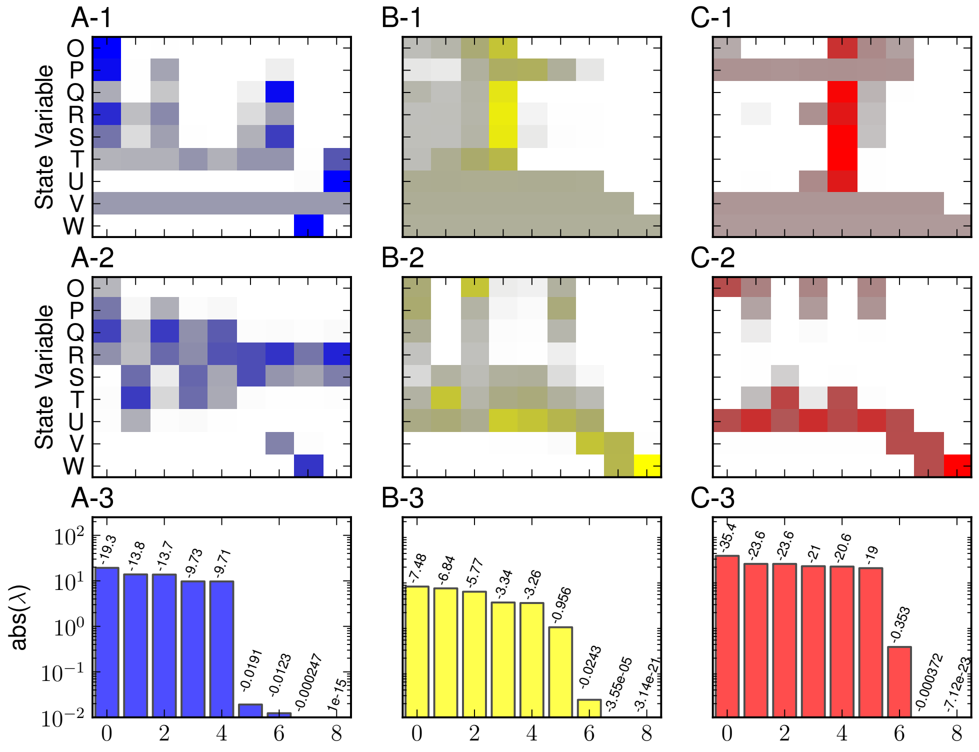


Figure 3.2: Eigenvalues and eigenvectors of I_{Na} model. First two rows show the entries in the matrices of left (1st row) and right (2nd row) eigenvectors, and the bottom row shows the absolute value of the eigenvalues (3rd row) in the I_{Na} model under constant voltage of $\mathrm{V_m} = -80$ mV – 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 correspond to the eigenvalue.

multiplication of specific transition rates by $1/\varepsilon$, where $\varepsilon \to 0$ is a small parameter. For practical purposes we use $\varepsilon = 0.1$ which means, that the corresponding transition rates are speed up tenfold.

Details of the complete procedure to perform the transition rates embedding is given in the subsection 3.2.1. Here we show the results of a larger number of embeddings done according to the same procedure, only with a different combinations of transition rates.

The Figure 3.3 shows the states occupancy under action potential. The action potential from model of the whole cell (Clancy, Rudy 2002) was recorded and the extracted I_{Na} model was driven by the recorded values of membrane voltage. 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 (RQ, ST, QP, TU, PO, and OU). As can be seen from the figure, all transition rates embeddings provide a good approximation of original model after

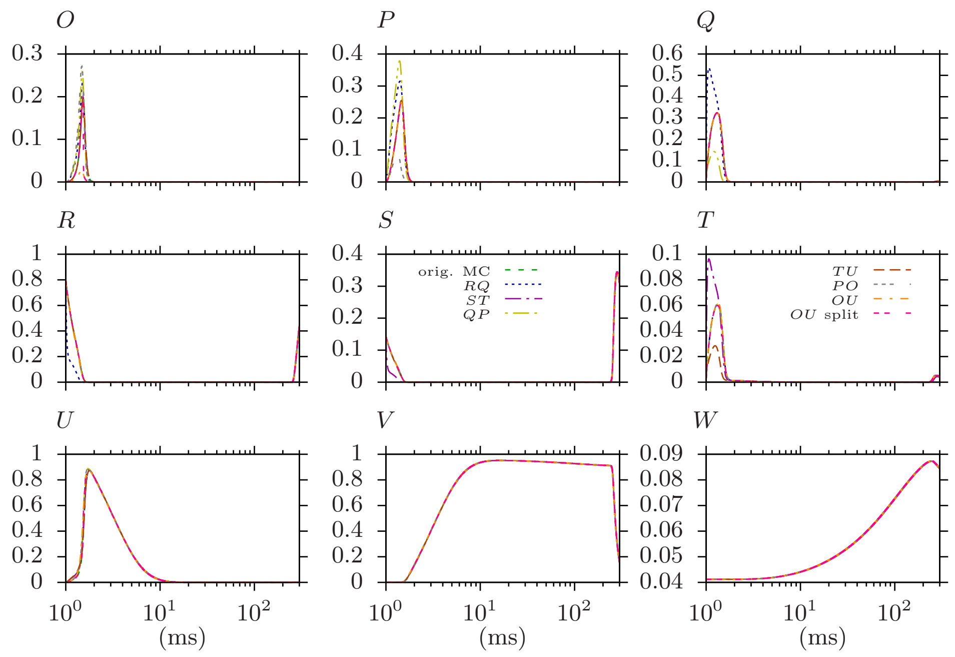


Figure 3.3: State occupancy of I_{Na} under action potential with one pair transition rates embeddings. From top right to bottom left the panels correspond to the occupancy of state O to W. All panels shows the same embeddings as denoted in the legend (in the panel S, T). The horizontal axis denotes time (in milliseconds) shown in logarithmic scale.

an initial transient, which is however the most important for the action potential onset, and where the I_{Na} channel plays a mayor role.

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 O state. 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 Figure 3.4 shows the first millisecond of the states occupancy under the action potential. This figure provides the 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 embedding.

The detailed view shows the embeddings PO, RQ, QP and OU important deviation from the original model in the state O. The embedding TU and ST visually overlap with the occupancy of the state O. Those embedding affect some of the closed states, but this should not have an effect on the overall performance of the model.

To further reduce the system we perform embeddings of two pairs of transition rates ($ST + TU$, $RQ + QP$, $ST + RQ$, $TU + QP$). The Figure 3.5 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 except the states T and U. The embeddings $ST + RQ$ and $TU + QP$ provide similar results in most states except S and T states.

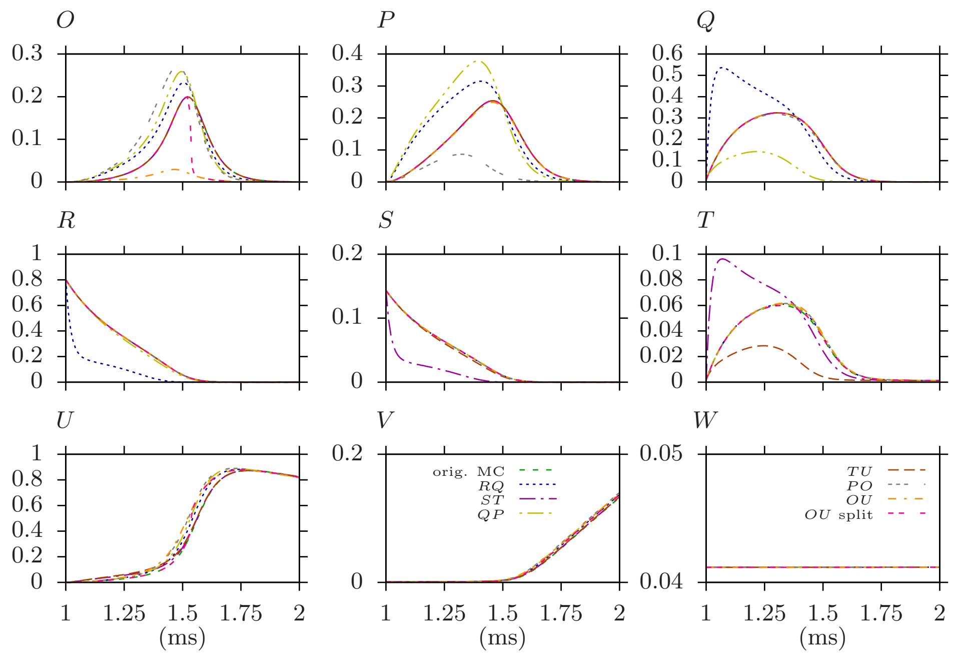


Figure 3.4: State occupancy under action potential with two pairs transition rates embedding: detail of the first millisecond. From top right to bottom left the panels correspond to the occupancy of state O to W. All panels shows the same embeddings as denoted in the legend (in the panel V, W).

However, both of them overshoot the open probability O. The embedding $RQ+QP$ is the least accurate embedding from the combinations of two-pairs of transition rates embeddings shown. It is also possible to perform other combinations of embeddings, however the ones in this figure are the most accurate.

To further reduce the system we do embedding of more than two pairs of transition rates as shown on Figure 3.6. The best result is achieved by the reduction of the transition rates between states $ST + TU + RQ$.

3.2 First Order OP-Reduction in I_{Na} Markov Chain

3.2.1 Embedding of OP States

In this section we apply the perturbation theory to the channel I_{Na}. The considered embedding includes transition rates $\alpha_{PO}(t)$ and $\alpha_{OP}(t)$ of the system of equations (3.1). Both transition rates are multiplied by $1/\varepsilon$ to give the embedded system, that reads as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{PO}(t)P + \alpha_{UO}(t)U - \left(\frac{1}{\varepsilon}\alpha_{OP}(t) + \alpha_{OU}(t)\right)O, \tag{3.3a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}(t)Q + \frac{1}{\varepsilon}\alpha_{OP}(t)O + \alpha_{UP}(t)U - \left(\frac{1}{\varepsilon}\alpha_{PO}(t) + \alpha_{PU}(t) + \alpha_{PQ}(t)\right)P, \tag{3.3b}$$

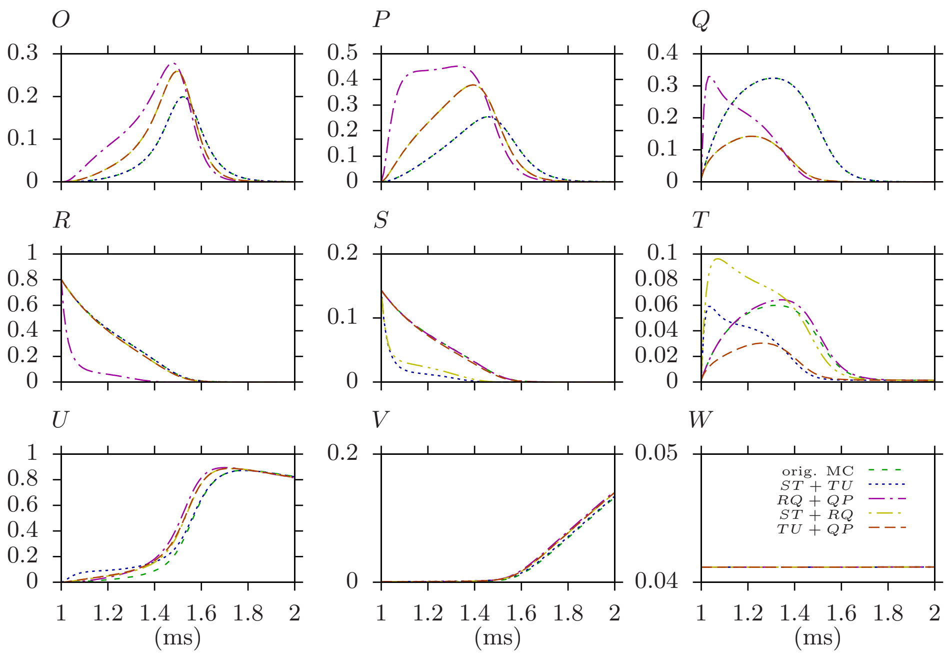


Figure 3.5: State occupancy under action potential with one pairs transition rates embedding: detail of the first millisecond. From top right to bottom left the panels correspond to the occupancy of state O to W. All panels shows the same embeddings as denoted in the legend (in panel W).

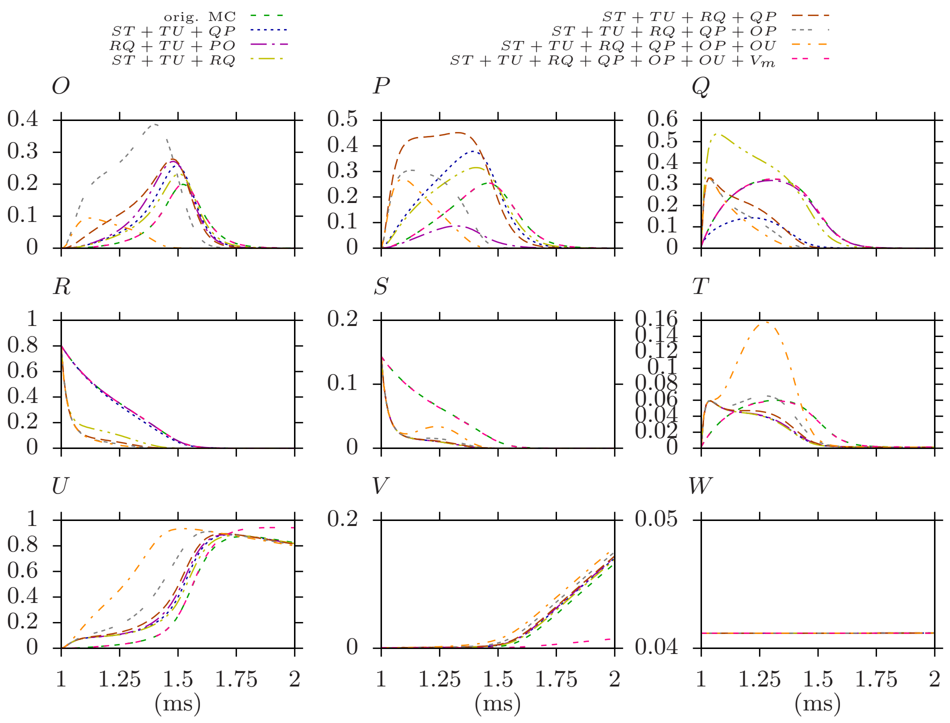


Figure 3.6: State occupancy under action potential with three pairs transition rates embedding: detail of the first millisecond. From top right to bottom left the panels correspond to the occupancy of state O to W. All panels shows the same embeddings as denoted in the legend (above the panels).

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{3.3c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{3.3d}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, \tag{3.3e}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, \tag{3.3f}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - (\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV})U, \tag{3.3g}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{3.3h}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.3i}$$

This system can be reformulated according to (2.1) where

$$\vec{x}(t) = \begin{bmatrix} O(t) \\ P(t) \end{bmatrix}, \tag{3.4a}$$

$$\vec{H}(t) = \begin{bmatrix} \alpha_{UO}(t)U(t) \\ \alpha_{QP}(t)Q(t) + \alpha_{UP}(t)U(t) \end{bmatrix}, \tag{3.4b}$$

$$\boldsymbol{A}(t) = \frac{1}{\varepsilon}\left(\boldsymbol{A}_0(t) + \varepsilon \boldsymbol{A}_1(t)\right) \tag{3.4c}$$

$$\boldsymbol{A}_0(t) = \frac{1}{\varepsilon}\begin{bmatrix} -\alpha_{OP}(t) & \alpha_{PO}(t) \\ \alpha_{OP}(t) & -\alpha_{PO}(t) \end{bmatrix}, \tag{3.4d}$$

$$\boldsymbol{A}_1(t) = \begin{bmatrix} -\alpha_{OU}(t) & 0 \\ 0 & -(\alpha_{PU}(t) + \alpha_{PQ}(t)) \end{bmatrix}. \tag{3.4e}$$

For convenience we only write a subsystem corresponding to states O and P.

3.2.2 Choice of Eigenvectors

To obtain the solution in form (2.32) 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 identify the adjoint vector $\vec{\omega}_2(t)$ of the eigenvector $u_2(t)$. Which yields

$$\lambda_1(t) = 0,$$

$$\lambda_2(t) = -(\alpha_{PO}(t) + \alpha_{OP}(t)) \tag{3.5a}$$

$$\vec{u}_1(t) = \begin{bmatrix} \alpha_{PO}(t) \\ \alpha_{OP}(t) \end{bmatrix},$$

$$\vec{u}_2(t) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \tag{3.5b}$$

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

$$\vec{\omega}_2(t) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} -\alpha_{OP}(t) \\ \alpha_{PO}(t) \end{bmatrix}, \tag{3.5c}$$

$$\tilde{\boldsymbol{P}}(t) = \begin{bmatrix} \alpha_{PO}(t) & -1 \\ \alpha_{OP}(t) & 1 \end{bmatrix},$$

$$\tilde{\boldsymbol{P}}^{-1}(t) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} 1 & 1 \\ -\alpha_{OP}(t) & \alpha_{PO}(t) \end{bmatrix}. \tag{3.5d}$$

The eigenvectors $\vec{u}_k(t)$ have to be normalised to satisfy the requirement of diagonal orthogonality. The normalisation coefficients were found according to (2.16) as

$$\begin{aligned} s_1(t) &= \exp\left(-\int (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} 1 & 1 \end{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t}\left(\begin{bmatrix} \alpha_{PO}(t) \\ \alpha_{OP}(t) \end{bmatrix}\right) \mathrm{d}t\right) \\ &= \exp\left(-\ln(\alpha_{PO} + \alpha_{OP})\right) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1}, \end{aligned} \tag{3.6a}$$

$$s_2(t) = \exp\left(-\int (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} -\alpha_{OP}(t) & \alpha_{PO}(t) \end{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t}\left(\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) \mathrm{d}t\right) = 1. \tag{3.6b}$$

Using the values of s_1 and s_2 and multiplying with the trial solution yields a new eigenvectors' matrix and its inverse as

$$\boldsymbol{P}(t) = \begin{bmatrix} \alpha_{PO}(t)(\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} & -1 \\ \alpha_{OP}(t)(\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} & 1 \end{bmatrix}, \tag{3.7a}$$

$$\boldsymbol{P}^{-1}(t) = \begin{bmatrix} 1 & 1 \\ -\alpha_{OP}(t)(\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} & \alpha_{PO}(t)(\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \end{bmatrix}. \tag{3.7b}$$

So the new eigenvector and its adjoint vector are

$$\vec{v}_1(t) = \begin{bmatrix} \alpha_{PO}(t)(\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \\ \alpha_{OP}(t)(\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \end{bmatrix}, \tag{3.8a}$$

$$\vec{v}_2(t) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \tag{3.8b}$$

$$\vec{w}_1(t) = \begin{bmatrix} 1 & 1 \end{bmatrix}, \tag{3.8c}$$

$$\vec{w}_2(t) = (\alpha_{PO}(t) + \alpha_{OP}(t))^{-1} \begin{bmatrix} -\alpha_{OP}(t) & \alpha_{PO}(t) \end{bmatrix}, \tag{3.8d}$$

$$\lambda_1 = 0, \tag{3.8e}$$

$$\lambda_2(t) = -(\alpha_{PO}(t) + \alpha_{OP}(t)). \tag{3.8f}$$

3.2.3 Reduction of States O and P to One State N

We reformulate the general equation (2.32) and (2.33) specifically for our system. Using a new state variable $N(t) = a_0(t)$ we get

$$\frac{\mathrm{d}N}{\mathrm{d}t} = N(t)\vec{w}_1(t)\boldsymbol{A}_1(t)\vec{v}_1(t) + \vec{w}_1(t)\vec{H}(t), \tag{3.9a}$$

$$O(t) = \frac{\alpha_{PO}(t)}{\alpha_{PO}(t) + \alpha_{OP}(t)}N(t), \tag{3.9b}$$

$$P(t) = \frac{\alpha_{OP}(t)}{\alpha_{PO}(t) + \alpha_{OP}(t)}N(t), \tag{3.9c}$$

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

Having obtained normalised eigenvalues and eigenvectors we can expand both non-homogeneous term and coefficient of $N(t)$ in (3.9a)

$$\vec{w}_1(t)\vec{H}(t) = (\alpha_{UO}(t) + \alpha_{UP}(t))U(t) + \alpha_{QP}(t)Q(t) \tag{3.10a}$$

$$\vec{w}_1(t)\boldsymbol{A}_1(t)\vec{v}_1(t) = \frac{\alpha_{OU}(t)\alpha_{PO}(t) + \alpha_{PQ}(t)\alpha_{OP}(t) + \alpha_{PU}(t)\alpha_{OP}(t)}{\alpha_{PO}(t) + \alpha_{OP}(t)} \tag{3.10b}$$

The differential equation for the new variable variable is

$$\frac{\mathrm{d}N}{\mathrm{d}t} = (\alpha_{UP} + \alpha_{UO})U + \alpha_{QP}Q - \left(\frac{\alpha_{OU}\alpha_{PO} + \alpha_{PU}\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}} + \frac{\alpha_{PQ}\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}}\right)N. \tag{3.11}$$

In the subsystem (3.1) the equations (3.1c), (3.1g) are dependent on states $O(t)$ and $P(t)$ which can be reformulated using (3.9b) and (3.9c) as

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \frac{\alpha_{PQ}\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{3.12a}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{VU}V + \frac{\alpha_{OU}\alpha_{PO} + \alpha_{PU}\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}}N - (\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV})U. \tag{3.12b}$$

We define new transition rates

$$\alpha_{UN} = \alpha_{UP} + \alpha_{UO}, \tag{3.13a}$$

$$\alpha_{QN} = \alpha_{QP}, \tag{3.13b}$$

$$\alpha_{NU} = \frac{\alpha_{OU}\alpha_{PO} + \alpha_{PU}\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}}, \tag{3.13c}$$

$$\alpha_{NQ} = \frac{\alpha_{PQ}\alpha_{OP}}{\alpha_{PO} + \alpha_{OP}}, \tag{3.13d}$$

which are substituted into equations (3.12b), (3.12b) and (3.11) to give

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \alpha_{UN}U + \alpha_{QN}Q - (\alpha_{NU} + \alpha_{NQ})N, \tag{3.14a}$$

$$\begin{tikzcd}
S \arrow[r, shift left, "\alpha_{ST}"] \arrow[d, shift left, "\alpha_{SR}"] & T \arrow[l, shift left, "\alpha_{TS}"] \arrow[r, shift left, "\alpha_{TU}"] \arrow[d, shift left, "\alpha_{TQ}"] & U \arrow[l, shift left, "\alpha_{UT}"] \arrow[r, shift left, "\alpha_{UV}"] \arrow[d, shift left, "\alpha_{UN}"] & V \arrow[l, shift left, "\alpha_{VU}"] \arrow[r, shift left, "\alpha_{VW}"] & W \arrow[l, shift left, "\alpha_{WV}"] \\
R \arrow[u, shift left, "\alpha_{RS}"] \arrow[r, shift left, "\alpha_{RQ}"] & Q \arrow[u, shift left, "\alpha_{QT}"] \arrow[l, shift left, "\alpha_{QR}"] \arrow[r, shift left, "\alpha_{QN}"] & N \arrow[u, shift left, "\alpha_{NU}"] \arrow[l, shift left, "\alpha_{NQ}"]
\end{tikzcd}$$

Figure 3.7: Diagram of the I_{Na} Markov chain with OP-reduction by state N.

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{NQ}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{3.14b}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{VU}V + \alpha_{NU}N - (\alpha_{UT} + \alpha_{UN} + \alpha_{UV})U. \tag{3.14c}$$

The equation (3.14a) substitute both equations (3.1a) and (3.1b). The equations (3.14b), (3.14c) replace (3.1c), (3.1g) respectively. The reduced model is then described by the following system of equations

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \alpha_{UN}U + \alpha_{QN}Q - (\alpha_{NU} + \alpha_{NQ})N, \tag{3.15a}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{NQ}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QN})Q, \tag{3.15b}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{3.15c}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, \tag{3.15d}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, \tag{3.15e}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{NU}N + \alpha_{VU}V - (\alpha_{UT} + \alpha_{UN} + \alpha_{UV})U, \tag{3.15f}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{3.15g}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.15h}$$

The diagram of th model is shown on figure Figure 3.7

The Figure 3.8 shows the first 2 ms of simulated states occupancies of Markov chain I_{Na} driven by the membrane voltage obtained from simulation of Clancy (2002) model[1]. The reduced model shows the same qualitative behaviour as the corresponding embedding. The biggest 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 those states, only a minimal deviation is observed. This is because the transition rates from of both O and P to U are slow, moreover the discrepancy from O is compensated by the opposite discrepancy caused by P.

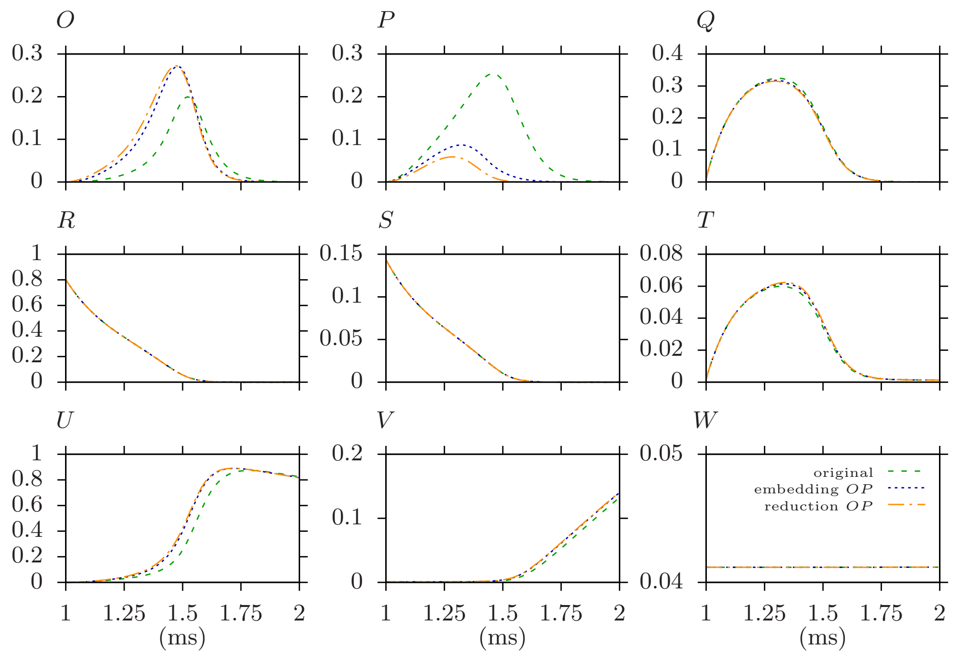


Figure 3.8: States occupancy of I_{Na} Markov chain under action potential onset. Green lines show original model, blue lines show embedding of states O, P, magenta lines show leading order approximation of the reduced model.

3.3 Correction Term for OP-Reduction of I_{Na}

3.3.1 Autonomisation of the System

So far we have found the necessary variables of the matrix $\boldsymbol{A}_0$. This system is non-autonomous due to its dependence on the independent variable t. To apply the perturbation theory with the correction term we need to reformulate the system to autonomous as described in section 2.1.3.

The subsystem (3.4) with variables as described on (3.3) can be extended by a new variable σ linked to time as specified by (2.67). The autonomous system has a form given by equation (2.34), which is obtained according to equation (2.69) as

$$\vec{f}(\vec{u}) = \begin{bmatrix} \boldsymbol{A}_0(\sigma)\vec{x}(\sigma) \\ 0 \end{bmatrix} = \begin{bmatrix} -\alpha_{OP}(\sigma)O^* + \alpha_{PO}(\sigma)P^* \\ \alpha_{OP}(\sigma)O^* - \alpha_{PO}(\sigma)P^* \\ 0 \end{bmatrix}, \tag{3.16a}$$

$$\vec{h}(\sigma) = \begin{bmatrix} \boldsymbol{A}_1(\sigma)\vec{x}(\sigma) + H(\sigma) \\ 1 \end{bmatrix} =$$
$$= \begin{bmatrix} -\alpha_{OU}(\sigma)O^* + \alpha_{UO}(\sigma)U(\sigma) \\ -(\alpha_{PU}(\sigma) + \alpha_{PQ}(\sigma))P^* + \alpha_{QP}(\sigma)Q(\sigma) + \alpha_{UP}(\sigma)U(\sigma) \\ 1 \end{bmatrix}, \tag{3.16b}$$

$$\vec{u}(\sigma) = \begin{bmatrix} \vec{x}(\sigma) \\ \sigma \end{bmatrix} = \begin{bmatrix} O^* \\ P^* \\ \sigma \end{bmatrix}. \tag{3.16c}$$

3.3.2 Finding Eigenvectors and Eigenvalues

Using the autonomous system we find eigenvalues Λ_k and corresponding eigenvectors $\vec{V}_k$ to the $\boldsymbol{F}(\vec{a}(\sigma))$. The eigenvalues Λ_k correspond to the eigenvalues λ_k of the matrix A_0 for $k = 1, \ldots, n$ where $n = 2$ for our case. The eigenvectors $\vec{V}_k$ are found as described (2.76) as

$$\vec{V}_1 = \begin{bmatrix} \vec{v}_1(\sigma) \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_{PO}(\sigma)(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^{-1} \\ \alpha_{OP}(\sigma)(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^{-1} \\ 0 \end{bmatrix}, \tag{3.17a}$$

$$\Lambda_1 = \lambda_1 = 0 \tag{3.17b}$$

$$\vec{V}_2 = \begin{bmatrix} \vec{v}_2(\sigma) \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \tag{3.17c}$$

$$\Lambda_2 = \lambda_2 = -(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma)). \tag{3.17d}$$

The dimensions of the new system is $n+1$ due to the new variable σ. Therefore, the Jacobian $\boldsymbol{F}$ will be extended by one dimension. We assign index zero to the eigenvalue and eigenvector corresponding to the new dimension. We find that $\Lambda_0 = 0$, and eigenvector is found according to (2.82) as

$$\vec{V}_0 = \begin{bmatrix} a_1 \frac{\mathrm{d}\vec{v}_1(\sigma)}{\mathrm{d}\sigma} \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 \left[\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma) - \frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}\alpha_{PO}(\sigma)\right] (\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^{-2} \\ -a_1 \left[\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma) - \frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}\alpha_{PO}(\sigma)\right] (\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^{-2} \\ 1 \end{bmatrix}, \tag{3.18}$$

where using the rules for the differentiation of the product we get

$$\frac{\mathrm{d}v_{11}}{\mathrm{d}\sigma} = \frac{\mathrm{d}}{\mathrm{d}\sigma}\left(\frac{\alpha_{PO}(\sigma)}{\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma)}\right) = -\frac{\mathrm{d}}{\mathrm{d}\sigma}\left(\frac{\alpha_{OP}(\sigma)}{\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma)}\right) = -\frac{\mathrm{d}v_{12}}{\mathrm{d}\sigma}, \tag{3.19}$$

so the vector

$$\frac{\mathrm{d}\vec{v}_1}{\mathrm{d}\sigma} = \frac{\mathrm{d}v_{11}}{\mathrm{d}\sigma} \begin{bmatrix} 1 \\ -1 \end{bmatrix}. \tag{3.20}$$

Finally, we find adjoint vectors $\vec{W^T}_k$ and $\vec{W^T}_0$ as described in (2.83) as

$$\vec{W^T}_0 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, \tag{3.21a}$$

$$\vec{W^T}_1 = \begin{bmatrix}\vec{w}_1 & -a_1\vec{w}_1\frac{\mathrm{d}\vec{v}_1}{\mathrm{d}\sigma}\end{bmatrix} = \begin{bmatrix}1 & 1 & 0\end{bmatrix}, \tag{3.21b}$$

$$\begin{aligned}\vec{W^T}_2 &= \begin{bmatrix}\vec{w}_2 & -a_1\vec{w}_2\frac{\mathrm{d}\vec{v}_1}{\mathrm{d}\sigma}\end{bmatrix} = \\ &= \begin{bmatrix} -\alpha_{OP}(\sigma)(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^{-1} \\ \alpha_{PO}(\sigma)(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^{-1} \\ a_1\left(\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma)-\alpha_{PO}(\sigma)\frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}\right)(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^{-2}\end{bmatrix}^T.\end{aligned} \tag{3.21c}$$

3.3.3 Solution of OP-Reduction with a Correction Term

The general equation for Markov chain model (2.96) was developed in the section 2.1.3. By substituting the variables found for our system we obtain

$$\frac{\mathrm{d}a_1}{\mathrm{d}t} = L_1(\sigma) + \varepsilon\left\{L_2(\sigma)\frac{\alpha_{OU}(\sigma)-(\alpha_{PU}(\sigma)+\alpha_{PQ}(\sigma))}{\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma)}\right\} + \mathcal{O}(\varepsilon^2), \tag{3.22}$$

where

$$\begin{aligned}L_1(\sigma) = &-\frac{\alpha_{PO}\alpha_{OU}(\sigma)+\alpha_{OP}(\alpha_{PU}(\sigma)+\alpha_{PQ}(\sigma))}{\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma)}N(\sigma)+\alpha_{QP}(\sigma)Q(\sigma)+ \\ &+(\alpha_{UO}(\sigma)+\alpha_{UP}(\sigma))U(\sigma),\end{aligned} \tag{3.23}$$

$$\begin{aligned}L_2(\sigma) = &\frac{\alpha_{OP}(\sigma)\alpha_{OU}(\sigma)\alpha_{PO}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma)-\frac{\alpha_{OP}\alpha_{PO}(\sigma)(\alpha_{PU}(\sigma)+\alpha_{PQ}(\sigma))}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma) \\ &+\frac{\alpha_{PO}(\sigma)\alpha_{QP}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))}Q(\sigma)+\frac{\alpha_{PO}(\sigma)\alpha_{UP}(\sigma)-\alpha_{OP}(\sigma)\alpha_{UO}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))}U(\sigma)+ \\ &+\frac{\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma)-\alpha_{PO}(\sigma)\frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma) = \\ =&\frac{\alpha_{OP}\alpha_{PO}(\sigma)(\alpha_{OU}(\sigma)-\alpha_{PU}(\sigma)-\alpha_{PQ}(\sigma))}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma)+ \\ &+\frac{\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma)-\alpha_{PO}(\sigma)\frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma)+\frac{\alpha_{PO}(\sigma)\alpha_{QP}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))}Q(\sigma)+ \\ &+\frac{\alpha_{PO}(\sigma)\alpha_{UP}(\sigma)-\alpha_{OP}(\sigma)\alpha_{UO}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))}U(\sigma).\end{aligned} \tag{3.24}$$

Substituting L_1 and L_2 into the relation (3.22) yields

$$\begin{aligned}\frac{\mathrm{d}N}{\mathrm{d}t} = &-\frac{\alpha_{PO}\alpha_{OU}(\sigma)+\alpha_{OP}(\alpha_{PU}(\sigma)+\alpha_{PQ}(\sigma))}{\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma)}N(\sigma)+\alpha_{QP}(\sigma)Q(\sigma)+ \\ &+(\alpha_{UO}(\sigma)+\alpha_{UP}(\sigma))U(\sigma)+ \\ &+\varepsilon\left\{\left[\frac{\alpha_{OP}\alpha_{PO}(\sigma)(\alpha_{OU}(\sigma)-\alpha_{PU}(\sigma)-\alpha_{PQ}(\sigma))}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma)+\right.\right. \\ &+\frac{\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma)-\alpha_{PO}(\sigma)\frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))^2}N(\sigma)+\frac{\alpha_{PO}(\sigma)\alpha_{QP}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))}Q(\sigma) \\ &\left.\left.+\frac{\alpha_{PO}(\sigma)\alpha_{UP}(\sigma)-\alpha_{OP}(\sigma)\alpha_{UO}(\sigma)}{(\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma))}U(\sigma)\right]\frac{\alpha_{OU}(\sigma)-(\alpha_{PU}(\sigma)+\alpha_{PQ}(\sigma))}{\alpha_{PO}(\sigma)+\alpha_{OP}(\sigma)}\right\}+\end{aligned}$$

$$+\mathcal{O}(\varepsilon^2). \tag{3.25}$$

The original states are obtained from the relation (2.71), which for our system gets the form $\vec{x} = a_1\vec{v}_1$. We substitute the eigenvector $\vec{v}_1$ from the equation (3.8a) and the coordinate on the slow manifold $N = a_1$ to obtain

$$O^* = \frac{\alpha_{PO}(\sigma)}{\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma)} N, \tag{3.26a}$$

$$P^* = \frac{\alpha_{OP}(\sigma)}{\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma)} N. \tag{3.26b}$$

We simplify the differential equation as

$$\begin{aligned}\frac{\mathrm{d}N}{\mathrm{d}t} =& \alpha_{UN}(\sigma)U(\sigma) + \alpha_{QN}(\sigma)Q(\sigma) - (\alpha_{NU}(\sigma) + \alpha_{NQ})\, N \\ &+ \varepsilon \left\{\alpha_{N2U}(\sigma)U(\sigma) + \alpha_{N2Q}(\sigma)Q(\sigma) + (\alpha_{N2N}(\sigma) + \alpha_{N2dN})N\right\} + \mathcal{O}(\varepsilon^2),\end{aligned} \tag{3.27}$$

which is identical with (3.11) in the $\mathcal{O}(\varepsilon)$ as expected. The transition rates $\alpha_{UN}(\sigma)$, $\alpha_{QN}(\sigma)$, $\alpha_{NU}(\sigma)$, $\alpha_{QN}(\sigma)$ are defined in (3.13) and new transition rates in $\mathcal{O}(\varepsilon)$ are

$$\alpha_{N2U} = -\left(\alpha_{OU}(\sigma) - \alpha_{PU}(\sigma) - \alpha_{PQ}(\sigma)\right)\frac{\alpha_{OP}(\sigma)\alpha_{UO}(\sigma) - \alpha_{PO}(\sigma)\alpha_{UP}(\sigma)}{(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^2}, \tag{3.28a}$$

$$\alpha_{N2Q} = (\alpha_{OU}(\sigma) - \alpha_{PU}(\sigma) - \alpha_{PQ}(\sigma))\frac{\alpha_{PO}(\sigma)\alpha_{QP}(\sigma)}{(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^2}, \tag{3.28b}$$

$$\alpha_{N2dN} = (\alpha_{OU}(\sigma) - \alpha_{PU}(\sigma) - \alpha_{PQ}(\sigma))\frac{\frac{\mathrm{d}\alpha_{PO}(\sigma)}{\mathrm{d}\sigma}\alpha_{OP}(\sigma) - \alpha_{PO}(\sigma)\frac{\mathrm{d}\alpha_{OP}(\sigma)}{\mathrm{d}\sigma}}{(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^3}, \tag{3.28c}$$

$$\alpha_{N2N} = \frac{\alpha_{PO}(\sigma)\alpha_{OP}(\sigma)(\alpha_{OU}(\sigma) - \alpha_{PU}(\sigma) - \alpha_{PQ}(\sigma))^2}{(\alpha_{PO}(\sigma) + \alpha_{OP}(\sigma))^2}. \tag{3.28d}$$

We calculate $\frac{\mathrm{d}\alpha_{OP}(\mathrm{V_m}(\sigma))}{\mathrm{d}\sigma}$ and $\frac{\mathrm{d}\alpha_{PO}(\mathrm{V_m}(\sigma))}{\mathrm{d}\sigma}$ by the chain rule derivative as

$$\frac{\mathrm{d}\alpha_{OP}(\mathrm{V_m}(\sigma))}{\mathrm{d}\sigma} = \frac{\partial\alpha_{OP}(\mathrm{V_m}(\sigma))}{\partial \mathrm{V_m}}\frac{\mathrm{dV_m}}{\mathrm{d}t}, \tag{3.29a}$$

$$\frac{\mathrm{d}\alpha_{PO}(\mathrm{V_m}(\sigma))}{\mathrm{d}\sigma} = \frac{\partial\alpha_{PO}(\mathrm{V_m}(\sigma))}{\partial \mathrm{V_m}}\frac{\mathrm{dV_m}}{\mathrm{d}t}. \tag{3.29b}$$

Because of the reformulation defined in (3.2c) the transition rates are $\alpha_{PO} = \alpha_{13}$ and $\alpha_{OP} = \beta_{13}$. Using the definition of α_{13} (1.40c) and β_{13} (1.40h) we find

$$\begin{aligned}\frac{\partial\alpha_{PO}}{\partial \mathrm{V_m}} =& \frac{\partial\alpha_{13}}{\partial \mathrm{V_m}} = \frac{\partial}{\partial \mathrm{V_m}}\left(\frac{3.802}{0.1027e^{-\mathrm{V_m}/12.0} + 0.25e^{-\mathrm{V_m}/150}}\right) = \\ =& \frac{3.802\left(0.1027/12.0e^{-\mathrm{V_m}/12.0} + 0.25/150e^{-\mathrm{V_m}/150}\right)}{(0.1027e^{-\mathrm{V_m}/12.0} + 0.25e^{-\mathrm{V_m}/150})^2},\end{aligned} \tag{3.30}$$

$$\frac{\partial \alpha_{OP}}{\partial \mathrm{V_m}} = \frac{\partial \beta_{13}}{\partial \mathrm{V_m}} = \frac{\partial}{\partial \mathrm{V_m}} \left(0.22 e^{-(\mathrm{V_m}-10)/20.3}\right) = -\frac{2.2}{203} e^{-(\mathrm{V_m}-10)/20.3}. \tag{3.31}$$

The development of differential equations for states Q and U can be followed in (3.12). The complete reduced model including correction term is as follows

$$\begin{aligned} \frac{\mathrm{d}N}{\mathrm{d}t} =& \alpha_{UN}U + \alpha_{QN}Q - (\alpha_{NU} + \alpha_{NQ})N + \\ & + \varepsilon \left\{\alpha_{N2U}(\sigma)U(\sigma) + \alpha_{N2Q}(\sigma)Q(\sigma) + (\alpha_{N2N}(\sigma) + \alpha_{N2dN})N\right\}, & \text{(3.32a)} \\ \frac{\mathrm{d}Q}{\mathrm{d}t} =& \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{NQ}N - (\alpha_{QR} + \alpha_{QT} + \alpha_{QN})Q, & \text{(3.32b)} \\ \frac{\mathrm{d}R}{\mathrm{d}t} =& \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, & \text{(3.32c)} \\ \frac{\mathrm{d}S}{\mathrm{d}t} =& \alpha_{TS}T + \alpha_{RS}R - (\alpha_{ST} + \alpha_{SR})S, & \text{(3.32d)} \\ \frac{\mathrm{d}T}{\mathrm{d}t} =& \alpha_{QT}Q + \alpha_{ST}S + \alpha_{UT}U - (\alpha_{TQ} + \alpha_{TS} + \alpha_{TU})T, & \text{(3.32e)} \\ \frac{\mathrm{d}U}{\mathrm{d}t} =& \alpha_{TU}T + \alpha_{NU}N + \alpha_{VU}V - (\alpha_{UT} + \alpha_{UN} + \alpha_{UV})U, & \text{(3.32f)} \\ \frac{\mathrm{d}V}{\mathrm{d}t} =& \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, & \text{(3.32g)} \\ \frac{\mathrm{d}W}{\mathrm{d}t} =& \alpha_{VW}V - \alpha_{WV}W, & \text{(3.32h)} \\ \frac{\mathrm{d}\sigma}{\mathrm{d}t} =& 1. & \text{(3.32i)} \end{aligned}$$

The embedded model (green lines on Figure 3.9) is identical with the original model for $\varepsilon = 1$ and approach to the reduced model as $\varepsilon \to 0$.

The state conservation law ($\sum X_i = 1$ for X_i states of the system) is not satisfied when the first order correction term $\mathcal{O}(\varepsilon)$ is used in reduced system as in the system (3.32). In our case the $\sum X_i \geq 1$ within 300 ms of simulation. The deviation from the state conservation law can be reduced by choosing a lower value of ε.

The Figure 3.9 shows simulation results when the state conservation law was imposed by normalising the state occupancies as

$$X_i = \frac{X_i}{\sum X_i}. \tag{3.33}$$

However, the normalisation has a major effect on the slow state W, which drifts from its value in the original model.

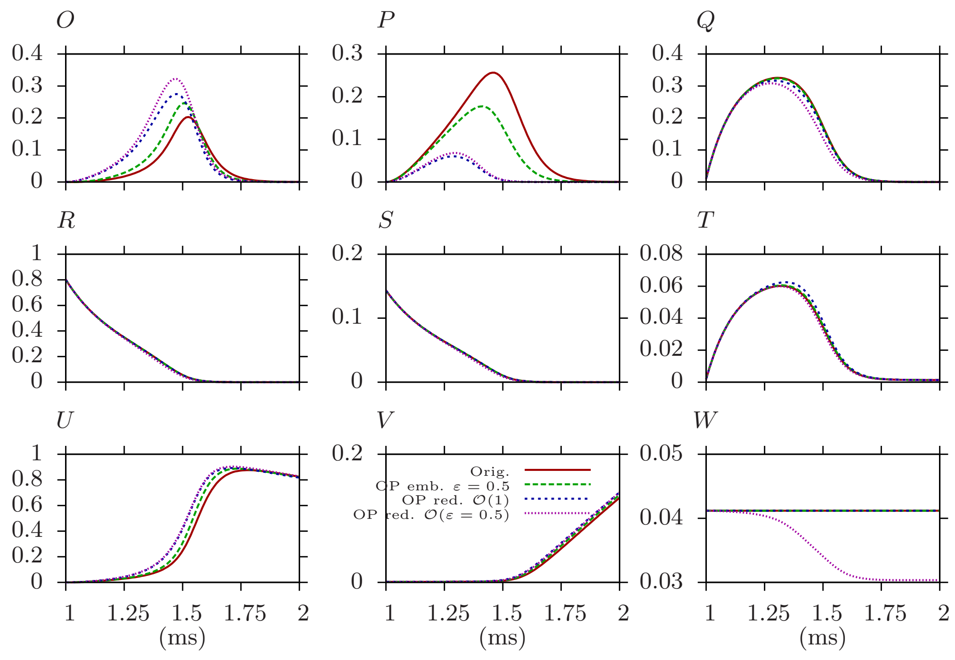


Figure 3.9: State occupancy under action potential (detail of the first millisecond). The red lines show original model (3.1), the green lines show embedded model (3.3) with the $\varepsilon = 0.5$, the blue lines show leading order reduction (3.15), magenta lines show first order $\mathcal{O}(\varepsilon)$ correction term with $\varepsilon = 0.5$. All models were extended as autonomous system. A normalisation was used to force the state conservation law in the reduced models.

3.4 First order STU-Reduction in I_{Na} Markov Chain

3.4.1 Embedding of STU States

Figure 3.5 shows that the embedding of states S, T and U provides a good approximation for the open states occupancy O, that governs the change of conductance of the I_{Na} channel. In this section we aim to reduce the states S, T, U with a new state M. Using the theory from the section 2.1.1.

First, we define the embedded system of states S, T, and U analogically to 3.2. This means multiply the transition rates between the states $T \leftrightarrow S$ and $S \leftrightarrow U$ in system of equations (3.1) by $1/\varepsilon$ (for $\varepsilon \to 0$). This yields the embedded system as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{UO}U - (\alpha_{OP} + \alpha_{OU})O, \tag{3.34a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{UP}U + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PU} + \alpha_{PO})P, \tag{3.34b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{TQ}T + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QT} + \alpha_{QP})Q, \tag{3.34c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S + \alpha_{QR}Q - (\alpha_{RS} + \alpha_{RQ})R, \tag{3.34d}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{TS}T + \alpha_{RS}R - \left(\frac{1}{\varepsilon}\alpha_{ST} + \alpha_{SR}\right)S, \tag{3.34e}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q + \frac{1}{\varepsilon}\alpha_{ST}S + \frac{1}{\varepsilon}\alpha_{UT}U - \left(\alpha_{TQ} + \frac{1}{\varepsilon}\alpha_{TS} + \frac{1}{\varepsilon}\alpha_{TU}\right)T, \tag{3.34f}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{TU}T + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - \left(\frac{1}{\varepsilon}\alpha_{UT} + \alpha_{UP} + \alpha_{UO} + \alpha_{UV}\right)U, \tag{3.34g}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{3.34h}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.34i}$$

Analogically, the equations (3.12) we expand the system for fast-slow time scale analysis. 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

$$\boldsymbol{A}_0 = \begin{bmatrix} -\alpha_{ST} & \alpha_{TS} & 0 \\ \alpha_{ST} & -(\alpha_{TS} + \alpha_{TU}) & \alpha_{UT} \\ 0 & \alpha_{TU} & -\alpha_{UT} \end{bmatrix} \tag{3.35}$$

and and the rest as

$$\boldsymbol{A}_1 = \begin{bmatrix} -\alpha_{SR} & 0 & 0 \\ 0 & -\alpha_{TQ} & 0 \\ 0 & 0 & -(\alpha_{UP} + \alpha_{UO} + \alpha_{UV}) \end{bmatrix}. \tag{3.36}$$

The non-homogeneous term and dynamical variables vector for this system are

$$\vec{x} = \begin{bmatrix} S \\ T \\ U \end{bmatrix} \qquad \vec{H} = \begin{bmatrix} \alpha_{RS}R \\ \alpha_{QT}Q \\ \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O \end{bmatrix} \tag{3.37}$$

For convenience we ignore the states without embedded transition rates.

3.4.2 Choice of Eigenvectors

First, we need to find eigenvalues and eigenvectors of matrix in (3.35). The roots of the determinant of its characteristic matrix $\det(\boldsymbol{A}_0 - \lambda\mathbf{I})$ (where $\mathbf{I}$ is identity matrix) are found

$$-\lambda^3 - \lambda^2(\alpha_{UT} + \alpha_{ST} + \alpha_{TU} + \alpha_{TS}) - \lambda(\alpha_{ST}\alpha_{UT} + \alpha_{TS}\alpha_{UT} + \alpha_{TU}\alpha_{ST}) = 0 \tag{3.38}$$

as expected one of the roots of the characteristic matrix is 0. Which gives us one zero eigenvalue $\lambda_1 = 0$. We find that the eigenvector corresponding to this eigenvalue is

$$\vec{u}_1 = \begin{bmatrix} \alpha_{UT}\alpha_{TS} \\ \alpha_{UT}\alpha_{ST} \\ \alpha_{TU}\alpha_{ST} \end{bmatrix}. \tag{3.39}$$

The adjoint vector $\vec{\omega}_1$ of the eigenvector $\vec{u}_1$ is

$$\vec{\omega}_1 = (\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU})^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{3.40}$$

The scaling factor (2.16) to satisfy the condition of dynamical orthogonality is

$$s_1 = (\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU})^{-1}, \tag{3.41}$$

which gives us the required eigenvectors

$$\vec{v}_1 = (\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU})^{-1} \begin{bmatrix} \alpha_{UT}\alpha_{TS} \\ \alpha_{UT}\alpha_{ST} \\ \alpha_{TU}\alpha_{ST} \end{bmatrix}, \qquad \vec{w}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{3.42}$$

The remaining eigenvalues are not required at this stage, as long as they satisfy the requirements for the solution to be bounded $\mathrm{Re}\{\lambda_{2,3}\} \leq 0$. To find eigenvalues $\lambda_{2,3}$ we find the remaining roots of the characteristic polynomial. We redefine (3.38) as $\lambda^2 + \beta\lambda + \gamma = 0$, where

$$\beta = \alpha_{UT} + \alpha_{ST} + \alpha_{TU} + \alpha_{TS}, \tag{3.43a}$$

$$\gamma = \alpha_{ST}\alpha_{UT} + \alpha_{TS}\alpha_{UT} + \alpha_{TU}\alpha_{ST}. \tag{3.43b}$$

The discriminant $\Delta = \beta^2 - 4\gamma$ can be used to find the solution as

$$\lambda_{2,3} = \frac{-\beta \pm \sqrt{\Delta}}{2}. \tag{3.44}$$

The requirement for the bounded solution gives us $\sqrt{\beta^2 - 4\gamma} < \beta$ which is true because the transition rates are always positive $\beta, \gamma > 0$.

3.4.3 Reduction of States S, T and U to One State M

We reduce the states S, T and U according to (2.32), where the eigenvalues are given in the equation (3.44) and the new state variable $a_0 = M = S + T + U$ is defined as

$$\begin{aligned}\frac{\mathrm{d}M}{\mathrm{d}t} =& \alpha_{RS}R + \alpha_{QT}Q + \alpha_{PU}P + \alpha_{VU}V + \alpha_{OU}O - \\ &- \frac{\alpha_{SR}\alpha_{UT}\alpha_{TS} + \alpha_{TQ}\alpha_{UT}\alpha_{ST} + (\alpha_{UP} + \alpha_{OU} + \alpha_{UV})\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M. \end{aligned} \tag{3.45}$$

The relation between the new and old states occupancy is found from the equation (2.33) as

$$S = \frac{\alpha_{UT}\alpha_{TS}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M, \tag{3.46a}$$

$$T = \frac{\alpha_{UT}\alpha_{ST}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M, \tag{3.46b}$$

$$U = \frac{\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}M. \tag{3.46c}$$

We define new transition rates

$$\alpha_{MS} = \frac{\alpha_{UT}\alpha_{TS}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47a}$$

$$\alpha_{MT} = \frac{\alpha_{UT}\alpha_{ST}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47b}$$

$$\alpha_{MU} = \frac{\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47c}$$

$$\alpha_{RM} = \alpha_{RS}, \tag{3.47d}$$

$$\alpha_{QM} = \alpha_{QT}, \tag{3.47e}$$

$$\alpha_{PM} = \alpha_{PU}, \tag{3.47f}$$

$$\alpha_{VM} = \alpha_{VU}, \tag{3.47g}$$

$$\alpha_{OM} = \alpha_{OU}, \tag{3.47h}$$

$$\alpha_{MO} = \frac{\alpha_{UO}\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47i}$$

$$\alpha_{MP} = \frac{\alpha_{UP}\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47j}$$

$$\alpha_{MQ} = \frac{\alpha_{TQ}\alpha_{UT}\alpha_{ST}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47k}$$

$$\alpha_{MR} = \frac{\alpha_{SR}\alpha_{UT}\alpha_{TS}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47l}$$

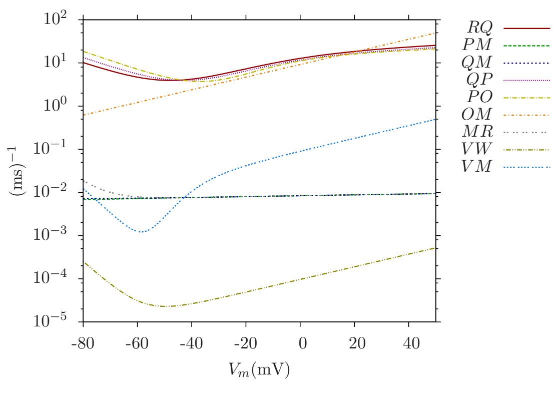
$$\alpha_{MV} = \frac{\alpha_{UV}\alpha_{ST}\alpha_{TU}}{\alpha_{UT}\alpha_{TS} + \alpha_{UT}\alpha_{ST} + \alpha_{ST}\alpha_{TU}}, \tag{3.47m}$$

which allows to reformulate the system of differential equations (3.34) as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{MO}M - (\alpha_{OP} + \alpha_{OM})O, \tag{3.48a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{MP}M + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PM} + \alpha_{PO})P, \tag{3.48b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R + \alpha_{MQ}M + \alpha_{PQ}P - (\alpha_{QR} + \alpha_{QM} + \alpha_{QP})Q, \tag{3.48c}$$



(a)

$$\begin{tikzcd}
R \arrow[r, shift left, "\alpha_{RM}"] \arrow[d, shift left, "\alpha_{RQ}"] & M \arrow[l, shift left, "\alpha_{MR}"] \arrow[r, shift left, "\alpha_{MV}"] \arrow[dl, shift left, "\alpha_{MQ}"] \arrow[d, shift left, "\alpha_{MP}"] \arrow[dr, shift left, "\alpha_{MO}"] & V \arrow[l, shift left, "\alpha_{VM}"] \arrow[r, shift left, "\alpha_{VW}"] & W \arrow[l, shift left, "\alpha_{WV}"] \\
Q \arrow[u, shift left, "\alpha_{QR}"] \arrow[ur, shift left, "\alpha_{QM}"] \arrow[r, shift left, "\alpha_{QP}"] & P \arrow[l, shift left, "\alpha_{PQ}"] \arrow[u, shift left, "\alpha_{PM}"] \arrow[r, shift left, "\alpha_{PO}"] & O \arrow[l, shift left, "\alpha_{OP}"] \arrow[ul, shift left, "\alpha_{OM}"] &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
 & M \arrow[dl, shift left, "\alpha_{ML}"] \arrow[d, shift left, "\alpha_{MP}"] \arrow[dr, shift left, "\alpha_{MO}"] \arrow[r, shift left, "\alpha_{MV}"] & V \arrow[l, shift left, "\alpha_{VM}"] \arrow[r, shift left, "\alpha_{VW}"] & W \arrow[l, shift left, "\alpha_{WV}"] \\
L \arrow[ur, shift left, "\alpha_{LM}"] \arrow[r, shift left, "\alpha_{LP}"] & P \arrow[l, shift left, "\alpha_{PL}"] \arrow[u, shift left, "\alpha_{PM}"] \arrow[r, shift left, "\alpha_{PO}"] & O \arrow[l, shift left, "\alpha_{OP}"] \arrow[ul, shift left, "\alpha_{OM}"] &
\end{tikzcd}$$

(c)

Figure 3.10: (a) Sum of the transition rates in the STU-reduced model as a function of membrane voltage (legend on the right of the panel), (b) diagram of the STU-reduction model, (c) diagram of RQ-reduction in STU-reduced model.

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{MR}M + \alpha_{QR}Q - (\alpha_{RM} + \alpha_{RQ})R, \tag{3.48d}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{MV}M + \alpha_{WV}W - (\alpha_{VM} + \alpha_{VW})V, \tag{3.48e}$$

$$\begin{aligned}\frac{\mathrm{d}M}{\mathrm{d}t} =& \alpha_{OM}O + \alpha_{PM}P + \alpha_{QM}Q + \alpha_{RM}R + \alpha_{VM}V- \\ & - (\alpha_{MO} + \alpha_{MP} + \alpha_{MQ} + \alpha_{MR} + \alpha_{MV})M,\end{aligned} \tag{3.48f}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.48g}$$

The diagram of the structure of the STU-reduced Markov chain model is shown on Figure 3.10(b). The Figure 3.11 shows the detail of the first millisecond of the simulation result of Markov chain I_{Na} channel driven by the membrane voltage recorded from simulation in original cellular model[1].

3.4.4 Correction Term in STU-Reduction

In this subsection we describe the second order correction term of the STU-reduction. The system was defined by the equations (3.34).

For the first order reduction we found the zero eigenvalue λ_0 and corresponding eigenvectors. We have also shown, that the remaining eigenvalues have negative real part, so that the solution is bounded. To find the correction term we get the remaining eigenvectors corresponding to the non-zero eigenvalues (3.44) as

$$\vec{u}_{2,3} = \alpha_{UT} \begin{bmatrix} \alpha_{TS} \\ \lambda_{2,3} + \alpha_{ST} \\ -(\lambda_{2,3} + \alpha_{ST} + \alpha_{TS}) \end{bmatrix}, \tag{3.49}$$

$$\vec{\omega}_{2,3} = \begin{bmatrix} \alpha_{ST}\alpha_{TU} & \alpha_{TU}(\lambda_{2,3} + \alpha_{ST}) & -\alpha_{UT}(\lambda_{2,3} + \alpha_{TS} + \alpha_{ST}) \end{bmatrix}. \tag{3.50}$$

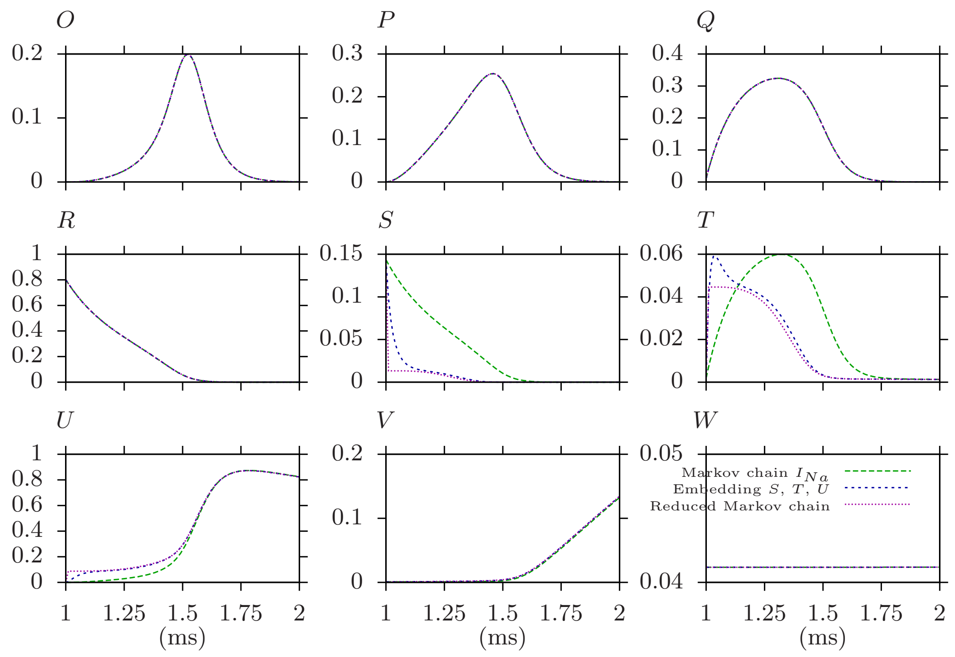


Figure 3.11: States occupancy in I_{Na} Markov chain under action potential onset. Green lines show original model, blue lines show embedding of states S, T, U with $\varepsilon = 0.1$, magenta lines show STU-reduced model. The panels shows occupancy of states from O to W (starting from the top right corner).

The left and right eigenvectors have to be scaled to satisfy the condition of dynamical orthogonality (2.13). To achive this we have to obtain a scaling factor s_k according to (2.16), which gives

$$s_k = \exp\left(-\int \omega_k \frac{\mathrm{d}u_k}{\mathrm{d}\sigma}\mathrm{d}t\right) = \exp\left(-\int K\mathrm{d}t\right), \tag{3.51}$$

for $k = 2, 3$. The integrand is obtained as

$$K = \frac{\mathrm{d}\alpha_{TS}}{\mathrm{d}\sigma}(\alpha_{ST}\alpha_{TU} + \alpha_{UT}x + \alpha_{UT}\alpha_{TS}) + \frac{\mathrm{d}x}{\mathrm{d}\sigma}(\alpha_{TU}x + \alpha_{UT}x + \alpha_{UT}\alpha_{TS}), \tag{3.52}$$

where $x = \lambda_k + \alpha_{ST}$.

The integral in the exponential in (3.51) can be solved only if we can find an integrating factor $\mathcal{I}(x, \alpha_{TS}, \alpha_{ST}, \alpha_{TU}, \alpha_{UT})$ such that

$$K\mathcal{I} = \frac{\mathrm{d}V}{\mathrm{d}t} = \frac{\partial V}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}\sigma} + \frac{\partial V}{\partial \alpha_{TS}}\frac{\mathrm{d}\alpha_{TS}}{\mathrm{d}\sigma} + \frac{\partial V}{\partial \alpha_{ST}}\frac{\mathrm{d}\alpha_{ST}}{\mathrm{d}\sigma} + \frac{\partial V}{\partial \alpha_{TU}}\frac{\mathrm{d}\alpha_{TU}}{\mathrm{d}\sigma} + \frac{\partial V}{\partial \alpha_{UT}}\frac{\mathrm{d}\alpha_{UT}}{\mathrm{d}\sigma} \tag{3.53}$$

where the partial derivatives are

$$\frac{\partial V}{\partial x} = (\alpha_{TU}x + \alpha_{UT}x + \alpha_{UT}\alpha_{TS})\mathcal{I}, \tag{3.54a}$$

$$\frac{\partial V}{\partial \alpha_{TS}} = (\alpha_{ST}\alpha_{TU} + \alpha_{UT}x + \alpha_{UT}\alpha_{TS})\mathcal{I}, \tag{3.54b}$$

$$\frac{\partial V}{\partial \alpha_{ST}} = 0, \tag{3.54c}$$
$$\frac{\partial V}{\partial \alpha_{TU}} = 0, \tag{3.54d}$$
$$\frac{\partial V}{\partial \alpha_{UT}} = 0. \tag{3.54e}$$

The requirement for the integral to be solvable is *(give reference)*

$$\frac{\partial^2 V}{\partial a_1 \partial a_2} = \frac{\partial^2 V}{\partial a_2 \partial a_1}, \qquad \forall a_1, a_2, \ldots, a_n \in x, \alpha_{TS}, \alpha_{ST}, \alpha_{TU}, \alpha_{UT}, \tag{3.55}$$

which yields the relation

$$\frac{\partial^2 V}{\partial x \partial \alpha_{ST}} = (\alpha_{TU} x + \alpha_{UT} x + \alpha_{UT}\alpha_{TS}) \frac{\partial \mathcal{I}}{\partial \alpha_{ST}}, \tag{3.56}$$
$$\frac{\partial^2 V}{\partial \alpha_{TS} \partial x} = \alpha_{UT}\mathcal{I} + (\alpha_{ST}\alpha_{TU} + \alpha_{UT} x + \alpha_{UT}\alpha_{TS}) \frac{\partial \mathcal{I}}{\partial x}, \tag{3.57}$$

that can not be satisfied in general, and therefore the analytic solution of (3.51) does not exist. This means that it is impossible to find analytic solution of eigenvectors, which satisfy the condition of dynamic orthogonality. Therefore, we can not obtain the solution for second order correction.

3.5 First Order RQ-Reduction in the STU-Reduced I_{Na} Model

3.5.1 Embedding of RQ States

The reduced model S, T, U shows a good approximation of the open state O which is needed to compute the conductance of the I_{Na} channel.

The Figure 3.12 shows the detail of the first millisecond of the states occupancy driven by the action potential. The embedding RQ of the reduced model and embedding ST, TU, RQ in original original model are affecting the same transition rates, but the state occupancy is different in some states (R, S, T, U). We find that the RQ-embedding visually overlaps with the open state O in the STU-reduced model. This makes is an ideal candidate for further reduction, so we define the embedding of the system (3.48) as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{MO}M - (\alpha_{OP} + \alpha_{OM})O, \tag{3.58a}$$
$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q + \alpha_{MP}M + \alpha_{OP}O - (\alpha_{PQ} + \alpha_{PM} + \alpha_{PO})P, \tag{3.58b}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \frac{1}{\varepsilon}\alpha_{RQ}R + \alpha_{MQ}M + \alpha_{PQ}P - \left(\frac{1}{\varepsilon}\alpha_{QR} + \alpha_{QM} + \alpha_{QP}\right) Q, \tag{3.58c}$$

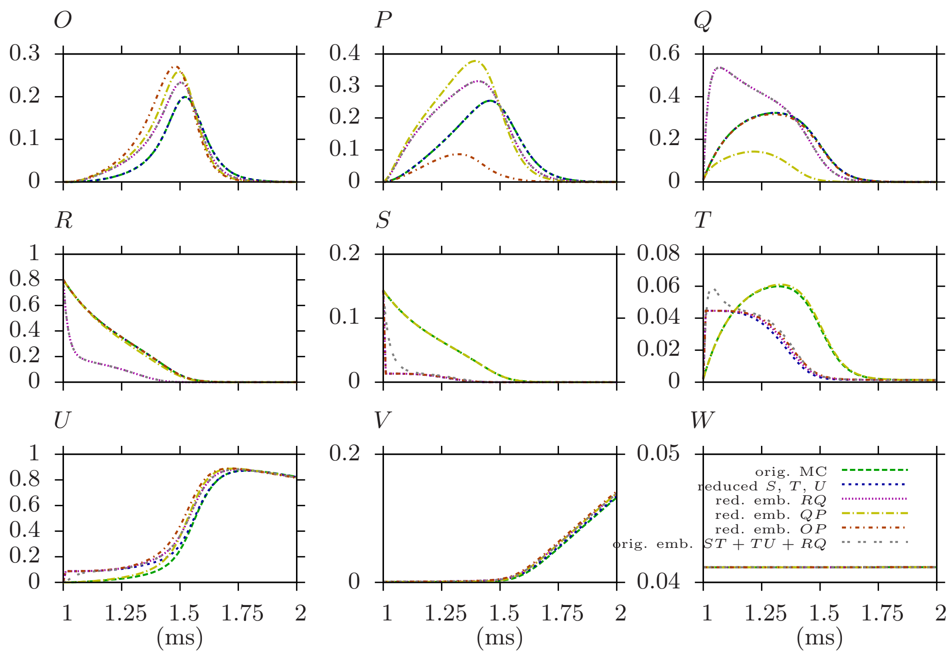


Figure 3.12: States occupancy in I_{Na} Markov chain model driven by recorded action potential onset. The panels show states O to W starting from the top left corner. Green lines show original model, blue lines show reduced S, T, U, magenta lines show embedding of states RQ, yellow lines show embedding of QP, brown lines show embedding of OP where the embedding is done on the reduced S, T, U model with $\varepsilon = 0.1$. Gray lines show embedding of original model with transition rates between states ST, TU and RQ with $\varepsilon = 0.1$.

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{MR}M + \frac{1}{\varepsilon}\alpha_{QR}Q - \left(\alpha_{RM} + \frac{1}{\varepsilon}\alpha_{RQ}\right)R, \tag{3.58d}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{MV}M + \alpha_{WV}W - (\alpha_{VM} + \alpha_{VW})V, \tag{3.58e}$$

$$\begin{aligned}\frac{\mathrm{d}M}{\mathrm{d}t} =& \alpha_{OM}O + \alpha_{PM}P + \alpha_{QM}Q + \alpha_{RM}R + \alpha_{VM}V - \\ & - (\alpha_{MO} + \alpha_{MP} + \alpha_{MQ} + \alpha_{MR} + \alpha_{MV})M,\end{aligned} \tag{3.58f}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.58g}$$

We put together the terms with ε and write the transition rates matrices and non-homogeneous terms for the states

$$\vec{x} = \begin{bmatrix} R \\ Q \end{bmatrix}, \tag{3.59}$$

that we aim to reduce, as follows

$$\boldsymbol{A}_0 = \begin{bmatrix} -\alpha_{RQ} & \alpha_{QR} \\ \alpha_{RQ} & -\alpha_{QR} \end{bmatrix}, \tag{3.60a}$$

$$\boldsymbol{A}_1 = \begin{bmatrix} -\alpha_{RM} & 0 \\ 0 & -(\alpha_{QM} + \alpha_{QP}) \end{bmatrix}, \tag{3.60b}$$

$$\vec{H} = \begin{bmatrix} \alpha_{MR} M \\ \alpha_{MQ} M + \alpha_{PQ} P \end{bmatrix}. \tag{3.60c}$$

3.5.2 Choice of Eigenvectors

The scaled eigenvector for zero eigenvalue and corresponding adjoint vector are found to be

$$\vec{v}_1 = (\alpha_{QR} + \alpha_{RQ})^{-1} \begin{bmatrix} \alpha_{QR} \\ \alpha_{RQ} \end{bmatrix}, \qquad \vec{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{3.61}$$

3.5.3 Reduction of states R and Q by One State L

We substitute the states R and Q of the STU-reduced system 3.48 by a single state $L = R + Q$. The relation between old and new dynamical variables is given by (2.33) as

$$R = \frac{\alpha_{RQ}}{\alpha_{QR} + \alpha_{RQ}} L, \tag{3.62a}$$

$$Q = \frac{\alpha_{QR}}{\alpha_{QR} + \alpha_{RQ}} L, \tag{3.62b}$$

which yields the following system

$$\frac{\mathrm{d}L}{\mathrm{d}t} = (\alpha_{MR} + \alpha_{MQ}) M + \alpha_{PQ} P - \left(\frac{\alpha_{QP}\alpha_{RQ}}{\alpha_{QR} + \alpha_{RQ}} + \frac{\alpha_{QM}\alpha_{RQ} + \alpha_{RM}\alpha_{QR}}{\alpha_{QR} + \alpha_{RQ}} \right) L, \tag{3.63a}$$

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \alpha_{OM} O + \alpha_{PM} P + \frac{\alpha_{QM}\alpha_{RQ} + \alpha_{RM}\alpha_{QR}}{\alpha_{QR} + \alpha_{RQ}} L + \alpha_{VM} V - \\ - (\alpha_{MO} + \alpha_{MP} + (\alpha_{MQ} + \alpha_{MR}) + \alpha_{MV}) M, \tag{3.63b}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \frac{\alpha_{QP}\alpha_{RQ}}{\alpha_{QR} + \alpha_{RQ}} L + \alpha_{MP} M + \alpha_{OP} O - (\alpha_{PQ} + \alpha_{PM} + \alpha_{PO}) P. \tag{3.63c}$$

We define new transition rates as

$$\alpha_{LP} = \frac{\alpha_{QP}\alpha_{RQ}}{\alpha_{QR} + \alpha_{RQ}}, \tag{3.64a}$$

$$\alpha_{ML} = \alpha_{MQ} + \alpha_{MR}, \tag{3.64b}$$

$$\alpha_{PL} = \alpha_{PQ}, \tag{3.64c}$$

$$\alpha_{LM} = \frac{\alpha_{QM}\alpha_{RQ} + \alpha_{RM}\alpha_{QR}}{\alpha_{QR} + \alpha_{RQ}}. \tag{3.64d}$$

The system can be then written as

$$\frac{\mathrm{d}L}{\mathrm{d}t} = \alpha_{ML} M + \alpha_{PL} P - (\alpha_{LP} + \alpha_{LM}) L, \tag{3.65a}$$

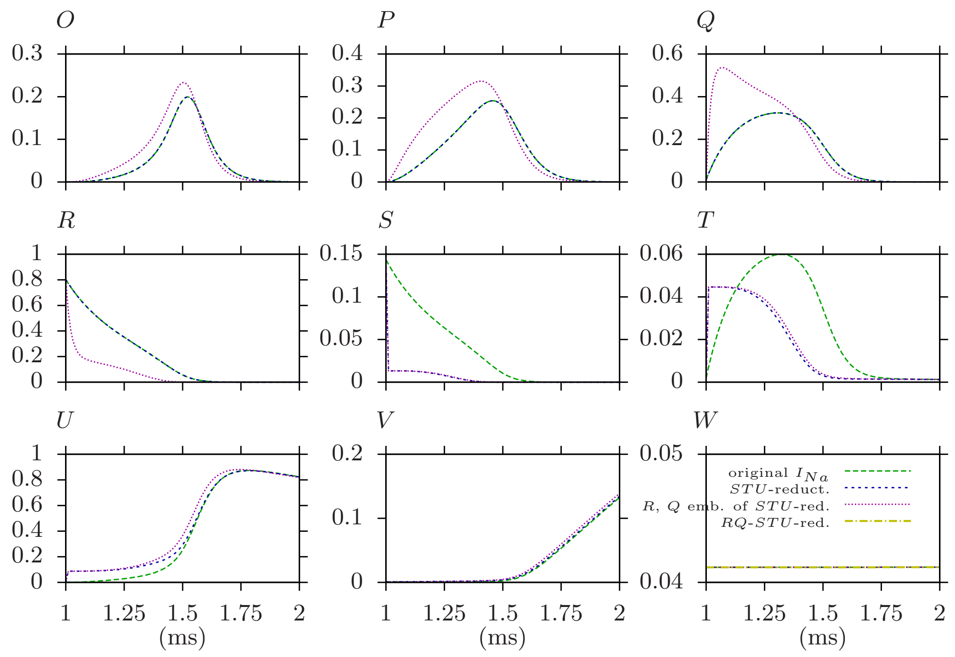


Figure 3.13: States occupancy in I_{Na} Markov chain driven by recorded action potential onset. The panels show states O to W starting from the top left corner. Green lines show the original model, blue lines shown the STU-reduced model, magenta lines shown embedding of states R, Q in STU-reduced model with $\varepsilon = 0.1$, yellow lines show RQ-STU-reduced model.

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \alpha_{OM}O + \alpha_{PM}P + \alpha_{LM}L + \alpha_{VM}V - (\alpha_{MO} + \alpha_{MP} + \alpha_{ML} + \alpha_{MV})M, \tag{3.65b}$$

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P + \alpha_{MO}M - (\alpha_{OP} + \alpha_{OM})O, \tag{3.65c}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{LP}L + \alpha_{MP}M + \alpha_{OP}O - (\alpha_{PL} + \alpha_{PM} + \alpha_{PO})P, \tag{3.65d}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{MV}M + \alpha_{WV}W - (\alpha_{VM} + \alpha_{VW})V, \tag{3.65e}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{3.65f}$$

The diagram of the RQ-STU-reduced Markov chain model is shown on Figure 3.10(c). The Figure 3.13 shows the detail of the first millisecond of the simulations of Markov chain of I_{Na} channel driven by recorded action potential onset obtained from simulation with original Markov chain model from the [1].

3.6 Testing of the Reduced Models within a Cell Model

3.6.1 Choice of the Cellular Model

The I_{Na} Markov chain model was reproduced from a published paper about where it was used for a simulation study within the whole cell model [1]. The authors did not published the code along with the article, however, Clancy has kindly provided her version of the source code on request.

For the sake of the testing of the methods reductions done in the previous sections of this chapter we have used I_{Na} model driven by a recorded action potential from Clancy model. Which means that the I_{Na} model was isolated from the rest of the model. The complete definition of the cellular model along with the other models we have considered is described in the Appendix A.

The authors model was not an autonomous model due to a time dependent stimulation current. We autonomised the model by removing the current. The action potential is initialised by changing the initial value of the membrane potential.

The action potential was simulated using original Markov chain formulation of I_{Na} channel, Markov chain model with parametric embedding of α_{OP} and α_{PO}, and reduced Markov chain model. The initial condition of I_{Na} were obtain by pacing during 5 pulses (cycle length 1 s) in the original Markov chain model. The same initial condition were used in embedded and reduced model. The initial state of the new variable in the reduced model was set as $N = O + P$.

The forward Euler integration method was used for the simulations. The step size used for the simulations in original and reduced I_{Na} model was set to 10 μs in the embedded model the time step was set to 1 μs.

The Figure 3.14 shows the evolution of the membrane potential at four different initial conditions for voltage $\mathrm{V_{m0}} = \{-65, -50, -35, -20\}$ mV. The panel A shows the sub-threshold potential. The potential in this case is insufficient to trigger the excitation – action potential. The panels B, C and D show supra-threshold potential, which result in fast increase of membrane potential due to the influx of sodium current. The higher is the initial value of membrane potential the fastest is the onset of action potential. The Figure 3.15 shows corresponding to the same action potentials. The onset of the I_{Na} current is faster in the embedded and reduced model with respect to the original model.

3.6.2 Stiffness of the Model

We measure the stiffness of the model by the maximum time step size h_{max} at which provides a stable solution in forward Euler solver for the isolated I_{Na} model driven by recorded action potential. The lowest is the value of h_{max} the more stiff is the model.

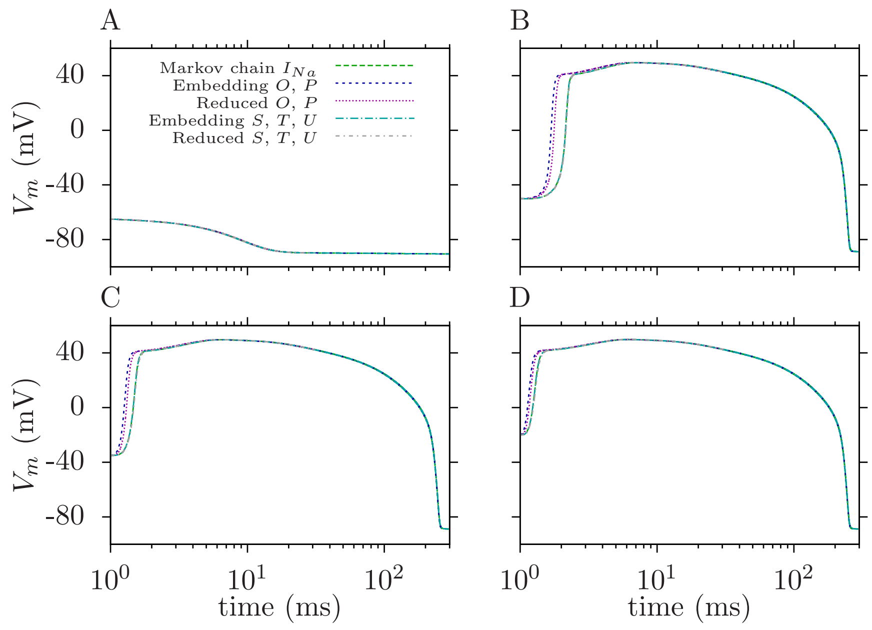


Figure 3.14: Dependence of the action potential ($\mathrm{V_m}$) on the initial conditions in isolated ventricular cell[1] green line show original Markov chain model [1] of I_{Na}, dark blue reduced O, P Markov chain model, light blue . A: $\mathrm{V_{m0}} = -65$ mV, B: $\mathrm{V_{m0}} = -50$ mV, C: $\mathrm{V_{m0}} = -35mV$, D: $\mathrm{V_{m0}} = -20$ mV.

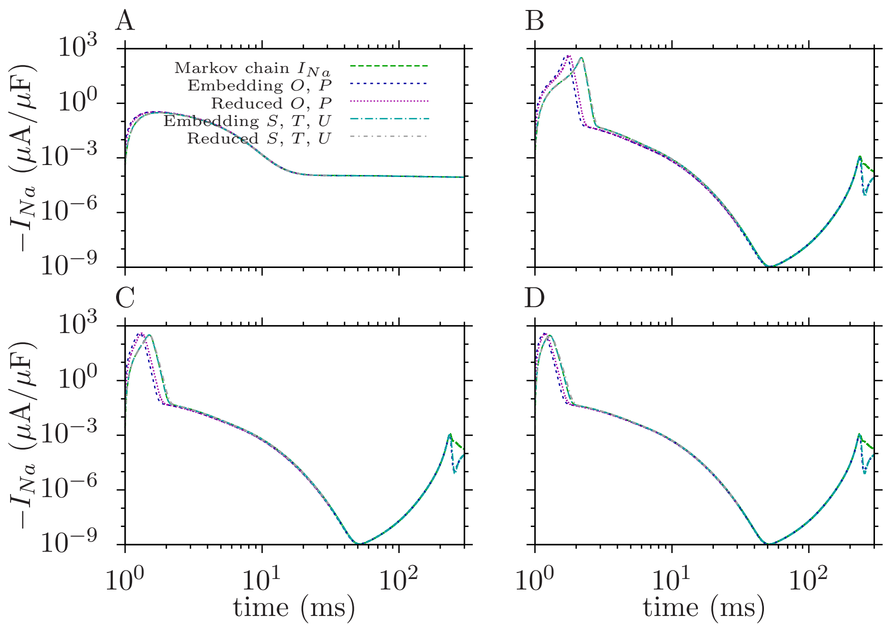


Figure 3.15: Dependence of I_{Na} on the initial conditions in isolated ventricular cell[25] – red original (gates) model, green original Markov chain model [1] of I_{Na}, blue reduced Markov chain model. A: $\mathrm{V_{m0}} = -65$ mV, B: $\mathrm{V_{m0}} = -50$ mV, C: $\mathrm{V_{m0}} = -35mV$, D: $\mathrm{V_{m0}} = -20$ mV.

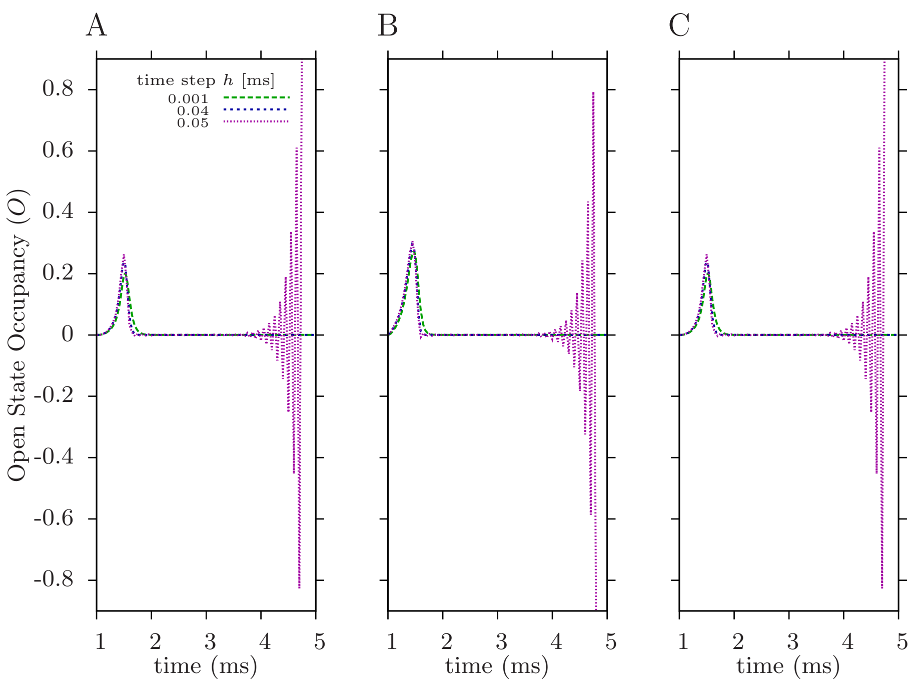


Figure 3.16: Stability of the solution at different values of the time step. A: original 9-states model; B: reduced (O, P) 8-states I_{Na} Markov chain model; C: reduced (S, T, U) 7-states I_{Na} Markov chain model. Green line shows a solution with small time step $h = 0.001$ ms; blue line shows the largest time step which provides stable solution – $h = 0.04$ ms; magenta line show unstable solution with the time step $h = 0.05$.

The simulation step $h = 0.04$ ms provides stable results although slightly deviated from the solution at the step size $h = 0.001$ in both original and reduced I_{Na} model. All three models loose stability around $h_t \approx 0.044$ ms.

When the threshold of stability h_t is reached, the numerical solution starts oscillate around the true solution. In a cellular model, this error propagates to the other components of the model causing a large inaccuracies and the solution become unrealistic. In some cases, the inaccuracies become so large, that they cause overflow of the numerical precision of the floating point variables.

3.7 Conclusions

In this chapter we have applied the asymptotic methods and perturbation theory to reduce the dimensionality of the I_{Na} Markov chain model by adiabatic elimination.

Initially, we have speed up some of the transition rates between the states, which were aimed for the reduction, to test if a transition rates embedding will lead to an accurate reduction. Having found reasonable embeddings we have proceed to the reduction according to the procedures described in the first chapter.

The Figure 3.16 shows the detail of the simulated traces during the first first 5 ms for three reduced models. The characteristics of the instabilities observed are very similar in all those cases.

Our expectation was that the reduced model would be less stiff due to the elimination of the fast transition rates. So, we could use higher step size and therefore the computational cost would be reduced. However, this has not been confirmed. The the instabilities have been found at similar time step sizes as in the original model.

The instability is likely to be caused by other similarly fast transition rates. However, the reductions based on eliminating more pairs of transition rates have been inaccurate and so was not attempted.

Chapter 4

Matrix Rush-Larsen Technique for Markov Chains

4.1 Application to I_{Na} Model

4.1.1 Hybrid Method

In this section we continue our work with the I_{Na} channel by Clancy and Rudy[1]. In the previous chapter we only considered embeddings of the pair of transition rates between two states, and different combinations of embeddings of those pairs. This approach leads to a reduction according to the 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 done depending on the speed at given voltage. We use the Figure 3.1(a) which shows the transition rates at the range of physiological voltages as observed during the action potential between -85 mV and +40 mV.

Markov chain is defined by the equation (2.110). In case of the I_{Na} model the transition rates matrix $\boldsymbol{A}(\mathrm{V_m}(t))$ depends on membrane voltage. If we fix the voltage at a specific value, then the solution can be obtained analytically.

We split the transition rates matrix based on the speed of the transition rates as

$$\boldsymbol{A} = \boldsymbol{A}_0 + \boldsymbol{A}_1 + \boldsymbol{A}_2 \tag{4.1}$$

where the first matrix contains fast transition rates at high voltage as

$$\boldsymbol{A}_0 = \begin{bmatrix} -\alpha_{OU} & \alpha_{PO} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\alpha_{PO} & \alpha_{QP} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\alpha_{QP} & \alpha_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\alpha_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\alpha_{ST} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha_{ST} & -\alpha_{TU} & 0 & 0 & 0 \\ \alpha_{OU} & 0 & 0 & 0 & 0 & \alpha_{TU} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.2}$$

the second matrix contains fast transition rates at low voltage as

$$\boldsymbol{A}_1 = \begin{bmatrix} -\alpha_{OP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha_{OP} & -\alpha_{PQ} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_{PQ} & -\alpha_{QR} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_{QR} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \alpha_{TS} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\alpha_{TS} & \alpha_{UT} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\alpha_{UT} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.3}$$

and the third matrix contains uniformly slow transition rates at both high and low voltages as

$$\boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \alpha_{UO} & 0 & 0 \\ 0 & -\alpha_{PU} & 0 & 0 & 0 & 0 & \alpha_{UP} & 0 & 0 \\ 0 & 0 & -\alpha_{QT} & 0 & 0 & \alpha_{TQ} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\alpha_{RS} & \alpha_{SR} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha_{RS} & -\alpha_{SR} & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_{QT} & 0 & 0 & -\alpha_{TQ} & 0 & 0 & 0 \\ 0 & \alpha_{PU} & 0 & 0 & 0 & 0 & A_{2,U} & \alpha_{VU} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \alpha_{UV} & -(\alpha_{VU}+\alpha_{VW}) & \alpha_{WV} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha_{VW} & -\alpha_{WV} \end{bmatrix} \tag{4.4}$$

where $A_{2,U} = -(\alpha_{UP} + \alpha_{UO} + \alpha_{UV})$.

The fast subsystems $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$ could be speed up by multiplication by $1/\varepsilon$. At the high value of membrane voltage the transition rates matrix $\boldsymbol{A}_0$ becomes leading order term as

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \frac{1}{\varepsilon}\boldsymbol{A}_0\vec{u} + (\boldsymbol{A}_1 + \boldsymbol{A}_2)\vec{u}, \tag{4.5}$$

while at the low values of the voltage the $\boldsymbol{A}_0$ becomes second order term and the leading order term is governed by the transition rates matrix $\boldsymbol{A}_1$ as

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{A}_0 u + \frac{1}{\varepsilon}\boldsymbol{A}_1\vec{u} + \boldsymbol{A}_2\vec{u}. \tag{4.6}$$

The matrix $\boldsymbol{A}_2$ is always in the second order term.

The solution is obtained as

$$\vec{u}_{n+1/3} = \exp(\Delta t\boldsymbol{A}_0(t_n))\vec{u}_n, \tag{4.7a}$$
$$\vec{u}_{n+2/3} = \exp(\Delta t\boldsymbol{A}_1(t_n))\vec{u}_{n+1/3}, \tag{4.7b}$$
$$\vec{u}_{n+1} = \vec{u}_{n+2/3} + \Delta t\boldsymbol{A}_2(t_n)\vec{u}_{n+2/3}, \tag{4.7c}$$

which is rewritten for the leading order system at high voltages as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{PO}P - \alpha_{OU}O, \tag{4.8a}$$
$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{QP}Q - \alpha_{PO}P, \tag{4.8b}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{RQ}R - \alpha_{QP}Q, \tag{4.8c}$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = -\alpha_{RQ}R, \tag{4.8d}$$
$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\alpha_{ST}S, \tag{4.8e}$$
$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{ST}S - \alpha_{TU}T, \tag{4.8f}$$
$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{TU}T + \alpha_{OU}O, \tag{4.8g}$$
$$\frac{\mathrm{d}V}{\mathrm{d}t} = 0, \tag{4.8h}$$
$$\frac{\mathrm{d}W}{\mathrm{d}t} = 0. \tag{4.8i}$$

Due to the specific coupling of those equations and using the assumption that the transition rates do not change much, we can obtain 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 to equations (4.8f) and (4.8c) which are now in a closed form so we obtain the solution of T and Q. The solution for Q is substituted into (4.8b) and solution of T into (4.8g) and we solve P and U. Finally, we solve the O after we had substituted the solution for P. So, we get the solution as

$$O_{n+1/3} = O_n\mu_{OU} + P_nK_{PO} + Q_nK_{QO} + R_nK_{RO} \tag{4.9a}$$

$$P_{n+1/3} = P_n\mu_{PO} + Q_nK_{QP} + R_nK_{RP} \tag{4.9b}$$

$$Q_{n+1/3} = Q_n\mu_{QP} + R_nK_{RQ} \tag{4.9c}$$

$$R_{n+1/3} = R_n\mu_{RQ} \tag{4.9d}$$

$$S_{n+1/3} = S_n\mu_{ST} \tag{4.9e}$$

$$T_{n+1/3} = T_n\mu_{TU} + S_nK_{ST} \tag{4.9f}$$

$$U_{n+1/3} = U_n + T_n[1-\mu_{TU}] + S_nK_{SU} + O_n[1-\mu_{OU}] + P_nK_{PU} + Q_nK_{QU} + R_nK_{RU} \tag{4.9g}$$

where $\mu_{jk} = \exp(-\alpha_{jk}\Delta t)$, and

$$K_{PO} = \frac{\alpha_{PO}(\mu_{PO}-\mu_{OU})}{\alpha_{OU}-\alpha_{PO}} \tag{4.10a}$$

$$K_{QO} = \frac{\alpha_{PO}\alpha_{QP}(\mu_{QP}-\mu_{OU})}{(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{QP})} - \frac{\alpha_{PO}\alpha_{QP}(\mu_{PO}-\mu_{OU})}{(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{PO})} \tag{4.10b}$$

$$\begin{aligned} K_{RO} = & -\frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{QP}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{QP})} + \\ & + \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{PO}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})(\alpha_{OU}-\alpha_{PO})} + \\ & + \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{RQ}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})(\alpha_{OU}-\alpha_{RQ})} - \\ & - \frac{\alpha_{PO}\alpha_{QP}\alpha_{RQ}(\mu_{PO}-\mu_{OU})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})(\alpha_{OU}-\alpha_{PO})} \end{aligned} \tag{4.10c}$$

$$K_{QP} = \frac{\alpha_{QP}(\mu_{QP}-\mu_{PO})}{\alpha_{PO}-\alpha_{QP}} \tag{4.10d}$$

$$K_{RP} = -\frac{\alpha_{QP}\alpha_{RQ}(\mu_{QP}-\mu_{PO})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})} + \frac{\alpha_{QP}\alpha_{RQ}(\mu_{RQ}-\mu_{PO})}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})} \tag{4.10e}$$

$$K_{RQ} = -\frac{\alpha_{RQ}(\mu_{QP}-\mu_{RQ})}{\alpha_{QP}-\alpha_{RQ}} \tag{4.10f}$$

$$K_{ST} = -\frac{\alpha_{ST}(\mu_{TU}-\mu_{ST})}{\alpha_{TU}-\alpha_{ST}} \tag{4.10g}$$

$$K_{SU} = 1 + \frac{\alpha_{ST}\mu_{TU}-\alpha_{TU}\mu_{ST}}{\alpha_{TU}-\alpha_{ST}} \tag{4.10h}$$

$$K_{PU} = 1 - \frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}} \tag{4.10i}$$

$$\begin{aligned} K_{QU} = & \frac{\alpha_{PO}}{\alpha_{PO}-\alpha_{QP}}\left(1-\frac{\alpha_{OU}\mu_{QP}-\alpha_{QP}\mu_{OU}}{\alpha_{OU}-\alpha_{QP}}\right) - \\ & - \frac{\alpha_{QP}}{\alpha_{PO}-\alpha_{QP}}\left(1-\frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}}\right) \end{aligned} \tag{4.10j}$$

$$K_{RU} = -\frac{\alpha_{PO}\alpha_{RQ}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})}\left(1-\frac{\alpha_{OU}\mu_{QP}-\alpha_{QP}\mu_{OU}}{\alpha_{OU}-\alpha_{QP}}\right) +$$

$$+\frac{\alpha_{QP}\alpha_{RQ}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{QP})}\left(1-\frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}}\right)+$$
$$+\frac{\alpha_{PO}\alpha_{QP}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})}\left(1-\frac{\alpha_{OU}\mu_{RQ}-\alpha_{RQ}\mu_{OU}}{\alpha_{OU}-\alpha_{RQ}}\right)-$$
$$-\frac{\alpha_{QP}\alpha_{RQ}}{(\alpha_{QP}-\alpha_{RQ})(\alpha_{PO}-\alpha_{RQ})}\left(1-\frac{\alpha_{OU}\mu_{PO}-\alpha_{PO}\mu_{OU}}{\alpha_{OU}-\alpha_{PO}}\right) \tag{4.10k}$$

The leading order system at low voltage can be written as follows

$$\frac{\mathrm{d}O}{\mathrm{d}t} =-\alpha_{OP}O, \tag{4.11a}$$
$$\frac{\mathrm{d}P}{\mathrm{d}t} =\alpha_{OP}O-\alpha_{PQ}P, \tag{4.11b}$$
$$\frac{\mathrm{d}Q}{\mathrm{d}t} =\alpha_{PQ}P-\alpha_{QR}Q, \tag{4.11c}$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} =\alpha_{QR}Q, \tag{4.11d}$$
$$\frac{\mathrm{d}S}{\mathrm{d}t} =\alpha_{TS}T, \tag{4.11e}$$
$$\frac{\mathrm{d}T}{\mathrm{d}t} =\alpha_{UT}U-\alpha_{TS}T, \tag{4.11f}$$
$$\frac{\mathrm{d}U}{\mathrm{d}t} =-\alpha_{UT}U, \tag{4.11g}$$
$$\frac{\mathrm{d}V}{\mathrm{d}t} =0, \tag{4.11h}$$
$$\frac{\mathrm{d}W}{\mathrm{d}t} =0. \tag{4.11i}$$

Again, using the specific coupling of the equations we can solve this equation. 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 (4.11f), solution for O into (4.11b) which gets in the closed form, so we obtain solutions for T and P. Substituting T also (4.11e) and P into (4.11c) we can solve S and Q. Finally, we substitute Q into (4.11d) to get R. The result can be written as

$$O_{n+2/3} =O_{n+1/3}\mu_{OP} \tag{4.12a}$$
$$P_{n+2/3} =O_{n+1/3}L_{OP}+P_{n+1/3}\mu_{PQ} \tag{4.12b}$$
$$Q_{n+2/3} =O_{n+1/3}L_{OQ}+P_{n+1/3}L_{PQ}+Q_{n+1/3}\mu_{QR} \tag{4.12c}$$
$$R_{n+2/3} =O_{n+1/3}L_{OR}+P_{n+1/3}L_{PR}+Q_{n+1/3}[1-\mu_{QR}]+R_{n+1/3} \tag{4.12d}$$
$$S_{n+2/3} =U_{n+1/3}L_{US}+T_{n+1/3}[1-\mu_{TS}]+S_{n+1/3} \tag{4.12e}$$
$$T_{n+2/3} =U_{n+1/3}L_{UT}+T_{n+1/3}\mu_{TS} \tag{4.12f}$$
$$U_{n+2/3} =U_{n+1/3}\mu_{UT} \tag{4.12g}$$
$$V_{n+2/3} =V_{n+1/3} \tag{4.12h}$$
$$W_{n+2/3} =W_{n+1/3} \tag{4.12i}$$

where

$$L_{OP} = \frac{\alpha_{OP}(\mu_{OP} - \mu_{PQ})}{\alpha_{PQ} - \alpha_{OP}} \tag{4.13a}$$

$$L_{OQ} = \frac{\alpha_{PQ}\alpha_{OP}(\mu_{OP} - \mu_{QR})}{(\alpha_{PQ} - \alpha_{OP})(\alpha_{QR} - \alpha_{OP})} - \frac{\alpha_{PQ}\alpha_{OP}(\mu_{PQ} - \mu_{QR})}{(\alpha_{PQ} - \alpha_{OP})(\alpha_{QR} - \alpha_{PQ})} \tag{4.13b}$$

$$L_{PQ} = \frac{\alpha_{PQ}(\mu_{PQ} - \mu_{QR})}{\alpha_{QR} - \alpha_{PQ}} \tag{4.13c}$$

$$L_{OR} = 1 + \frac{\alpha_{PQ}(\alpha_{OP}\mu_{QR} - \alpha_{QR}\mu_{OP})}{(\alpha_{PQ} - \alpha_{OP})(\alpha_{QR} - \alpha_{OP})} - \frac{\alpha_{OP}(\alpha_{PQ}\mu_{QR} - \alpha_{QR}\mu_{PQ})}{(\alpha_{PQ} - \alpha_{OP})(\alpha_{QR} - \alpha_{PQ})} \tag{4.13d}$$

$$L_{PR} = 1 + \frac{\alpha_{PQ}\mu_{QR} - \alpha_{QR}\mu_{PQ}}{\alpha_{QR} - \alpha_{PQ}} \tag{4.13e}$$

$$L_{US} = 1 + \frac{\alpha_{UT}\mu_{TS} - \alpha_{TS}\mu_{UT}}{\alpha_{TS} - \alpha_{UT}} \tag{4.13f}$$

$$L_{UT} = \frac{\alpha_{UT}(\mu_{UT} - \mu_{TS})}{\alpha_{TS} - \alpha_{UT}} \tag{4.13g}$$

The leading order term at high potential $\boldsymbol{A}_0$ and low potential $\boldsymbol{A}_1$ have been solved analytically and so can be used for all computations. The complementary second order term contains transition rates which contain only uniformly slow transition rates (both at high and low potentials). This system reads as

$$\frac{\mathrm{d}O}{\mathrm{d}t} = \alpha_{UO}U, \tag{4.14a}$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \alpha_{UP}U - \alpha_{PU}P, \tag{4.14b}$$

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = \alpha_{TQ}T - \alpha_{QT}Q, \tag{4.14c}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \alpha_{SR}S - \alpha_{RS}R, \tag{4.14d}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \alpha_{RS}R - \alpha_{SR}S, \tag{4.14e}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \alpha_{QT}Q - \alpha_{TQ}T, \tag{4.14f}$$

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \alpha_{PU}P + \alpha_{VU}V - (\alpha_{UP} + \alpha_{UO} + \alpha_{UV})U, \tag{4.14g}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \alpha_{UV}U + \alpha_{WV}W - (\alpha_{VU} + \alpha_{VW})V, \tag{4.14h}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \alpha_{VW}V - \alpha_{WV}W. \tag{4.14i}$$

This system can not be solved analytically. However, the instability threshold is higher, which is because it contains only slow transition rates, so we approximate the solution using forward Euler method

$$O_{n+1} = O_{n+2/3} + \alpha_{UO}U_{n+2/3}\Delta t, \tag{4.15a}$$

$$P_{n+1} = P_{n+2/3} + \left(\alpha_{UP}U_{n+2/3} - \alpha_{PU}P_{n+2/3}\right)\Delta t, \tag{4.15b}$$

$$Q_{n+1} = Q_{n+2/3} + \left(\alpha_{TQ} T_{n+2/3} - \alpha_{QT} Q_{n+2/3}\right) \Delta t, \tag{4.15c}$$

$$R_{n+1} = R_{n+2/3} + \left(\alpha_{SR} S_{n+2/3} - \alpha_{RS} R_{n+2/3}\right) \Delta t, \tag{4.15d}$$

$$S_{n+1} = S_{n+2/3} + \left(\alpha_{RS} R_{n+2/3} - \alpha_{SR} S_{n+2/3}\right) \Delta t, \tag{4.15e}$$

$$T_{n+1} = T_{n+2/3} + \left(\alpha_{QT} Q_{n+2/3} - \alpha_{TQ} T_{n+2/3}\right) \Delta t, \tag{4.15f}$$

$$U_{n+1} = U_{n+2/3} + \left(\alpha_{PU} P_{n+2/3} + \alpha_{VU} V_{n+2/3} - (\alpha_{UP} + \alpha_{UO} + \alpha_{UV}) U_{n+2/3}\right) \Delta t, \tag{4.15g}$$

$$V_{n+1} = V_{n+2/3} + \left(\alpha_{UV} U_{n+2/3} + \alpha_{WV} W_{n+2/3} - (\alpha_{VU} + \alpha_{VW}) V_{n+2/3}\right) \Delta t, \tag{4.15h}$$

$$W_{n+1} = W_{n+2/3} + \left(\alpha_{VW} V_{n+2/3} - \alpha_{WV} W_{n+2/3}\right) \Delta t. \tag{4.15i}$$

Then the equation (4.7) can be reformulated as:

$$\vec{u}^{n+2/3} = \boldsymbol{T}(t, \Delta t) \vec{u}^{n} \tag{4.16a}$$

$$\vec{u}^{n+1} = \vec{u}^{n+2/3} + \Delta t \boldsymbol{A}_2(t_n) \vec{u}^{n+2/3} \tag{4.16b}$$

where $\boldsymbol{T}(t, \Delta t) = \exp(\Delta t \boldsymbol{A}_1(t_n)) \exp(\Delta t \boldsymbol{A}_0(t_n))$ as

$$\boldsymbol{T}(t, \Delta t) = \begin{bmatrix} \mu_{OP} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ L_{OP} & \mu_{PQ} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ L_{OQ} & L_{PQ} & \mu_{QR} & 0 & 0 & 0 & 0 & 0 & 0 \\ L_{OR} & L_{PR} & [1-\mu_{QR}] & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & [1-\mu_{TS}] & L_{US} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & L_{UT} & \mu_{TS} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_{UT} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mu_{OU} & K_{PO} & K_{QO} & K_{RO} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_{PO} & K_{QP} & -K_{RP} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_{QP} & -K_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_{RQ} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_{ST} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -K_{ST} & \mu_{TU} & 0 & 0 & 0 \\ [1-\mu_{OU}] & K_{PU} & K_{QU} & K_{RU} & K_{SU} & [1-\mu_{TU}] & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.17}$$

To speed up our solution algorithm, we precompute the matrix $\boldsymbol{T}(t, h)$ and $\Delta t \boldsymbol{A}_2$ for a physiological range of voltages.

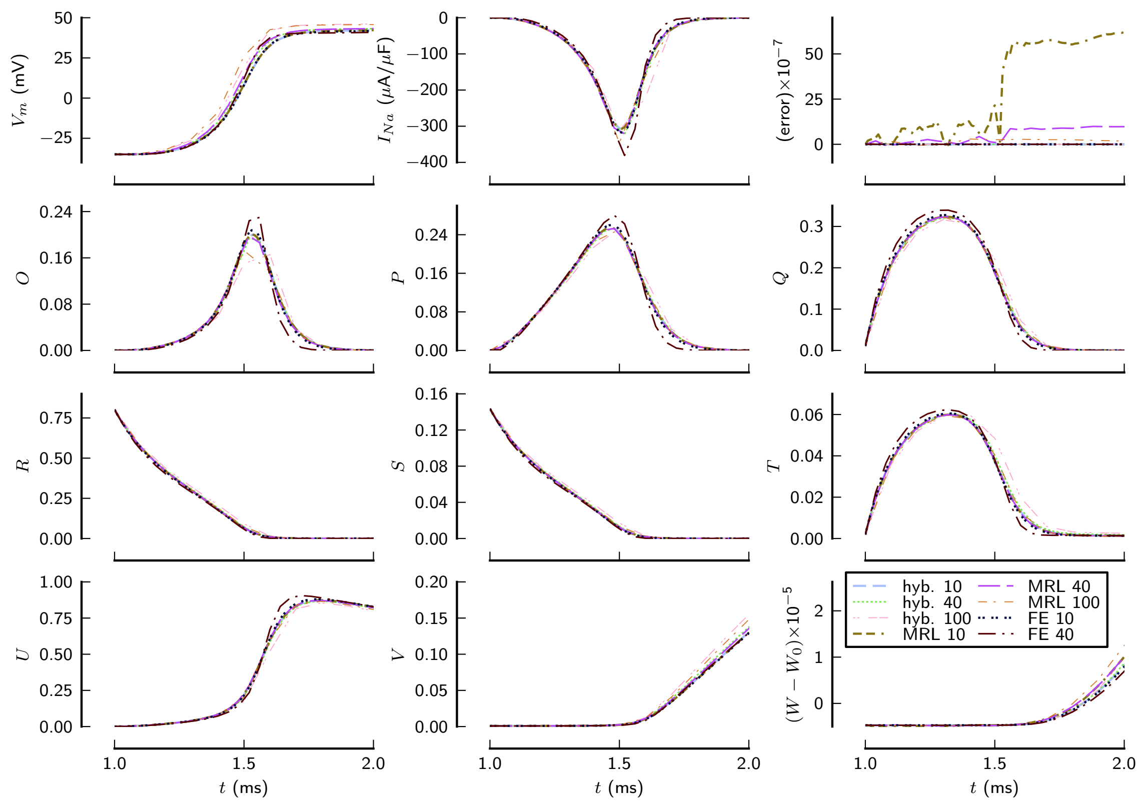


Figure 4.1: Action potential (V_m), I_{Na} current, error from conservation law, and I_{Na} Markov chain states occupancy (zoom to first 1 ms). Forward Euler solution of slow subsystem combined with analytic solution of fast subsystem at low and high potential (an+fE); solution using tabulated eigenvalue and eigenvector matrices (eig); and forward Euler solution (fE). The solution was obtained at time step $h = 0.01, 0.04$ and 0.10 ms.

4.1.2 Simulation in Single Cell Model

The model (4.7) was implemented into a single cell model by Clancy and Rudy (2002)[1] as specified in the Appendix A. Figure 4.1 shows the simulations. The I_{Na} Markov chain model was used and solved using:

- forward Euler solution of the I_{Na} model,
- combination of numerical solution of the fast subsystem and analytical solution of the slow subsystem, and
- combination of numerical solution of the fast subsystem and analytical solution of the slow subsystem using pre-calculated table of transition rates.

The table of pre-calculated values was created for values of potential ranging from -100–70 mV with grid of 0.01 mV. The figure shows simulation at three different values of time steps ($h = 0.01, 0.04, 0.10$ ms). In the numerical solution of the system using the first method the solutions were unstable from the time step $h = 0.05$. The traces of combined solution of slow and fast subsystem have overlapped.

Table 4.1 shows the elapsed CPU time during the simulations in the cell model. The simulation were performed for 100 pulses of CL 1000 ms. Five consecutive

Table 4.1: Time elapsed [s] in cell simulation during 100 pulses with CL= 1000 ms

	$h = 0.01$		$h = 0.04$		$h = 0.10$	
I_{Na} Model	I_{Na}	Total	I_{Na}	Total	I_{Na}	Total
H-H	5.03	23.66	1.09	5.94		
forward Euler (FE) + Heun	5.26	24.12	1.44	6.12		
FE	5.44	24.01	1.29	6.02		
FE (tab.tr.)	2.74	21.38	0.69	5.36		
MRL (tab.)	4.79	24.36	1.23	6.23	0.54	2.45
combined FE + analytic	9.51	28.13	2.22	7.04	0.79	2.83
combined FE + analytic (tab.)	2.89	21.71	0.77	5.49	0.37	2.21

runs were performed and the minimum necessary time is shown. Each action potential was initiated by setting the membrane potential to $\mathrm{V_m} = -35$ mV the time step was set to $h = 0.01, 0.04$ and in the combined numerical and analytic solution also with $h = 0.10$ ms.

4.2 Application to RyR and $I_{\mathrm{Ca}(L)}$ Models

4.2.1 Cellular model

Cardiac myocyte model published by Faber et al. (2007)[2] 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 Faber's model obtained from the Rudy Laboratory website[26]. In the model all the gate ion channels are implemented using Rush-Larsen method, which always yields stable solution. The Markov chain models are used for Ryanodine receptor (RyR) and $I_{\mathrm{Ca}(L)}$ and solved using forward Euler scheme, which is also employed for the integration of ionic concentrations ($[\mathrm{Ca}^{2+}]_i$, $[\mathrm{Na}^{+}]_i$, $[\mathrm{K}^{+}]_i$) and 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 causing significant deviation from the true solution, and above $\Delta t = 9\ \mu\mathrm{s}$ in the overflow in the double floating point values (the numerical blow-up).

In this section we describe the extension of the Rush-Larsen method to the RyR and $I_{\mathrm{Ca}(L)}$ channels. This helps to avoid the instabilities due to the stiffness of such models allowing larger time steps and substantially reducing computational cost. In our case, we achieved to increase the time step providing stable solution from 6 $\mu\mathrm{s}$ to 180 $\mu\mathrm{s}$. This leads to reduction of computational cost almost 30 times.

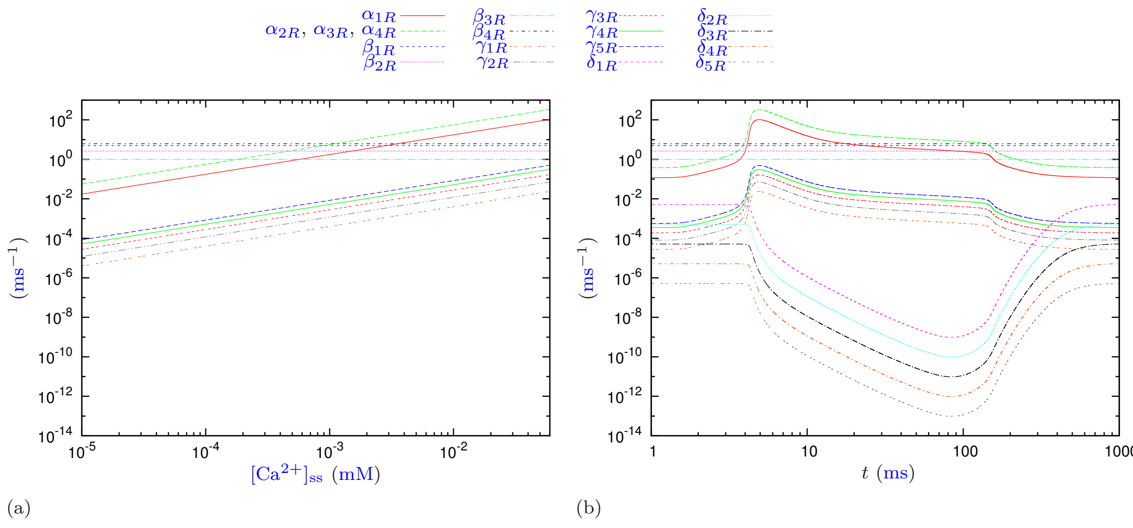


Figure 4.2: Transition rates in the Markov chain model of RyR (diagram in Figure 1.8). Panel (a) shows transition rates as function of $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. Panel (b) shows the transition rates during the action potential. The plots are in logarithmic scale.

4.2.2 RyR Markov Chain Model

The RyR model of Faber et al. (2007) was described in the subsection 1.4.3. The diagram of the Markov chain model of RyR is shown on the Figure 1.8. The figure Figure 4.2 shows transition rates, i.e. the probability of the transitions per millisecond, given that the channel is in a particular state.

Transition rates between the states in the top row (horizontal transition rates on Figure 1.8) are identical in both rows, i.e. a transition rate in the top row correspond to the transition rate at the same location of the bottom row. The transition rates β's towards more closed states, which are shown in the diagram as transitions from right to left, are given constants not depending on any other variables. The transition rates α's towards open states are functions of sub-space 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 mM, range from 10^{-2} to about 10^{2} $\mu\mathrm{s}^{-1}$.

The transitions between the top and the bottom row on the diagram (vertical transition rates on Figure 1.8) are uniformly slow. Their maximum value only reaches 1 $\mu\mathrm{s}^{-1}$. The transition rates γ's, shown as the transitions from the top to the bottom on the diagram, depend only on the sub-space calcium concentration $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$, while the transitions δ's which are in the opposite direction from the bottom to the top on the diagram are function of two calcium concentrations. The first is the calcium concentration in sub-space $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$, like in the other transition rates, and second is the calcium concentration in junctional space of sarcoplasmic reticulum (JSR) $[\mathrm{Ca}^{2+}]_{\mathrm{JSR}}$. Because of the dependence dependence on two transition rates it has not been shown on the panel (a), but only on panel (b) in Figure 4.2.

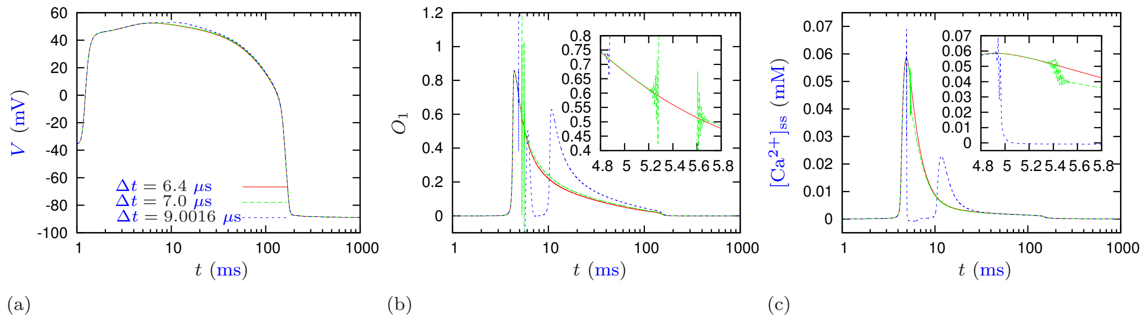


Figure 4.3: Numerical instability caused by RyR model using forward Euler integration.

To study the numerical properties of the cellular model, we perform a number of simulations with different time step sizes. Figure 4.3 shows the result of the simulations. The observed numerical instability first occurred in the state O_1 at the $\Delta t = 6.7\ \mu s$. Because, the state O_1 is directly connected with the calcium release from the sarcoplasmic reticulum, at the time step size $\Delta t = 7.0\ \mu s$ we observe clear instability in the calcium concentration in the subspace. $[Ca^{2+}]_{ss}$.

The membrane voltage V_m and other dynamical variables do not present instability until the time-step size reaches values $\Delta t > 9.0016\ \mu s$. 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 low 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 the error propagates to other variables in the model, and the simulation fails (not shown).

Comparing the form of the speed of the transition rates in under the action potential Figure 4.2b with Figure 4.3 we observe a clear correspondence between the time point when the instability occurs, and the time when the fastest transition rates ($\alpha_{2R} = \alpha_{3R} = \alpha_{4R}$) reach their maximum. Which confirms that the instability in the forward Euler solver occurs due to the fast transition rates.

To address the instability we suggest to employ the matrix Rush-Larsen solver as described in Subsection 2.3.3, together with tabulating the coefficients of the exponential operator matrix. However, in the case of RyR channel, the complication is the dependence of the transition rates on more then one dynamical variables. The tabulation on 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 divided into subsystems according to the speeds. First subsystem contains α's and β's, which are fast, and 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 a form

$$\boldsymbol{M}([\mathrm{Ca}^{2+}]_{\mathrm{ss}}, \mathrm{CSQN}) = \left(\begin{array}{c|c} \boldsymbol{F} & 0 \\ \hline 0 & \boldsymbol{F} \end{array}\right) + \boldsymbol{G} \tag{4.18}$$

where the first matrix splits into two identical matrices $\boldsymbol{F}$ Due to the particular structure of the RyR diagram , 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 as

$$\boldsymbol{F} = \begin{bmatrix} -\beta_{1R} & \alpha_{1R} & 0 & 0 & 0 \\ \beta_{1R} & -(\alpha_{1R}+\beta_{2R}) & \alpha_{2R} & 0 & 0 \\ 0 & \beta_{2R} & -(\alpha_{2R}+\beta_{3R}) & \alpha_{3R} & 0 \\ 0 & 0 & \beta_{3R} & -(\alpha_{3R}+\beta_{4R}) & \alpha_{4R} \\ 0 & 0 & 0 & \beta_{4R} & -\alpha_{4R} \end{bmatrix}, \tag{4.19}$$

and $\|\boldsymbol{G}(\dots)\| \lesssim 1\,\mathrm{ms}^{-1}$ represents the "vertical" transition rates as

$$\boldsymbol{G} = \begin{bmatrix} -\gamma_{1R} & 0 & 0 & 0 & 0 & \delta_{1R} & 0 & 0 & 0 & 0 \\ 0 & -\gamma_{2R} & 0 & 0 & 0 & 0 & \delta_{2R} & 0 & 0 & 0 \\ 0 & 0 & -\gamma_{3R} & 0 & 0 & 0 & 0 & \delta_{3R} & 0 & 0 \\ 0 & 0 & 0 & -\gamma_{4R} & 0 & 0 & 0 & 0 & \delta_{4R} & 0 \\ 0 & 0 & 0 & 0 & -\gamma_{5R} & 0 & 0 & 0 & 0 & \delta_{5R} \\ \gamma_{1R} & 0 & 0 & 0 & 0 & -\delta_{1R} & 0 & 0 & 0 & 0 \\ 0 & \gamma_{2R} & 0 & 0 & 0 & 0 & -\delta_{2R} & 0 & 0 & 0 \\ 0 & 0 & \gamma_{3R} & 0 & 0 & 0 & 0 & -\delta_{3R} & 0 & 0 \\ 0 & 0 & 0 & \gamma_{4R} & 0 & 0 & 0 & 0 & -\delta_{4R} & 0 \\ 0 & 0 & 0 & 0 & \gamma_{5R} & 0 & 0 & 0 & 0 & -\delta_{5R} \end{bmatrix}. \tag{4.20}$$

Hence, for this Markov chain we use a mix of Matrix Rush-Larsen and forward Euler according to Lie-style operator splitting as

$$\vec{v}_{n+1/2} = \exp\left(\Delta t\, \boldsymbol{F}(t_n)\right)\, \vec{v}_n \tag{4.21}$$

$$\vec{w}_{n+1/2} = \exp\left(\Delta t\, \boldsymbol{F}(t_n)\right)\, \vec{w}_n \tag{4.22}$$

$$\vec{u}_{n+1} = \vec{u}_{n+1/2} + \Delta t\, \boldsymbol{G}(t_n)\, \vec{u}_{n+1/2}. \tag{4.23}$$

where

$$\vec{v} = [I_1, I_2, I_3, I_4, I_5]^T$$

contains the states of the top row, and

$$\vec{w} = [C_1, C_2, C_3, C_4, O_1]^T$$

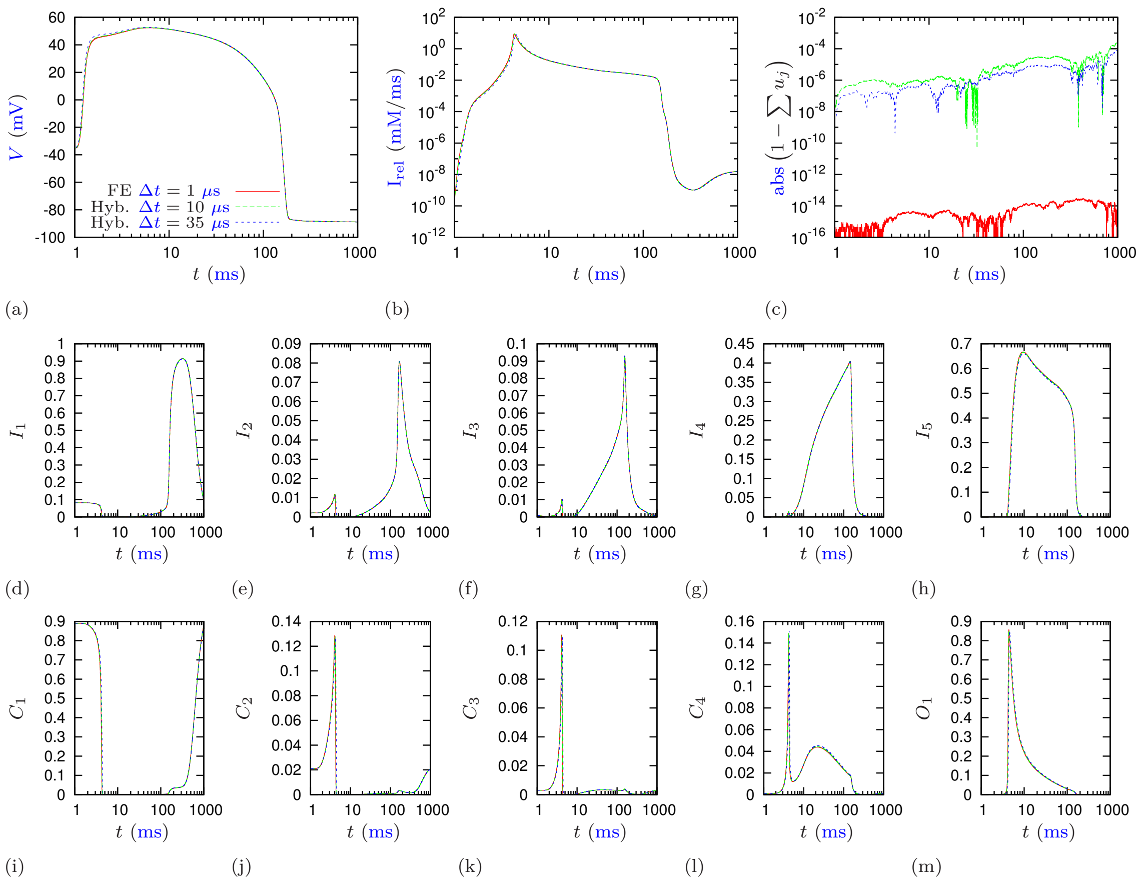


Figure 4.4: Comparison of integrators (forward Euler and hybrid) for RyR Markov chain model: (a) action potential; (b) I_{rel}; (c) deviation from states conservation law; (d-m) states occupancy in RyR MC model.

contains the states of the bottom row in the diagram. Then the state vector including all the states can be written as $\vec{u} = (\vec{v}, \vec{w})$.

Before we start with the computation, we create a look-up table of eigenvalues and eigenvectors for a range of values of dependent variable in the transition rates for $\boldsymbol{F}$. The transition rates in the $\boldsymbol{G}$ are not tabulated as the tabulation does not offer mayor speed up in the forward Euler method, which will be used for the integration of this slow sub-system.

The tabulation variable for the $\boldsymbol{F}$ is remains $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. Since $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$ ranges in five orders of magnitude, the tabulation for linearised space of sufficient precision would be computationally and memory expensive. Therefore, we benefit from using logarithmic space, despite the higher computational demand on using logarithmic functions.

Using the suggested hybrid method, the instability observed previously at the time step values $\Delta t > 6.7$ μs has disappeared.

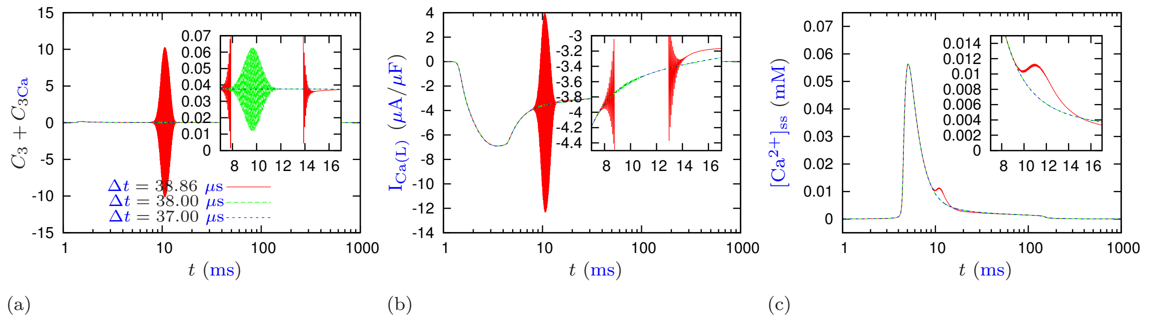


Figure 4.5: Numerical instability caused by $I_{\mathrm{Ca}(L)}$ model using forward Euler integration and hybrid method for RyR channel.

4.2.3 $I_{\mathrm{Ca}(L)}$ Markov Chain Model

The primary limiting factor in increasing the step size was RyR Markov chain model. The Faber et al. (2006) model contains other 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 $I_{\mathrm{Ca}(L)}$ model is the next cause of instability, which is observed as we increase the time step to even higher values.

Figure 4.5 shows the instability cause by the $I_{\mathrm{Ca}(L)}$ model using forward Euler solution. The largest time step allowing stable solution is $\Delta t = 37\ \mu\mathrm{s}$. The state $C_3 + C_{\mathrm{Ca}3}$ is the first to become unstable at the $\Delta t = 38\ \mu\mathrm{s}$. This instability can be hardly observed in other variables, because of only minor influence on the $I_{\mathrm{Ca}(L)}$ current. However, we observe wide oscillations around the exact solution at time steps $\Delta t = 38.86$. This oscillations propagate to $I_{\mathrm{Ca}(L)}$ and cause a drift in $[\mathrm{Ca}^{2+}]_{\mathrm{ss}}$. Higher time steps lead to propagation of fatal numerical errors.

There is no clear separation between the speed of the transition rates of the $I_{\mathrm{Ca}(L)}$ model (as shown on the Figure 4.6). Similarly to RyR model, the instability occurs at the same time point, as the one fastest transition rate (γ_f) reaches its maximum.

The detailed description of $I_{\mathrm{Ca}(L)}$ model was given in Subsubsection 1.4.2. The diagram of the model is shown on the Figure 1.7. The $I_{\mathrm{Ca}(L)}$ model operates in two regimes: Ca-mode and $\mathrm{V_m}$-mode. The conductive state O is present only the $\mathrm{V_m}$-mode, and all the states in the Ca-mode are non-conductive. The transitions between the modes is calcium dependent. The states and transitions rates within both modes correspond to each other, i.e. given that the location is the same, the probability of transitions to the neighbouring states is identical in both modes.

This allows to factor out the Ca^{2+} dependent inactivation as a gate model. This gate model is then combined with a single layer of the Markov chain similarly to the procedures to convert between the gate and Markov chain models as described in the Subsection 1.3.2. The idea of factoring out the calcium dependent transition

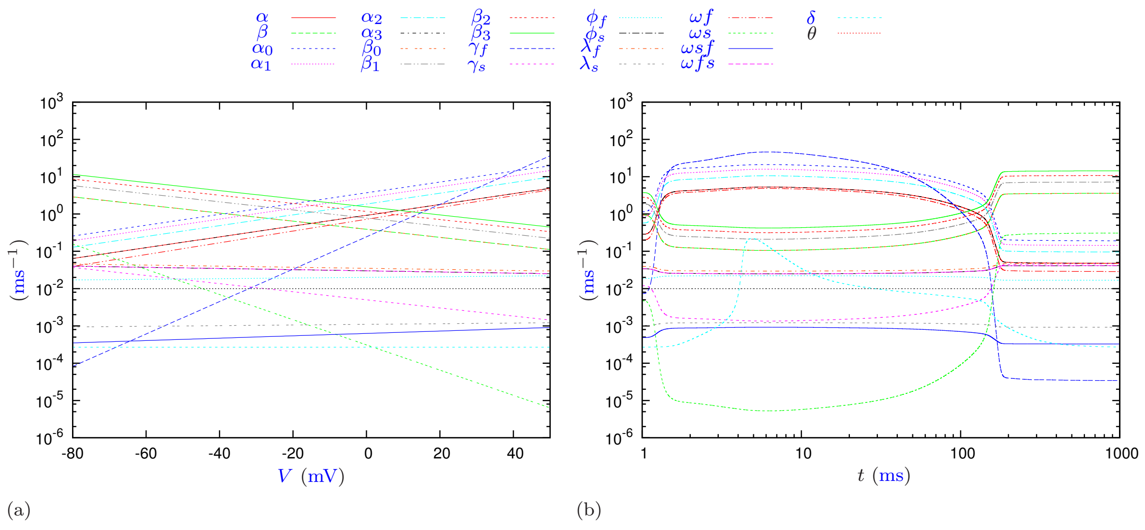


Figure 4.6: Transition rates in $I_{\mathrm{Ca}(L)}$ model. (a) as function of membrane voltage ($\mathrm{V_m}$); (b) during action potential.

is also used in the authors' code[26], however the equations in the paper are presented in the long form with 14 rather than 7 states[2].

This factorisation procedure gives the following system of ODEs

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \alpha_q(1-q) - \beta_q q, \tag{4.24a}$$

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \boldsymbol{D}(\mathrm{V_m}(t))\vec{u} \tag{4.24b}$$

where q is a gate controlling the calcium dependent switching between the modes, and $\boldsymbol{D}$ is the transition rates matrix of a single layer of the Markov chain. The transition rates matrix reads as

$$\boldsymbol{D} = \begin{bmatrix} -\alpha_0 & \beta_0 & 0 & 0 & 0 & 0 & 0 \\ \alpha_0 & -(\alpha_1+\beta_0) & \beta_1 & 0 & 0 & 0 & 0 \\ 0 & \alpha_1 & -(\beta_1+\alpha_2) & \beta_2 & 0 & 0 & 0 \\ 0 & 0 & \alpha_2 & D_{C3} & \beta_3 & \omega_f & \omega_s \\ 0 & 0 & 0 & \alpha_3 & D_O & \lambda_f & \lambda_s \\ 0 & 0 & 0 & \gamma_f & \phi_f & D_{Ivf} & \omega_{sf} \\ 0 & 0 & 0 & \gamma_s & \phi_s & \omega_{fs} & D_{Vs} \end{bmatrix}, \tag{4.25}$$

where the diagonal entries are

$$D_{C3} = -(\gamma_f + \gamma_s + \alpha_3 + \beta_2), \tag{4.26}$$

$$D_O = -(\phi_f + \phi_s + \beta_3), \tag{4.27}$$

$$D_{Ivf} = -(\omega_f + \lambda_f + \omega_{fs}), \tag{4.28}$$

$$D_{Vs} = -(\omega_s + \lambda_s + \omega_{sf}). \tag{4.29}$$

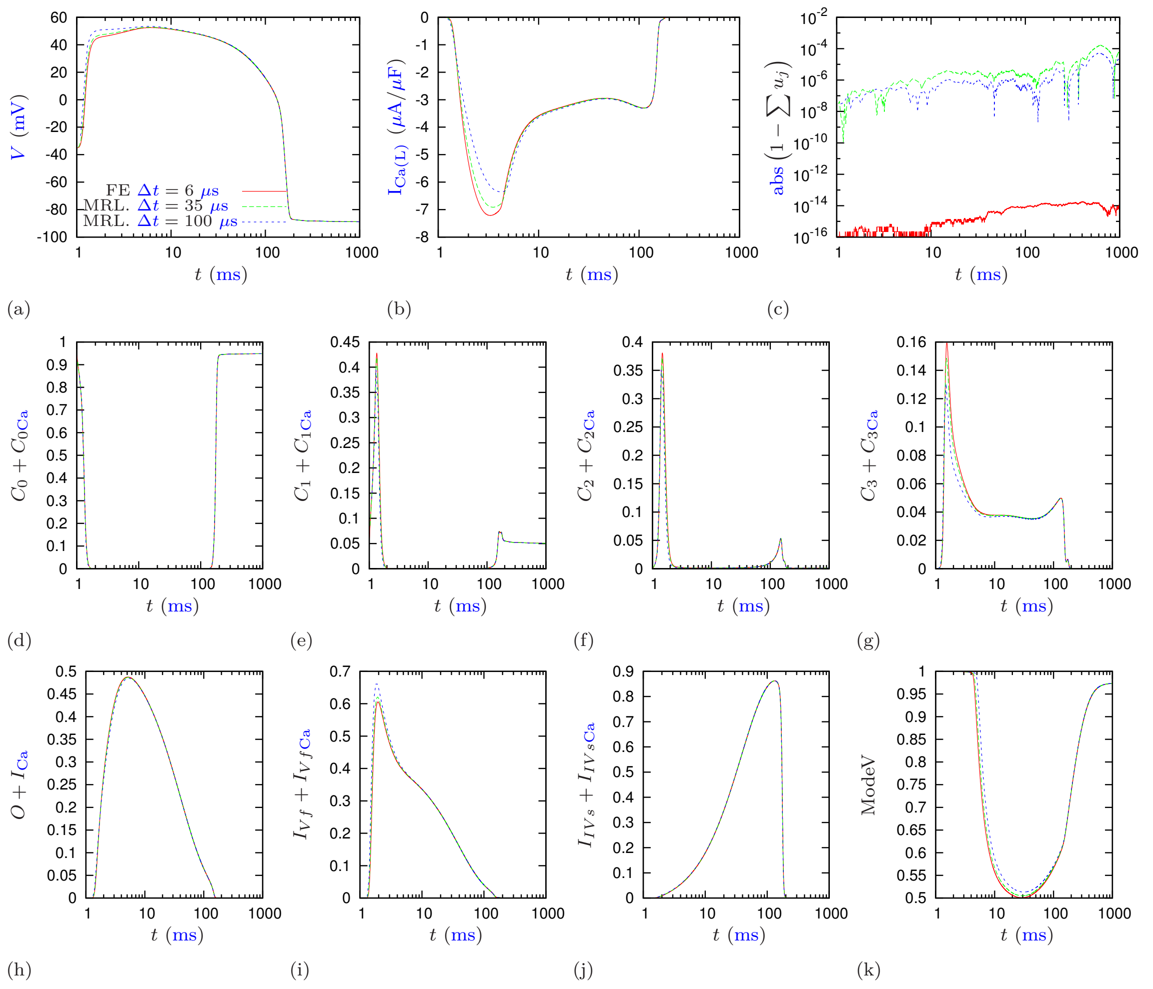


Figure 4.7: Comparison of integrators for $I_{\mathrm{Ca}(L)}$ Markov chain model: (a) action potential; (b) $I_{\mathrm{Ca}(L)}$; (c) deviation from states conservation law; (d-j) states occupancy in $I_{\mathrm{Ca}(L)}$ MC model; (k) the gating variable q controlling between Mode-Ca and Mode-$\mathrm{V_m}$.

The transition rates matrix is dependent only on voltage which simplifies the tabulation of its eigenvectors and eigenvalues. For the purposes of tabulation we use values of $\mathrm{V_m}$ in a range from -100 mV to 70 mV with a time-step of 0.01 mV.

The solution is obtained using Rush-Larsen method for the gate model, and matrix Rush-Larsen for the Markov chain as

$$q_{n+1} = \frac{\alpha_q}{\alpha_q + \beta_q} - \left(\frac{\alpha_q}{\alpha_q + \beta_q} - q_n \right) \exp\left(-(\alpha_q + \beta_q)\Delta t\right), \tag{4.30}$$

$$\vec{u}_{n+1} = \exp\left(\boldsymbol{D}\Delta t\right) u_n. \tag{4.31}$$

The open probability is obtained as a product of the open state and the gate as $P_{\mathrm{open}} = Oq$.

Figure 4.7 shows results of the integration using the suggested schemes for RyR and $I_{\mathrm{Ca}(L)}$ channels. The states of the model visually overlap except in states on panel (g), (i), and (k).

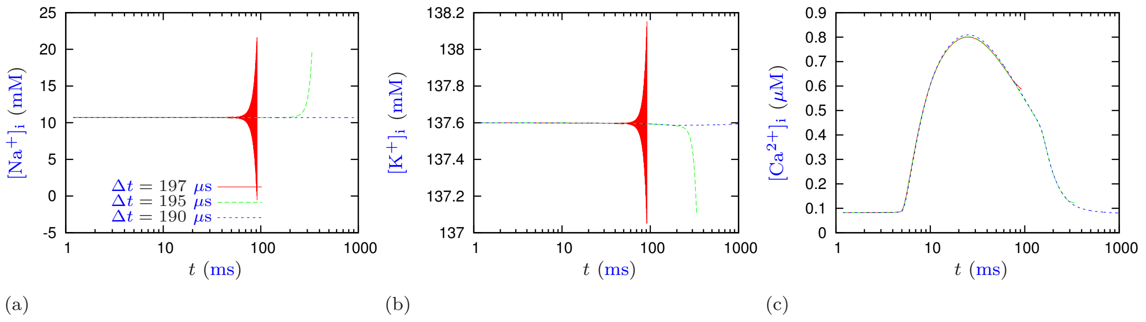


Figure 4.8: Numerical instability caused by computation of intra-cellular ionic concentrations of (a) sodium $[\mathrm{Na}^+]_i$; (b) potassium $[\mathrm{K}^+]_i$; and (c) calcium $[\mathrm{Ca}^{2+}]_i$

Table 4.2: 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		
$I_{\mathrm{Ca}(L)}$	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

4.2.4 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 up to $190\ \mu s$). The cause of the instability at larger time step sizes are the intra-cellular sodium $[\mathrm{Na}^+]_i$ and potassium $[\mathrm{K}^+]_i$ concentrations, that are calculated using Forward Euler method (Figure 4.8). Because, those variables are coupled with other dynamical variables of the cellular model (ion currents), we can not directly apply the MRL methods.

The computational cost is shown in the Table 4.2. The simulation was performed for 100 beats with cycle length of 1000 ms. During this computation the output of the variables was omitted.

The main benefit of applying suggested methods, is the possibility to increase the time step size. This leads to reduction of the computational cost, without the danger that the solver becomes unstable. Our results show, it is possible to increase the time step from $\Delta t = 6\ \mu s$, which was a limit for the stable solution in the forward Euler method, by using matrix Rush-Larsen methods for RyR and $I_{\mathrm{Ca}(L)}$ models, which allow increase of the time step size up to $\Delta t = 180\ \mu s$ without a loss of accuracy.

Chapter 5

Implementation of MRL to BeatBox

The cardiac tissue is a system characterised by equations that describe the change of the concentrations of substances and corresponding alteration of ionic currents and membrane voltage in time. Such system is known as a reaction-diffusion system. The reaction-diffusion system is described by equations for local reactions coupled with equations for diffusion through space. The simplest form of reaction-diffusion system has a form of the following partial differential equation

$$\frac{\partial \vec{u}}{\partial t} = D\nabla^2\vec{u} + \vec{f}(\vec{u}), \tag{5.1}$$

where $\vec{u}$ is a vector of dynamical variables. The first term on the right hand side describes the diffusion system, where D is a diffusion coefficient. The operator

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

is computed for three spacial dimensions x, y, z. The second term $\vec{f}(\vec{u})$ on the right hand side of (5.1) represents local reactions within a cell, i.e. the exchange of ions through the cellular membrane and between internal cellular compartments, and reactions with other substances for specific types of ions. The exchange of ions through the cellular membrane takes place via ionic channels that can be modelled by gate model or Markovian state models.

There is a number of computational packages for solving reaction-diffusion systems. 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[27]. 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 spacial propagation. The space models include regular 1D thread, 2D sheet, and 3D box or even anatomically detailed geometry provided from experimental data.

The Markovian state models within BeatBox were solved using explicit methods which often suffer from instability issues. We have extended BeatBox by a solvers which exploit specific properties of Markov chains as described in the previous chapter. During this project we have developed new parts of the package related to exponential integration of Markov Chain models. This chapter describes the usage of the Matrix Rush-Larsen (MRL) method in BeatBox as well as specifications of required functions.

The Section 5.1 gives a formal definition of the reaction system and its division into subsystems according to the form of the ODEs. The Section 5.2 describes numerical methods used for solving of the subsystems in BeatBox. The Section 5.3 gives a brief overview of the usage of BeatBox package. For detailed description please refer to the BeatBox documentation[28] and to the McFarlane's (2010) thesis[29]. The Section 5.4 describes the implementation of the reaction system into BeatBox in two ways different ways – as `rhs` modules and `ionic` modules. The Section 5.5 describes the procedures within the `euler` and the `rushlarsen` devices used for the solution of `rhs` and `ionic` modules respectively. The Section 5.6 specifies the format of structures in `ionic` modules as used by the extended `rushlarsen` device by matrix Rush-Larsen method. The section 5.7 shows some of the results of the simulations using the extended version of BeatBox.

The modifications of the old code and newly created parts of BeatBox package are marked with a $\ast$ sign in the text of this chapter.

5.1 Definition of Reaction System

There are a variety of methods for the solution of reaction and diffusion part of the system. The computation of the diffusion system is out of the scope of this chapter. By ignoring the equations for the reaction part of the system from the equation (5.1) we obtain ordinary differential equations (ODEs) in the following form

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \vec{f}(\vec{u}), \tag{5.2}$$

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 ionic 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 ionic 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 models has more than one Markov chain, the variable $\vec{z}$ is composed 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 ionic channel.

The ODEs of the whole reaction system (5.2) is reformulated in terms of specific type equations for each of the groups as

$$\frac{\mathrm{d}w^i}{\mathrm{d}t} = \alpha^i(\mathrm{V_m})(1 - w^i) - \beta^i(\mathrm{V_m})w^i, \tag{5.3a}$$

$$\frac{\mathrm{d}x^j}{\mathrm{d}t} = \alpha^j(\vec{y})(1 - x^j) - \beta^j(\vec{y})x^j, \tag{5.3b}$$

$$\frac{\mathrm{d}\vec{y}}{\mathrm{d}t} = \vec{\phi}(\vec{w}, \vec{x}, \vec{y}, \vec{z}), \tag{5.3c}$$

$$\frac{\mathrm{d}\vec{z}^k}{\mathrm{d}t} = \boldsymbol{A}^k(\vec{y})\vec{z}^k, \tag{5.3d}$$

where α’s and β’s 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 of solution of each of the individual groups of the system (5.3) were described in different parts of the thesis. For the readers’ convenience we present the formulas once more in the following section 5.2.

5.2 Solution of Reaction System

5.2.1 Tabulation

The cellular models involve functions in a form $\kappa(r(t))$ dependent on a dynamical variable r, an example of this type of function are the transition rates of gating variables $\alpha(\mathrm{V_m}(t))$ and $\beta(\mathrm{V_m}(t))$. The value of those functions have to be found multiple times during the simulation. In many cases, this computations are done for the same, or very similar value of the input variable r.

To avoid the 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 the simulation start. During the simulation we fetch an approximated functional value from a specific entry in the look-up table as will be described bellow.

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 linear grid defined as $\mathcal{T}_j = \mathcal{T}_{\min} + j\Delta\mathcal{T}$ for table entries $j = 1, 2, \dots, k$. 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 table index $j(n)$ as $\kappa(t_n) \approx \kappa_{j(n)}$ where the index is found as

$$j(n) = \mathrm{round}\left(\frac{\mathcal{T}(t_n) - \mathcal{T}_{\min}}{\Delta\mathcal{T}}\right).$$

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 of entries in the table k as

$$\Delta\mathcal{T} = \frac{1}{k}\ln\left(\frac{r_{\max}}{r_{\min}}\right),$$

and the table index is obtained as

$$j(n) = \text{round}\left(\frac{\ln(r(t_n)) - \ln(r_{\min})}{\Delta\mathcal{T}}\right) = \text{round}\left[\frac{1}{\Delta\mathcal{T}}\ln\left(\frac{r(t_n)}{r_{\min}}\right)\right].$$

The entries of the transition rates 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 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 an extensive computational time and memory resources 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 rates matrix into a sum of a number of “submatrices” depending on a single control variable. If our example the transition rates matrix has a sum form as

$$\boldsymbol{A}^k(\mathrm{V_m}, [\mathrm{Ca}^{2+}]_i) = \boldsymbol{A}_1^k(\mathrm{V_m}) + \boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_i) \tag{5.4}$$

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 $\boldsymbol{A}_1^k(\mathrm{V}_{\mathrm{m}j(n)}) \approx \boldsymbol{A}_1^k(\mathrm{V_m}(t_n))$ on a linear grid of voltage $\mathrm{V_m}$, and $\boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_{ij(n)}) \approx \boldsymbol{A}_2^k([\mathrm{Ca}^{2+}]_i(t_n))$ 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 rates 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 Subsection 2.3.4.

5.2.2 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 (5.2) reads as

$$\vec{u}_{n+1} = \vec{u}_n + \Delta t \vec{f}(\vec{u}_n) \tag{5.5}$$

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 (5.3). The particular time-stepping scheme is obtained by substituting the $\vec{f}(\vec{u})$ in equation (5.5) with the right-hand side of the corresponding equations for each of the groups as

$$w^i_{n+1} = w^i_n + \Delta t\left[\alpha^i(\mathrm{V}_{\mathrm{m}j(n)})(1-w^i_n) - \beta^i(\mathrm{V}_{\mathrm{m}j(n)})w^i_n\right], \tag{5.6a}$$

$$x^j_{n+1} = x^j_n + \Delta t\left[\alpha^j(\vec{y}_n)(1-x^j_n) - \beta^j(\vec{y}_n)x^j_n\right], \tag{5.6b}$$

$$\vec{y}_{n+1} = \vec{y}_n + \Delta t\left[\vec{\phi}(\vec{w}_n, \vec{x}_n, \vec{y}_n, \vec{z}_n)\right], \tag{5.6c}$$

$$\vec{z}^k_{n+1} = \vec{z}^k_n + \Delta t\left[\boldsymbol{A}^k(\vec{y}_n)\vec{z}^k_n\right]. \tag{5.6d}$$

The accuracy of the forward Euler method is of $\mathcal{O}\cdot\sqcup$, and as the $\Delta t \to 0$ the solution converges to 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.

5.2.3 Exponential Integration for Hodgkin-Huxley Type Gates

Exponential integration methods for the Hodgkin-Huxley type gate model was proposed by Rush-Larsen[23]. Their scheme assumes that the transition rates α and β vary slowly and can be "frozen" (assumed to be approximately constant) during one time step. This approximate system system can be solved analytically. The analytic solution of the equation was described in detail in the section 2.2.3, with the final result in equation (2.107).

This result can be used to deduce an iterative numerical method, here presented for gating variables 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 (5.3b) and (5.3a) respectively. The exponential integration scheme then reads as

$$v^i_{n+1} = \frac{\alpha^i(\vec{y}_n)}{\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n)} - \left[\frac{\alpha^i(\vec{y}_n)}{\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n)} - v^i_n\right]\exp\left[-(\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n))\Delta t\right], \tag{5.7}$$

where $\alpha^i(\vec{y}_n)$ and $\beta^i(\vec{y}_n)$ are the values of the transition rates "frozen" at the beginning of the time step. We can include the exponentials into the definition of the coefficients as

$$a(\vec{y}_n, \Delta t) = \frac{\alpha^i(\vec{y}_n)}{\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n)}\left(1-\exp\left[-(\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n))\Delta t\right]\right), \tag{5.8a}$$

$$b(\vec{y}_n, \Delta t) = \exp\left[-(\alpha^i(\vec{y}_n)+\beta^i(\vec{y}_n))\Delta t\right]. \tag{5.8b}$$

The integration method for the two groups of gating variables (5.3b), (5.3a) is

$$w^i_{n+1} = a^i(\mathrm{V}_{\mathrm{m}j(n)}, \Delta t) - b^i(\mathrm{V}_{\mathrm{m}j(n)}, \Delta t)w^i_n, \tag{5.9a}$$

$$x^j_{n+1} = a^j(\vec{y}_n, \Delta t) - b^j(\vec{y}_n, \Delta t)x^i_n, \tag{5.9b}$$

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.

5.2.4 Exponential Integration for Markov Chains

We have suggested and described an exponential integration scheme for Markov chains called matrix Rush-Larsen (MRL) in the Subsection 2.3.3 (final equation (2.130)). Briefly, we assume that the transition rates 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 (5.3d) reads as

$$\vec{z}^k_{n+1} = \exp\left(\boldsymbol{A}^k(\vec{y}_n)\Delta t\right)\vec{z}^k_n. \tag{5.10}$$

The exponential operator matrix can be transformed to

$$\boldsymbol{T}^k(\vec{y}_n, \Delta t) = \exp\left(\boldsymbol{A}^k(\vec{y}_n)\Delta t\right) = \boldsymbol{V}^k(\vec{y}_n)\exp\left[\boldsymbol{\Lambda}^k(\vec{y}_n)\Delta t\right]\boldsymbol{W^T}^k(\vec{y}_n), \tag{5.11}$$

where $\boldsymbol{A}^k = \boldsymbol{V}^k\boldsymbol{\Lambda}^k\boldsymbol{W^T}^k$ is an eigenvalue decomposition of the transition rates 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 $\boldsymbol{I} = \boldsymbol{V}^k\boldsymbol{W^T}^k$.

The motivation to introduce eigenvalue decomposition in (5.10) is a straightforward computation of exponential of diagonal eigenvalue matrix as

$$\exp\left(\begin{bmatrix}\lambda_1\Delta t & 0 & \dots & 0\\ 0 & \lambda_2\Delta t & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & \lambda_N\Delta t\end{bmatrix}\right) = \begin{bmatrix}\exp(\lambda_1\Delta t) & 0 & \dots & 0\\ 0 & \exp(\lambda_2\Delta t) & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & \exp(\lambda_N\Delta t)\end{bmatrix}. \tag{5.12}$$

For the transition rates matrices dependent on a multiple variables such as in the formula (5.4), the iterative scheme uses operator splitting method as

$$\vec{z}^k_{n+1/2} = \exp\left(\boldsymbol{A}^k_1(\mathrm{V}_{\mathrm{m}j(n)})\Delta t\right)\vec{z}^k_n, \tag{5.13a}$$

$$\vec{z}^k_{n+1} = \exp\left(\boldsymbol{A}^k_2([\mathrm{Ca}^{2+}]_{ij(n)})\Delta t\right)\vec{z}^k_{n+1/2}. \tag{5.13b}$$

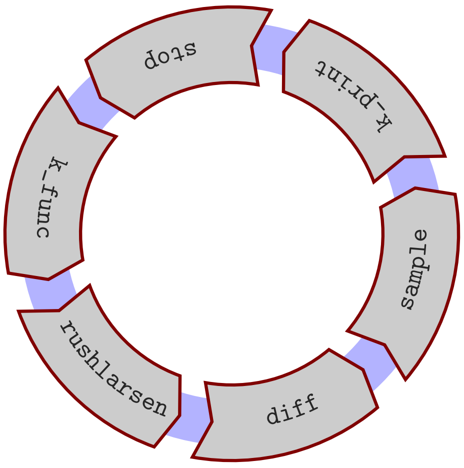


Figure 5.1: Conceptual "ring of devices". The ring is constructed from the instructions provided in the bbs script. Each revolution of the ring corresponds to one time step.

Before the simulation starts we create a look-up tables for the eigenvalues and eigenvector matrices of $\boldsymbol{A}_1^k$ and $\boldsymbol{A}_2^k$. The tables are then be used in the numerical scheme similarly to (5.11).

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 overweight the cost of eigenvalue decomposition at each time step.

5.3 Running BeatBox Simulation

The design of BeatBox follows a modular paradigm which allows to construct 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. BeatBox reads the bbs script and constructs the simulation according to the commands and specific calls of the devices.

BeatBox processes the bbs script to set up a "ring of devices" – a conceptual arrangement of the devices as shown on Figure 5.1. 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 set in the bbs script independently for each instance of a device.

The execution of a device can also be conditional to a particular spacial position of the cell within the tissue, and specific time of the simulation.

A scripting guide describing how the bbs script is processed and including details about generic device parameters can be found in the BeatBox documentation. The section 5.5 specifies the parameters of two most commonly used devices for the computation of reaction system – `euler` and `rushlarsen`.

5.4 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.).

5.4.1 `rhs` Modules

A `rhs` (right-hand side) modules implement right-hand side of the system described by equation (5.2) i.e. the kinetics of all dynamical equations, within one C-function.

The `rhs` format is quite generic and straightforward to implement. However, the `rhs` format lacks an information about the role of the variables. Therefore, it does not allow the possibility of exploring “hybrid” integration methods which treat the variables by different scheme according to their role and form. This limitation is addressed in `ionic` modules described in the following subsection.

For illustration a minimal working example of `rhs` module for Hodgkin-Huxley squid axon is listed in the Appendix section B.2.4.

5.4.2 `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 form of the equations describing the system.

The division which is used by `ionic` modules correspond to the group of equations in (5.3) i.e. tabulated and non-tabulated gating variables, “other” variables, and Markov chains. 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 sub-system (5.3c) 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 (5.3b).

The voltage-dependent transition rates of the gating variables specified by equation (5.3a) are implemented in another C-function within `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 the Subsection 5.2.1.

Unlike the `rhs` module 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 Hodgkin-Huxley squid axon model as `ionic` module is listed in the Appendix section B.2.4.

5.4.3 Extension of `ionic` Modules *

Previously, no specific C-functions to describe Markov chains were present in `ionic` modules. When the user wished 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.

The integration by explicit methods in this case can bring difficulties as many of the Markov chains are numerically stiff. In such models, an artefact called numerical instability appears when the time step is above a certain threshold. The numerical instability renders the solution unusable, so the time step size is limited by the threshold. Due to the small time steps, we are forced to use in such models, the computational cost is high. To address the issue of numerical instability and so allow larger time steps leading to the reduction of computational cost, we have suggested the use of exponential solvers.

The `ionic` modules are an ideal target for the implementation of Markov chains. We have implemented the exponential solver for Markov chains into BeatBox and here we describe the details of the implementation.

Each of them Markov chains in the ionic module (as specified by the equation (5.3d)) is split into a number of subchains according to the control variable on which the transition rates depend (as in operator splitting in equation (5.4)). The ionic module implements a separate C-function of each subchain to fill the entries of the corresponding transition rates 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 Hodgkin-Huxley squid axon model with the ionic channels converted into Markov chain description can be found in the Appendix section B.2.4 .

5.5 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 an 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.

5.5.1 euler Device

Overview 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.

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 the Table 5.1.

Parameters of euler Device

The parameter ht defines a duration of one time step used in forward Euler calculations. This corresponds to the Δt in the equation (5.5).

The parameter ode is a name of 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

Table 5.1: Parameters to set up euler device in bbs script (adapted from [30] *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.

specified in the module code for the number of steps specified in `rest`. This aims to approximate steady-state conditions.

Model dependent parameters of `rhs` module are passed to `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 into the "ring of devices" of BeatBox simulation. An example of a usage of `euler` device can be found in `bbs` script listed in Listing B.13. The parts relevant for the call of the `euler` device are:

Listing 5.1: 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. The `v0`, and `v1`, for instance, specify the number of the lower and the upper layer number bound `v`. More details on the generic device conditions can be found in the BeatBox documentation.

5.5.2 `rushlarsen` Device

Overview `rushlarsen` Device

The `rushlarsen` device is an integration device for the `ionic` models. Recall that 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 input and output of the C-functions implementing Markov chain ion channel models are specified as part of this project. 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 rates 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 simulated value of 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 rushlarsen device is shown in the Table 5.2. The horizontal lines separate corresponding parameters to euler device (as described in Table 5.1) from the specific parameters for 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 rushlarsen device are analogical 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 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 on 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 rest parameter.

Specific Parameters of rushlarsen

The following part describes bbs script parameters specific to rushlarsen device i.e. parameters which do not have an analogy in 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

Table 5.2: Parameters to set up rushlarsen device in bbs script (adapted from [30] *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 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 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)

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 specifies 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 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. Namely 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 be 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*

Parameters for the integration of Markov chains `exp_mc` can be provided in a bbs script. The possible values are `mcfe` for forward Euler, `tabmrl` for MRL with tabulation, and `ntabmrl` for MRL without tabulation. However, the method used on a particular subchain depends on other characteristics of the subchain as will be explained bellow.

Running `rushlarsen`

The `rushlarsen` device can be run during simulation in BeatBox by setting up a corresponding line in a bbs script. An example of setup of `rushlarsen` device can be found in the bbs script listed in Listings B.19, and B.23 for a case with and without Markov chain models. The relevant parts are:

Listing 5.2: 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};
```

5.6 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 the subsection 5.4.2.

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 `rushlarsen` device using forward Euler and exponential methods. This section also specifies some of the features of `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 operates on the data structures. The C-functions are implemented within the cellular modules.

A minimalist example of `ionic` code is shown in Appendix subsections B.2.4 and B.2.4.

5.6.1 Data Structures

An implementation of data structures in BeatBox is in a hierarchical way. A diagram on Figure 5.2 shows the data structure used by `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 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 5.2). 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.

The detailed information about the elements initialised from `bbs` script can be found in the Table 5.2. The other elements of the data structures are used to participate on internal tasks of the `rushlarsen` device such as hold intermediate results, look up tables, or translation a `bbs` script variables in a different type.

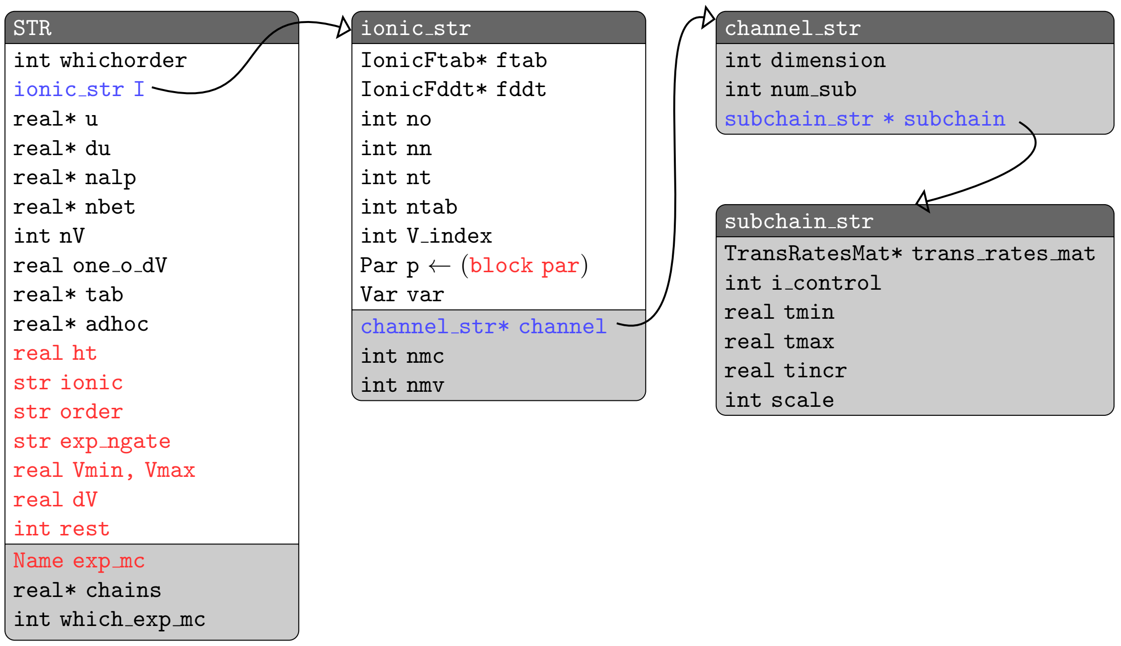


Figure 5.2: Data structures in rushlarsen device. Elements accepted from bbs script (red font); elements specifying underlying substructures (blue font, with arrow); new elements introduced for the implementation of Markov chain models (grey background).

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 5.3 describes the elements of the structure.

The element trans_rates_mat is a pointer to a C-function <ionic>_<subchain> which computes the transition rates matrix of the subchain. The functionality of this C-function is described in the following subsubsection.

The element i_control is an index of the control variable within the state array u which controls the transition rates matrix. For instance, if the transition rates

Table 5.3: Elements of a data structure subchain_str describing subchains of a Markov chain.

type	name	description
TransRatesMat*	trans_rates_mat	C-function computing transition rates 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 linear, 1 for logarithmic (must be reflected in TransRatesMat)

Table 5.4: 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

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 when the tabulation for the subchain should be disabled. In this case the transition rates matrix is computed on-the-fly. When the transition rates 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 linear grid and a value of 1 means a logarithmic grid. The tabulation for the linear and the logarithmic scale was described in the Subsection 5.2.1.

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 linear scale for scale=0 or logarithmic scale 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 in the BeatBox repository and is currently set to 4 subchains. Elements of the channel_str data structure are summarised in the Table 5.4.

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.

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 the Table 5.5.

The element ftab refer to C-functions computing the "tabulated" transition rates of gating variables. The element fddt refers to a C-function which computes

Table 5.5: Elements of data structure ionic_str of an ionic module (adapted from [30]).

type	name	description
IonicFtab*	ftab	C-function of voltage-dependent functions that are tabulated
IonicFddt*	fddt	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	sum of variables in all Markov chains

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.

Dependent parameters are given by structure var which saves the conditions of specific parameters that depend on spacial 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 saves elements which help to set up the simulation protocol within rushlarsen device and save intermediate results of the calculations. The summary of the elements of STR structure is given in the Table 5.6.

Table 5.6: Elements of data structures of rushlarsen device (adapted from [30]).

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's
real*	nbet	array of non-tabulated transition rates β's
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)

The element whichorder is used to save 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 vector of all dynamical variables is saved in the array u. The time derivative of “other” non-gating variables is saved in the array du. 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 non-tabulated transition rates are saved in the nalp and nbet arrays 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 element chains is an array containing the tabulated matrices of Markov chains. Depending on the method used, it contains or the transition rates 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 rates matrices, and value 2 corresponds to matrix Rush-Larsen with computation of transition rates 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.

5.6.2 C-Functions and Template Macros

The data structure of rushlarsen device refer to a number of C-functions defined within ionic modules. This subsections describes the C-functions to provide a necessary information for implementation of an 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 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 the Table 5.7.

The IONIC_CREATE_HEAD(<ionic>) and IONIC_CREATE_TAIL(<ionic>) macros define a IonicCreate C-function. Once the macros have been expanded the name of the C-function becomes create_<ionic>.

Table 5.7: 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 (5.3c) and computing the non-tabulated transition rates of gates corresponding to α^j and β^j in equation (5.3b). The input arguments of fddt_<ionic> C-function are specified in Table 5.8.

The IONIC_FDDT_HEAD(<ionic>,NV,NTAB,NO,NN) and IONIC_FDDT_TAIL(<ionic>) macros define 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 5.8: Input arguments of the C-function computing the kinetics of non-gating and non-tabulated transition rates of gating variables in 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's
real *	nbet	array of non-tabulated β's
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 the Table 5.9.

The IONIC_FTAB_HEAD(<ionic>) and IONIC_FTAB_HEAD(<ionic>) macros define IonicFtab C-function. Once the macros have been expanded the name of the C-function becomes ftab_<ionic>.

Table 5.9: 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 Markov chain model.

The C-function `<ionic>_<subchain>` is a 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 rates matrix. The input arguments of the C-function `<ionic>_<subchain>` are specified in the Table 5.10.

The macro `CHANNEL_TR_MATRIX(<ionic>_<subchain>)` defines `TransRatesMat` C-function. After the macro has been expanded the name of the C-function becomes `<ionic>_<subchain>`.

Table 5.10: 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 rates 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 rates matrix.

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 suggest the first two arguments specify two states `from`, `to` between which the transition rates are defined in `direct`, and `reverse` expressions.

The Figure 5.3 illustrates the process of filling the entries of the transition rates matrix. The panel (a) shows the form of the function-like macro with arguments (description of function-like macros can be found on page 152). 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 given function-like macro is shown on the panel (b). The panel (c) shows the corresponding entries in the transition rates 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 rates matrix, we

```
_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)

$$\begin{array}{c|cccc|}
 & S_1 & \texttt{<channel>\_from} & \texttt{<channel>\_to} & S_2 \\
\hline
S_1 & \ddots & \dots & 0 & 0 \\
\texttt{<channel>\_from} & \dots & -(\texttt{direct}+\dots) & \texttt{reverse} & 0 \\
\texttt{<channel>\_to} & 0 & \texttt{direct} & -(\texttt{reverse}+\dots) & \dots \\
S_2 & 0 & 0 & \dots & \ddots
\end{array}$$

(c)

Figure 5.3: Construction of the transition rates 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 rates matrix constructed from the function-like macro.

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* provides also 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 to 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 `ionic` module with the definitions of two Markov chain models can be found in Appendix section B.2.4.

5.7 Testing of BeatBox Cellular Modules

For the sake of implementation and testing of 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)[1] cellular model including of I_{Na} channel. This model can be implemented as `rhs` module, however not as `ionic` module due to (a) 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 overcame by identical 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 familiarity of the readers 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)[2] which is a mammalian ventricular AP that includes stiff Markov chains.

5.7.1 Hodgkin-Huxley Minimalist Model

The code of a minimalist model was implemented in a standalone version, and as three BeatBox module one in `rhs` format and two in `ionic` format: the first in the original form with Hodgkin-Huxley gate formulation of ionic channels, the second convert the gate models as the equivalent models in 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 B.

The Figure 5.4 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 `rhs` module with a small time step $\Delta t = 0.1\ \mu\mathrm{s}$ (denoted as #`rhs`) compared with the simulations using the `rhs` module, and both `ionic` modules with larger time step $\Delta t = 1\ \mu\mathrm{s}$ (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

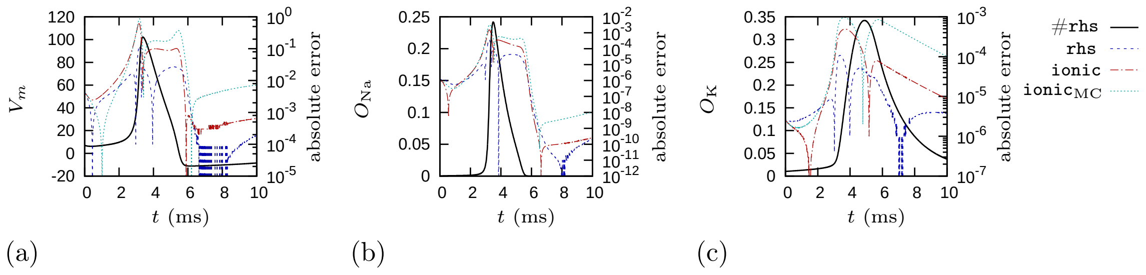


Figure 5.4: Simulation results and absolute error of Hodgkin-Huxley squid model in BeatBox: (a) membrane voltage $\mathrm{V_m}$, (b) open probability of Na^+ channels O_{Na}, (c) open probability of K^+ channels O_{K}. Black solid line shows the “accurate” simulated traces of `rhs` model with $\Delta t = 0.1\ \mu\mathrm{s}$ (left axis); the remaining lines show the difference between “accurate” simulation and simulations with time step $\Delta t = 1\ \mu\mathrm{s}$ using `rhs` model (blue long-dashed lines), `ionic` model with gate model of ionic channels, `ionic` model with ionic channels converted to equivalent Markov chain model. Ionic models were solved using exponential integration without tabulation (transition rates computed on-the-fly).

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 splitting of the gating and the Markov chain variables from membrane voltage. The absolute error in both `ionic` models visually overlaps, which suggests that the gate and Markov chain models of ionic channels are equivalent.

The absolute difference of the solutions from both `ionic` modules remains bellow 10^{-11} for both open probabilities O_{Na} and O_{K} and bellow 10^{-8} for the membrane voltage. This the 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 rates matrix).

The minimalist model described in this subsection is a demonstration of the functionality of a 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.

5.7.2 TenTusscher-Panfilov (2006) Model

The demonstration of the functionality of the `rushlarsen` device is done using TenTusscher-Panfilov (2006) human ventricular model (TTP)[31]. 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

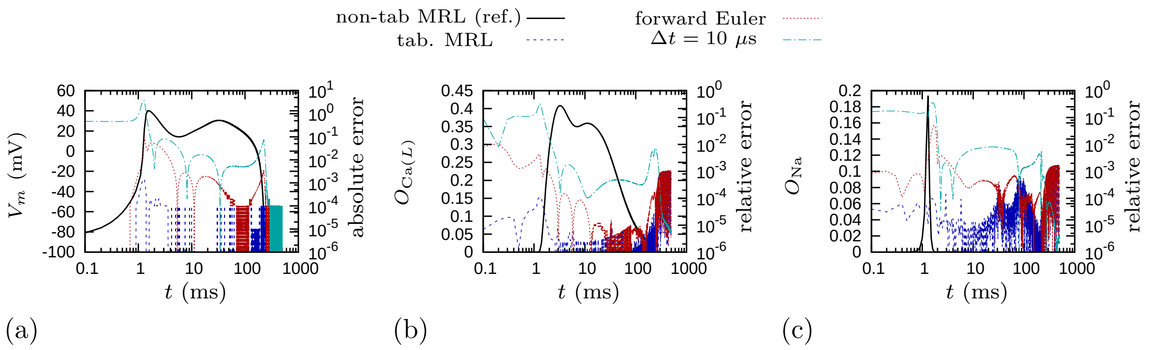


Figure 5.5: Comparison of the simulation methods of the Markov chain models for TTP model. Panel (a) shows membrane voltage $\mathrm{V_m}$, panel (b) shows open probability $I_{\mathrm{Ca}(L)}$, and panel (c) shows open probability I_{Na} currents. Black lines show the reference solution for the Matrix Rush-Larsen (MRL) method at the time-step $\Delta t = 1\ \mu$s, coloured lines show the relative error of the reference solution (left axis) with: the MRL method with tabulated transition rates matrix (blue lines), the forward Euler solution (red lines), both methods use the same time-step as the reference solution $\Delta t = 1\ \mu$s; cyan lines show the simulations with the same method as the reference solution with the time-step $\Delta t = 10\ \mu$s.

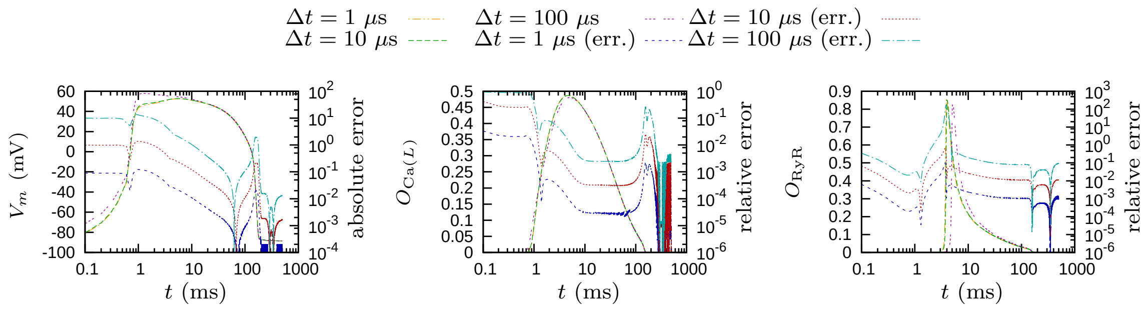
done for the sake of testing of the numerical methods developed into BeatBox. Previously we have used MRL method for currents of calcium channel $I_{\mathrm{Ca}(L)}$[2], Ryanodine Receptor (RyR)[2] and sodium channel I_{Na} [1]. We have decided to implement two of those Markov chain models into BeatBox TTP module, namely the $I_{\mathrm{Ca}(L)}$ and I_{Na} models.

The BeatBox distribution already contained `rhs` definition of TTP model. Before we proceed to the conversion to `ionic` format, we need to replace the implementation of some of the calcium concentrations which 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 the Appendix B.2.5.

The ttp `ionic` module was implemented in BeatBox. Further details about the conversion from `rhs` module can be found in appendix Section B.2.5.

The Figure 5.5 shows the open probability of the Markov chains. The open probabilities were used for the computation of I_{Na} and $I_{\mathrm{Ca}(L)}$ ionic currents. The gating variables of Hodgkin-Huxley type gates were computed using Rush-Larsen method. The voltage-dependent coefficients in the Rush-Larsen method were tabulated with the tabulation step $\Delta V_{m,tab.} = 0.01\ \mathrm{mV}$. The reference solution was obtained using the MRL method. The discrepancy of the MRL solution with tabulation show a small error comparable with corresponding tabulation inaccuracy for tabulated gating variables (e.g. in Figure B.2).

The discrepancy from the forward Euler solution is negligible compared to the error due to the increase of the time-step size to $\Delta t = 10\ \mu$s in the the same



(a) (b) (c)

Figure 5.6: Comparison of simulations with exponential methods using Faber et al. (2007) model. Panel (a) shows membrane voltage V_m, panel (b) shows open probability of $I_{Ca(L)}$, and panel (c) shows open probability of RyR currents. The reference solution (not shown) was computed at the time step $\Delta t = 0.1\ \mu s$ using the tabulation for the transition rates matrices (`tabmrl`). The simulation using `tabmrl` at the time steps Δt = 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.

method as reference solution (non tabulated MRL). The error due to the time-step increase will be discussed in detail in the following subsection.

5.7.3 Faber et al. (2007) model

In the previous chapter we have used Faber et al. (2007) model. Although, the Faber et. al (2007) model was published by the same lab as 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 $I_{Ca(L)}$ (which we used in the previous subsection within TTP model), and RyR model. In the previous chapter we have developed exponential integration of the RyR and $I_{Ca(L)}$ models within the standalone code of Faber et. al (2007). Here we aim to extend the BeatBox framework to this model, which required further modifications of `rushlarsen` device.

The exponential integration of the RyR model was more challenging. Unlike the I_{Na} and $I_{Ca(L)}$ models, that contained only voltage-dependent transition rates in the operator matrix, the transition rates matrix of the RyR model depends on multiple variables corresponding to intracellular calcium concentrations. The transition rates 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 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 the Section 4.2.2.

Table 5.11: Computational time in seconds for simulation of 500 ms in Faber et al.[2]

method `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

The Figure 5.6 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 value of the error is rather high e.g. almost up to 100 mV for the membrane voltage. However, this discrepancy is due to fast change in the action potential onset at the initiation of the action potential, and is compensated in the later phases of the simulation. The same error propagates to other dynamical variables.

The Table 5.11 lists the computational time required for a simulations of 500 ms of simulation time 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 spend by the 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 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 can not be used. Meanwhile the MRL method still provide 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 pre-computing corresponding look-up tables.

Chapter 6

Conclusions

The main results of this work are:

- Development of asymptomatic methods for dimensionality reduction of generic Markov chain, including the correction term for improved accuracy of the reduced system.
- Development of numerical method called Matrix Rush-Larsen (MRL) for the Markov chain models of ionic channels. MRL maintains stability at larger time step and improve accuracy. The time step increase leads to lower computational cost.
- Theoretical assessment of the accuracy of the numerical integration methods and operator splitting methods for hybrid integration used for the Markov chain models.
- Parameter embeddings and application of methods of dimensionality reduction to Markov chain model of I_{Na} channel.
- Application of MRL method and hybrid analytic-numerical approach based on operator splitting for the solution of I_{Na} model.
- Testing convergence of MRL and hybrid methods and practical testing of computational cost in I_{Na} model.
- Applications of MRL methods to stiff RYR and $I_{\mathrm{Ca}(L)}$ models to Faber et al. (2007), which allows 30 fold reduction of the computational cost.
- Implementation of MRL and hybrid methods to BeatBox package for cardiac simulations. Specifically, to `rushlarsen` device for solving cellular models in `ionic` format.
- Implementation of minimalist Hodgkin-Huxley model in BeatBox format including Markov chain ionic channels.

- Translation of other popular cellular models to `ionic` format to benefit from the new numerical methods in the `rushlarsen` device.
- Including the code to the BeatBox distribution and documentation of the developed methods and models.

Possible directions of future work include:

- Testing the methods within spatial 1D, 2D and 3D models of cardiac tissue.
- Implementation of second order scheme for matrix Rush-Larsen method.

Appendix A

Definition of Clancy-Rudy (2002) Model

A.1 Cell Model Definition

The following section contains the definition of the model according to the author's code[1]. The format of equations and subsections aims to correspond to the papers where those equations were published to facilitate a straightforward comparison. The differences with the papers are marked by the sign $\#$.

Standard ionic concentrations

$$[\mathrm{Na}^+]_i = 7.9 \quad \#\text{(initial value of dynamical variable)} \tag{A.1}$$

$$[\mathrm{Na}^+]_o = 140 \quad \# \tag{A.2}$$

$$[\mathrm{K}^+]_i = 147.23 \quad \#\text{(initial value of dynamical variable)} \tag{A.3}$$

$$[\mathrm{K}^+]_o = 4.5 \quad \# \tag{A.4}$$

$$[\mathrm{Ca}^{2+}]_o = 1.8 \tag{A.5}$$

which differs from the [32] 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

$$x_{s1} = 0\text{? not initialised} \tag{A.6}$$

$$x_{s2} = 0\text{? not initialised} \tag{A.7}$$

$$V = -95 \tag{A.8}$$

$$[\mathrm{Ca}^{2+}]_{\mathrm{NSR}} = 1.8 \tag{A.9}$$

$$[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} = 1.8 \tag{A.10}$$

$$[\mathrm{Ca}^{2+}]_i = 0.00012 \tag{A.11}$$

$$b = 0.00141379 \tag{A.12}$$

$$g = 0.98831 \tag{A.13}$$

$$d = 6.17507 \cdot 10^{-6} \tag{A.14}$$

$$f = 0.999357 \tag{A.15}$$

$$X_r = 2.14606 \cdot 10^{-4} \tag{A.16}$$

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[32]

$I_{\mathrm{K}s}$, the Slow Component of the Delayed Rectifier K^+ Current

$$I_{\mathrm{K}s} = \bar{G}_{\mathrm{K}s} x_{s1} x_{s2} (V - E_{\mathrm{K}s}) \tag{A.31}$$

$$E_{\mathrm{K}s} = (RT/F) \log((4.5 + P_{\mathrm{NaK}} 150)/([\mathrm{K}^+]_i + P_{\mathrm{NaK}} [\mathrm{Na}^+]_o)) \quad \# \tag{A.32}$$

$$P_{\mathrm{NaK}} = 0.01833 \tag{A.33}$$

$$\bar{G}_{\mathrm{K}s} = (0.433(1 + 0.6/(1 + (0.000038/[\mathrm{Ca}^{2+}]_i)^{1.4}))) \cdot 0.615 \tag{A.34}$$

$$x_{s1\infty} = 1/(1 + \exp(-(V - 1.5)/16.7)) \tag{A.35}$$

$$x_{s2\infty} = x_{s1\infty} \tag{A.36}$$

$$\tau_{xs1} = \left(\frac{7.19 \cdot 10^{-5}(V + 30)}{1 - \exp(-0.148(V + 30))} + \frac{1.31 \cdot 10^{-4}(V + 30)}{\exp(0.0687(V + 30)) - 1} \right)^{-1} \tag{A.37}$$

$$\tau_{xs2} = 4\tau_{xs1} \tag{A.38}$$

the definition of $E_{\mathrm{K}s}$ equation (A.32) differs from the Viswanathan et al. (1999) [33] by the hard coded term for the $[\mathrm{K}^+]_o = 4.5$ and $[\mathrm{Na}^+]_o = 150$ rather than values defined by equations (A.4) (A.2) where $[\mathrm{K}^+]_o = 4.5$ and $[\mathrm{Na}^+]_o = 140$.

To simulate the intramural heterogeneity the definition of $\bar{G}_{\mathrm{K}s}$ is multiplied by 0.615

Otherwise the $I_{\mathrm{K}s}$ definition is identical to Viswanathan et al. (1999)[33].

$$\frac{\mathrm{d}x_{s1}}{\mathrm{d}t} = \frac{x_{s1\infty} - x_{s1}}{\tau_{xs1}} \tag{A.39}$$

$$\frac{\mathrm{d}x_{s2}}{\mathrm{d}t} = \frac{x_{s2\infty} - x_{s2}}{\tau_{xs2}} \tag{A.40}$$

$I_{\mathrm{K}r}$, 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)[34]. The original notation for $R_{\mathrm{K}r}$ is R (here the R is used for the gas constant).

$$\frac{\mathrm{d}X_r}{\mathrm{d}t} = \frac{X_{r\infty} - X_r}{\tau_{xr}} \tag{A.47}$$

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[32].

$$\mathrm{K1}_{\infty} = \alpha_{\mathrm{K1}}/(\alpha_{\mathrm{K1}} + \beta_{\mathrm{K1}}) \tag{A.53}$$

$$\tag{A.54}$$

Plateau K^+ current: $I_{\mathrm{K}p}$

$$I_{\mathrm{K}p} = 0.00552K_p(V - E_{\mathrm{K1}}) \tag{A.55}$$

$$K_p = 1/(1 + \exp((7.488 - V)/5.98)) \tag{A.56}$$

equivalent to Luo-Rudy[32] with update from Zeng et al. (1995)[34].

$$i_{\mathrm{K}} = I_{\mathrm{K1}} + I_{\mathrm{K}p} \tag{A.57}$$

Currents through the L-type Ca^{+2} channel $I_{\mathrm{Ca}L}$

$$I_{\mathrm{Ca}L} = I_{\mathrm{Ca}} + I_{\mathrm{CaK}} + I_{\mathrm{CaNa}} \tag{A.58}$$

$$I_{\mathrm{Ca}} = dff_{\mathrm{Ca}}\bar{I}_{\mathrm{Ca}} \tag{A.59}$$

$$I_{\mathrm{CaK}} = dff_{\mathrm{Ca}}\bar{I}_{\mathrm{CaK}} \tag{A.60}$$

$$I_{\mathrm{CaNa}} = dff_{\mathrm{Ca}}\bar{I}_{\mathrm{CaNa}} \tag{A.61}$$

$$\bar{I}_{\mathrm{Ca}} = P_{\mathrm{Ca}}{z_{\mathrm{Ca}}}^2\frac{(VF^2)}{RT} \cdot \frac{\gamma_{\mathrm{Ca}i}[\mathrm{Ca}^{2+}]_i\exp((z_{\mathrm{Ca}}VF)/(RT)) - \gamma_{\mathrm{Ca}o}[\mathrm{Ca}^{2+}]_o}{\exp((z_{\mathrm{Ca}}VF)/(RT)) - 1} \tag{A.62}$$

$$\bar{I}_{\mathrm{CaNa}} = P_{\mathrm{Na}} {z_{\mathrm{Na}}}^2 \frac{(VF^2)}{RT} \cdot \frac{\gamma_{\mathrm{Na}i}[\mathrm{Na}^+]_i \exp((z_{\mathrm{Na}}VF)/(RT)) - \gamma_{\mathrm{Na}o}[\mathrm{Na}^+]_o}{\exp((z_{\mathrm{Na}}VF)/(RT)) - 1} \tag{A.63}$$

$$\bar{I}_{\mathrm{CaK}} = P_{\mathrm{K}} {z_{\mathrm{K}}}^2 \frac{(VF^2)}{RT} \cdot \frac{\gamma_{\mathrm{K}i}[\mathrm{K}^+]_i \exp((z_{\mathrm{K}}VF)/(RT)) - \gamma_{\mathrm{K}o}[\mathrm{K}^+]_o}{\exp((z_{\mathrm{K}}VF)/(RT)) - 1} \tag{A.64}$$

$$P_{\mathrm{Ca}} = 5.4 \cdot 10^{-4} \qquad \gamma_{\mathrm{Ca}i} = 1 \qquad \gamma_{\mathrm{Ca}o} = 0.341 \tag{A.65}$$

$$P_{\mathrm{Na}} = 6.75 \cdot 10^{-7} \qquad \gamma_{\mathrm{Na}i} = 0.75 \qquad \gamma_{\mathrm{Na}o} = 0.75 \tag{A.66}$$

$$P_{\mathrm{K}} = 1.93 \cdot 10^{-7} \qquad \gamma_{\mathrm{K}i} = 0.75 \qquad \gamma_{\mathrm{K}o} = 0.75 \tag{A.67}$$

$$f_{\mathrm{Ca}} = 1/(1 + [\mathrm{Ca}^{2+}]_i / K_{m\mathrm{Ca}}) \tag{A.68}$$

$$K_{m\mathrm{Ca}} = 0.0006 \tag{A.69}$$

$$d_\infty = 1/(1 + \exp(-(V + 10)/6.24)) \tag{A.70}$$

$$\tau_d = d_\infty (1 - \exp(-(V + 10)/6.24))/(0.035(V + 10)) \tag{A.71}$$

$$f_\infty = (1/(1 + \exp((V + 32)/8))) + (0.6/(1 + \exp((50 - V)/20))) \quad \# \tag{A.72}$$

$$\tau_f = 1/(0.0197 \exp(-(0.0337(V + 10)^2)) + 0.02) \tag{A.73}$$

the (A.72) differs from Luo-Rudy. The expression from the code: $\exp((V + 32)/8)$ is $\exp((V + 32)/8.6)$ in Luo, Rudy (1994)[32]. Otherwise, this equations are exactly the same as in Luo-Rudy model [32].

$$z_{\mathrm{Na}} = 1 \tag{A.74}$$

$$z_{\mathrm{K}} = 1 \tag{A.75}$$

$$z_{\mathrm{Ca}} = 2 \tag{A.76}$$

$$\frac{\mathrm{d}d}{\mathrm{d}t} = \frac{d_\infty - d}{\tau_d} \tag{A.77}$$

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{f_\infty - f}{\tau_f} \tag{A.78}$$

Ca^{2+} Current Through T-Type Ca^{2+} Channels $I_{\mathrm{Ca}(T)}$[34]

$$I_{\mathrm{Ca}(T)} = \bar{G}_{\mathrm{Ca}(T)} b^2 g (V - E_{\mathrm{Ca}}) \tag{A.79}$$

$$\bar{G}_{\mathrm{Ca}(T)} = 0.05 \tag{A.80}$$

$$b_\infty = 1/(1 + \exp(-(V + 14)/10.8)) \tag{A.81}$$

$$g_\infty = 1/(1 + \exp((V + 60)/5.6)) \tag{A.82}$$

$$E_{\mathrm{Ca}} = (RT/(2F)) \log([\mathrm{Ca}^{2+}]_o / [\mathrm{Ca}^{2+}]_i) \tag{A.83}$$

$$\tau_b = 3.7 + 6.1/(1 + \exp((V + 25)/4.5)) \tag{A.84}$$

$$\tau_g = -0.875V + 12 \text{ for: } V \leq 0; \text{ and } \tau_g = 12 \text{ for: } V > 0 \tag{A.85}$$

which correspond exactly to Zeng et al. (1995) [34].

$$\frac{\mathrm{d}b}{\mathrm{d}t} = \frac{b_\infty - b}{\tau_b} \tag{A.86}$$

$$\frac{\mathrm{d}g}{\mathrm{d}t} = \frac{g_\infty - g}{\tau_g} \tag{A.87}$$

Na^{+}-Ca^{+} exchanger: I_{NaCa}

$$I_{\mathrm{NaCa}} = \frac{2.5\cdot 10^{-4}\exp((\eta-1)V\frac{F}{RT})\exp(V\frac{F}{RT})[\mathrm{Na}^+]_i{}^3[\mathrm{Ca}^{2+}]_o - [\mathrm{Na}^+]_o{}^3[\mathrm{Ca}^{2+}]_i}{1 + 1\cdot 10^{-4}\exp((\eta-1)V\frac{F}{RT})(\exp(V\frac{F}{RT})[\mathrm{Na}^+]_i{}^3[\mathrm{Ca}^{2+}]_o + [\mathrm{Na}^+]_o{}^3[\mathrm{Ca}^{2+}]_i)} \quad \# \tag{A.88}$$

$$\eta = 0.15 \quad \# \tag{A.89}$$

The definition of I_{NaCa} in the Luo-Rudy model[32] depends on concentrations $[\mathrm{Ca}^{2+}]_o$, $[\mathrm{Na}^+]_o$ only. Whereas in this model it depends also on intra-cellular concentrations $[\mathrm{Na}^+]_i$, $[\mathrm{Ca}^{2+}]_i$.

Nonspecific Ca^{2+}-activated current: $I_{ns(\mathrm{Ca})}$

$$\bar{I}_{ns\mathrm{K}} = 1.75\cdot 10^{-7}\frac{VF^2}{RT}\cdot\frac{0.75[\mathrm{K}^+]_i\exp((VF)/(RT)) - 0.75[\mathrm{K}^+]_o}{\exp(VF/(RT)) - 1} \tag{A.90}$$

$$I_{ns\mathrm{K}} = \bar{I}_{ns\mathrm{K}}\frac{1}{1 + (0.0012/[\mathrm{Ca}^{2+}]_i)^3} \tag{A.91}$$

$$\bar{I}_{ns\mathrm{Na}} = 1.75\cdot 10^{-7}\frac{VF^2}{RT}\cdot\frac{0.75[\mathrm{Na}^+]_i\exp((VF)/(RT)) - 0.75[\mathrm{Na}^+]_o}{\exp(VF/(RT)) - 1} \tag{A.92}$$

$$I_{ns\mathrm{Na}} = \bar{I}_{ns\mathrm{Na}}\frac{1}{1 + (0.0012/[\mathrm{Ca}^{2+}]_i)^3} \tag{A.93}$$

$$I_{ns(\mathrm{Ca})} = I_{ns\mathrm{K}} + I_{ns\mathrm{Na}} \tag{A.94}$$

$$P_{ns(\mathrm{Ca})} = 1.75\cdot 10^{-7} \tag{A.95}$$

This seems to be identical to Luo-Rudy model[32], where the extra definition of $E_{ns(\mathrm{Ca})}$ is unused.

Sarcolemmal Ca^{+2} pump: $I_{p(\mathrm{Ca})}$

$$I_{p(\mathrm{Ca})} = 1.15\frac{[\mathrm{Ca}^{2+}]_i}{0.0005 + [\mathrm{Ca}^{2+}]_i} \tag{A.96}$$

identical to Luo-Rudy model[32].

Ca^{+2} background current: $I_{\mathrm{Ca}b}$

$$I_{\mathrm{Ca}b} = 0.003016(V - E_{\mathrm{Ca}}) \tag{A.97}$$
$$E_{\mathrm{Ca}} = RT/(2F)\log([\mathrm{Ca}^{2+}]_o/[\mathrm{Ca}^{2+}]_i) \tag{A.98}$$

identical to Luo-Rudy model[32].

Na^{+} background current: $I_{\mathrm{Na}b}$

$$I_{\mathrm{Na}b} = 0.00141(V - E_{\mathrm{Na}}) \tag{A.99}$$

identical to Luo-Rudy model[32].

Ca^{2+} uptake and leakage of NSR: I_{up} and I_{leak}

$$I_{up} = 0.00875[\mathrm{Ca}^{2+}]_i/([\mathrm{Ca}^{2+}]_i + 0.00092) \tag{A.100}$$
$$K_{leak} = 0.005/15 \tag{A.101}$$
$$I_{leak} = K_{leak}[\mathrm{Ca}^{2+}]_{\mathrm{NSR}} \tag{A.102}$$

the definition of I_{up} in the Luo, Rudy (1994)[32] is ambiguous. This version is consistent with one possible understanding.

Ca^{+2} Fluxes in NSR

$$\frac{\mathrm{d}[\mathrm{Ca}^{2+}]_{\mathrm{NSR}}}{\mathrm{d}t} = (I_{up} - I_{leak} - I_{tr}V_{\mathrm{JSR}}/V_{\mathrm{NSR}}) \tag{A.103}$$

Ca^{2+} Fluxes in Myoplasm

$$I_{t\mathrm{Ca}} = I_{\mathrm{Ca}} + I_{\mathrm{Ca}b} + I_{p(\mathrm{Ca})} - 2I_{\mathrm{NaCa}} + I_{\mathrm{Ca}(T)} \tag{A.104}$$
$$\begin{aligned}\Delta[\mathrm{Ca}^{2+}]_i = &-\Delta t\Big(((I_{t\mathrm{Ca}}A_{Cap})/(V_{myo}2F)) + ((I_{up} - I_{leak})V_{\mathrm{NSR}}/V_{myo}) - \\ &- (I_{rel}V_{\mathrm{JSR}}/V_{myo})\Big)\end{aligned} \tag{A.105}$$
$$[\mathrm{Ca}^{2+}]_{ion} = \mathrm{TRPN} + \mathrm{CMDN} + \Delta[\mathrm{Ca}^{2+}]_i + [\mathrm{Ca}^{2+}]_i \tag{A.106}$$
$$B = 0.05 + 0.07 - [\mathrm{Ca}^{2+}]_{ion} + 0.0005 + 0.00238 \tag{A.107}$$
$$\begin{aligned}C = &(0.00238 \cdot 0.0005) - ([\mathrm{Ca}^{2+}]_{ion}(0.0005 + 0.00238)) + \\ &+ (0.07 \cdot 0.00238) + (0.05 \cdot 0.0005)\end{aligned} \tag{A.108}$$
$$D = -0.0005 \cdot 0.00238[\mathrm{Ca}^{2+}]_{ion} \tag{A.109}$$

$$F_{ab} = \sqrt{(B^2 - 3C)} \tag{A.110}$$

$$[\mathrm{Ca}^{2+}]_i = 1.5F_{ab}\cos(\arccos((9BC - 2B^3 - 27D)/(2(B^2 - 3C)^{1.5}))/3) - (B/3) \tag{A.111}$$

Ca^{2+} Fluxes in JSR

$$\Delta[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} = \Delta t(I_{tr} - I_{rel}) \tag{A.112}$$

$$b_{\mathrm{JSR}} = 10 - \mathrm{CSQN} - \Delta[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} - [\mathrm{Ca}^{2+}]_{\mathrm{JSR}} + 0.8 \tag{A.113}$$

$$c_{\mathrm{JSR}} = 0.8(\mathrm{CSQN} + \Delta[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} + [\mathrm{Ca}^{2+}]_{\mathrm{JSR}}) \tag{A.114}$$

$$[\mathrm{Ca}^{2+}]_{\mathrm{JSR}} = (\sqrt{(b_{\mathrm{JSR}}{}^2 + 4c_{\mathrm{JSR}})} - b_{\mathrm{JSR}})/2 \tag{A.115}$$

Sodium Ion Fluxes

$$I_{t\mathrm{Na}} = i_{\mathrm{Na}} + I_{\mathrm{Na}b} + I_{\mathrm{CaNa}} + I_{ns\mathrm{Na}} + 3I_{\mathrm{NaK}} + 3I_{\mathrm{NaCa}} \tag{A.116}$$

$$\frac{\mathrm{d}[\mathrm{Na}^+]_i}{\mathrm{d}t} = -(I_{t\mathrm{Na}}A_{Cap})/(V_{myo}F) \tag{A.117}$$

Potassium Ion Fluxes

$$I_{t\mathrm{K}} = I_{\mathrm{K}r} + I_{\mathrm{K}s} + i_{\mathrm{K}} + I_{\mathrm{CaK}} + I_{ns\mathrm{K}} - 2I_{\mathrm{NaK}} + I_{to} + I_{st} \tag{A.118}$$

$$\frac{\mathrm{d}[\mathrm{K}^+]_i}{\mathrm{d}t} = -(I_{t\mathrm{K}}A_{Cap})/(V_{myo}F) \tag{A.119}$$

Ca^{2+} buffers in the myoplasm

$$\mathrm{TRPN} = 0.07[\mathrm{Ca}^{2+}]_i/([\mathrm{Ca}^{2+}]_i + 0.0005) \tag{A.120}$$

$$\mathrm{CMDN} = 0.05[\mathrm{Ca}^{2+}]_i/([\mathrm{Ca}^{2+}]_i + 0.00238) \tag{A.121}$$

identical to Luo-Rudy model[32].

Ca^{2+} buffer in JSR and SCQN

$$\mathrm{CSQN} = 10([\mathrm{Ca}^{2+}]_{\mathrm{JSR}}/([\mathrm{Ca}^{2+}]_{\mathrm{JSR}} + 0.8)) \tag{A.122}$$

(A.123)

identical to Luo-Rudy model[32].

CICR From Junctional SR (JSR)

$$I_{rel} = G_{rel}\mathrm{ryr}_{open}\mathrm{ryr}_{close}([\mathrm{Ca}^{2+}]_{\mathrm{JSR}} - [\mathrm{Ca}^{2+}]_i) \tag{A.124}$$

$$G_{rel} = 150/(1 + \exp(I_{t\mathrm{Ca}} + 5)/0.9) \tag{A.125}$$

$$\mathrm{ryr}_{open} = 1/(1 + \exp((-t_c + 4)/0.5)) \tag{A.126}$$

$$\mathrm{ryr}_{close} = 1 - (1/(1 + \exp((-t_c + 4)/0.5))) \tag{A.127}$$

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)[33] and Luo-Rudy model the delay of calcium release was around 2 ms after the the time of the maximum $\frac{\mathrm{d}V}{\mathrm{d}t}$.

Translocation of Ca^{2+} ions from NSR to JSR: I_{tr}

$$I_{tr} = ([\mathrm{Ca}^{2+}]_{\mathrm{NSR}} - [\mathrm{Ca}^{2+}]_{\mathrm{JSR}})/180 \tag{A.128}$$

identical to Luo-Rudy model[32].

Total time-independent current: I_v

$$I_v = I_{\mathrm{Na}b} + I_{\mathrm{NaK}} + I_{p(\mathrm{Ca})} + I_{\mathrm{K}p} + I_{\mathrm{Ca}b} + I_{\mathrm{K1}} \tag{A.129}$$

identical to Luo-Rudy model[32].

Total Current

$$\begin{aligned} I_t = & I_{\mathrm{K}r} + I_{\mathrm{K}s} + i_{\mathrm{K}} + I_{\mathrm{CaK}} + I_{ns\mathrm{K}} - 2I_{\mathrm{NaK}} + i_{\mathrm{Na}} + I_{\mathrm{Na}b} + I_{\mathrm{CaNa}} + I_{ns\mathrm{Na}} + 3I_{\mathrm{NaK}} + \\ & 3I_{\mathrm{NaCa}} + I_{\mathrm{Ca}} + I_{\mathrm{Ca}b} + I_{p(\mathrm{Ca})} - 2I_{\mathrm{NaCa}} + I_{\mathrm{Ca}(T)} \end{aligned} \tag{A.130}$$

Membrane Potential

$$\frac{\mathrm{d}V}{\mathrm{d}t} = -\ I_t \tag{A.131}$$

Appendix B

Implementation of Cellular Models

B.1 Standalone Code

B.1.1 Hodgkin-Huxley Squid Model

The goal of this appendix is to demonstrate the implementation of minimalist cellular models. The implementation is inspired by Hodgkin-Huxley squid giant axon description[4], which is the first electrophysiological model. The original description contains only four dynamical states, one for membrane voltage and two for gating variables of I_{Na} and one for gating variable of I_{K}.

The model is implemented as standalone code and as BeatBox modules. The BeatBox modules include implementation as `rhs` module and two `ionic` module. The first is identical with the original model description, second contains Markov chain models for the sake of demonstration of the time integration on Markov chains. For this purpose we convert both I_{Na} and I_{K} channels into equivalent Markov chain models. *add references*

Listings of Hodgkin-Huxley Standalone Code

The implementation of the giant squid axon model in standalone C is listed in the source code bellow:

Listing B.1: Code of standalone Hodgkin-Huxley model in file *squid_driver.c*.

```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* with contains function like macros for each of the dynamical variables.

Listing B.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

Compilation and linking of the program can be done by gcc:

Listing B.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 that can be redirected to a file by > operator.

Listing B.4: Execution of standalone model using standard output redirection.

```
  ./squid_driver > squid.dat
```

B.2 Implementation as BeatBox Modules

B.2.1 Enumeration of Variables

A feature of C preprocessor called function-like macros is routinely used for the enumeration of variables and other functions within BeatBox. As the understanding of the function-like macros is essential for the understanding of building of BeatBox modules, in this subsection, we briefly describe the functionality of the preprocessor macros.

The function-like macros used in BeatBox are normally in the format `_(<code1>,<code2>)`, where `<code1>`, and `<code2>` are pieces of code. The function like macros are processed by C preprocessor (cpp) according to the definition of a particular macro. An example of such definition is:

```
#define _(variable,initial) var_##variable,
```

where the string `var_` is concatenated with the first argument called `variable` (through cpp operator `##`). During the preprocessing stage of the compilation the cpp processes the code and each construct the specified form is substituted by the defined expression. For instance a code defining initial value of membrane voltage: `_(V,-80.0)`) would be substituted by `var_V`, and the value of `-80.0` would be ignored in this case).

After the function-like macro is not needed any more, it can be undefined by `#undef _` and redefined again when required. The definition in any of the cases is completely arbitrary, so it allows even two different definitions according to the needs of the code.

The advantage of using this approach becomes evident, when we realise another feature of C preprocessor cpp, which is import statements.

The import statements can include the content of any file into the preprocessed code. The syntax of import statement is:

```
#import "<filename>.h"
```

which includes the code in the file *<filename>.h* into the preprocessed code. If the file contains any function-like macros, which have been defined, their expansion is done in the same manner as described above. So, if the code defines two different function-like macros and after each of the two definition includes a code from an external file. The function-like macros within the included code will expand differently for each case of inclusion.

The specific definition of the function-like macro can be changed depending on a 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 needs to be done in one place only, which minimises redundant code and helps to avoid possible errors and simplify the maintenance of the module.

The following code listing illustrates the idea of expansion of function-like macros included from header files. This case shows for enumeration of all dynamical variables:

Listing B.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 a function-like macros e.g. `_(V,-80.0)`.

Notice, that due to the use of enumeration the `var_V` can be used interchangeably with a number corresponding to the position position of 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 number of "other" (`NO`), tabulated (`NT`) and non-tabulated gating variables (`NN`), as well as number of tabulated transition rates (`NTAB`).

The counting and initialisation of variables can be done through including of 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 some of routinely used file is shown in Table B.1.

Some `ionic` modules also import large parts of the code to minimise the size of the main file `<ionic>.c`. However, those files are included only once, and normally

Table B.1: Examples of imported 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

do not use function-like macros. Typically a module includes file `<ionic>_ftab.h` that specifies equations of tabulated voltage-dependent transition rates and other functions; 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 (e.g. temperature, Faraday constant).

B.2.2 Implementation of Markov Chains

We use the included files to preprocess of 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 this variable (e.g. `_(ltypeCzero, 0.948)`). Using this approach we can enumerate the variables of a single Markov chain, which is necessary to assign to the `dimensionality` of the `channel_str`.

Additional function-like macros serve to the initialisation of the `subchain_str`. Transition rates functions are constructed from macro `_VFUN(variable,expression)` where `variable` is the name of the transition rate and `expression` is corresponding mathematical expression.

The transition rates matrix is constructed from macro `_RATE(from,to,direct,reverse)` This macro specifies transition rates between two states (`from`, `to`) and the transition rates in between (`direct`, `reverse`) as described on page 129.

The macro `_RATE` helps to generate `TransRatesMat` functions that construct a function which constructs transition rates matrix of a subchain.

A macro called `CHANNEL_TR_MATRIX` can be used for this purpose. The following code listing shows an example which generates a function computing transition rates within an `ionic` module:

Listing B.6: Function for population of transition rates 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` is expanded using another function-like macro `TR_MAT` The `TR_MAT(chan,from,to,direct,reverse)` is defined in *`channel.h`* as:

Listing B.7: Function-like macro for population of transition rates 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;
```

B.2.3 Including Modules into BeatBox

The inclusion of cellular modules into BeatBox requires changes of BeatBox source files and 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 `ionic` module starts by including its name into the *ioniclist.h* in a form `D(<ionic>)`.

Listing B.8: Specification of new `ionic` modules in *src/ioniclist.h*.

```
  D(hh52)
  D(hh52m)
```

The building of a new `rhs` module starts by including its name into the *rhslist.h* in a form `D(<rhs>)`.

Listing B.9: Specification of new `rhs` modules in *src/rhslist.h*.

```
#if HH
  D(hh)
#endif
```

The `rhs` modules also require 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 main module file contains formal definition of the model. It is expected in the file *<name>.c*. The time stepping devices require the module module to initialise specific structures and functions.

All files used in for the implementation of those modules have to be listed within `common_sources` variable in *src/Makefile.am* in BeatBox root directory:

Listing B.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 build on a target computer architecture, that BeatBox executable is aimed for. This allows for 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).

B.2.4 Hodgkin-Huxley Squid Model

Code Listings as `rhs` Module

The listing bellow shows main file *hh.c* of the model `hh` which is a minimalist implementation of original Hodgkin-Huxley code in BeatBox in the `rhs` format.

Listing B.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 of the model as follows:

Listing B.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 command line argument to BeatBox (i.e. `Beatbox hh.bbs`).

Listing B.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;
```

Code Listings as `ionic` Module with Gate Models

The `rhs` module can be converted to a 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 ionic channel models, as “other” variables, calculated using forward Euler. This allows us to compare the solution of ionic module using rushlarsen device with its rhs counterpart solved by euler device. This comparison proved the identity of both results.

Consequently, we have implemented a C-function calculating voltage-dependent functions such as transition rates of the gating variables, and perform necessary changes to specify that the gating variables are now considered as tabulated gates. This implementation allows us to employ Rush-Larsen technique.

The listing bellow shows main file *hh52.c* of the model hh52 which is a minimalist implementation of original Hodgkin-Huxley code in BeatBox in the ionic format.

Listing B.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 imported from a file *hh52_ftab.h*.

Listing B.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:

Listing B.16: Other variables *hh52_other.h*.

```
/* format: _(name, initial) */
/* V  - membrane potential (mV) */
_(V, 7.0)
```

Listing B.17: Tabulated gate 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 parameters are defined in a similar format to variables.

Listing B.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);
```

Those parameters can be modified from bbs script as specified bellow.

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 B.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 `par` parameter of to `rushlarsen` device. For instance we could set up different membrane capacitance and sodium conductance by the following setup of `bbs` script:

```
rushlarsen v0=0 v1=neqn-1 ht=dt ionic=hh52 order=tog
  par={ht=dt C_m=2.0 G_Na=130};
```

Code Listings as `ionic` Module with Markov Chain Models

To illustrate the functionality of Matrix Rush-Larsen method on the shown minimal example of `ionic` module, we convert the gating variables from the `hh52` module into Markov chain.

Each combination of states for both I_{K} and I_{Na} channels correspond to one state of resulting Markov chain model as described in Subsection 1.3.3.

The definition of the gates is specified in imported files listed bellow *hh52m_ik.h*:

Listing B.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 this model in BeatBox format is specified in the file *hh52m_ina.h*:

Listing B.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 B.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>
#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);
```

This *hh52m* imports some files listed in previous section. Those are *hh52_other.h*, *hh52_tgates.h*, *hh52_other.h*, *hh52_par.h*.

The bbs script for hh52m module has only a few differences to *hh52.bbs*. As the gate variables were converted into corresponding Markov chain models the number of dynamical equations increased. So, the parameter, which specifies the number of dynamical variables `neqn` is increased to `14`. Also, the name of the `ionic` module to use in `rushlarsen` device call should read as `hh52m`. Finally, we need to sample open probabilities of both I_{Na} and I_{K} and save them.

Listing B.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;
```

B.2.5 TenTusscher-Panfilov (2006) Model

Calcium Dynamics

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 producing or binding free calcium) and the flow of calcium between two different compartments.

In certain conditions the membranes separating the compartments become permeable and allow the calcium to flux from one to another compartment of the cell, which can be defined in general terms as

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]}{\mathrm{d}t} = f([\mathrm{Ca}^{+2}], \ldots) \tag{B.1}$$

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

The calcium molecules then diffuse within the compartment to achieve homogeneous calcium concentration. Additionally, calcium is a subject of a reaction with molecules called buffers – proteins which bind the calcium and so it becomes inaccessible for 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[35]. The total concentration of calcium $[\mathrm{Ca}^{+2}]_t$ and buffer molecules B_t is given by:

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

$$B_t = B_f + [\mathrm{Ca}^{+2}]_b. \tag{B.3}$$

From equation (B.2) 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[35]

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]_b}{\mathrm{d}t} = k_{\mathrm{on}}[\mathrm{Ca}^{+2}]_f \left(B_t - [\mathrm{Ca}^{+2}]_b\right) - k_{\mathrm{off}}[\mathrm{Ca}^{+2}]_b. \tag{B.4}$$

If the diffusion to happen in a fast time scale, the whole compartment would have isotropic Ca^{+2} concentration very fast. Then 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

$$0 = k_{\mathrm{on}}[\mathrm{Ca}^{+2}]_f \left(B_t - [\mathrm{Ca}^{+2}]_b\right) - k_{\mathrm{off}}[\mathrm{Ca}^{+2}]_b, \tag{B.5}$$

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

$$[\mathrm{Ca}^{+2}]_b = \frac{B_t[\mathrm{Ca}^{+2}]_f}{[\mathrm{Ca}^{+2}]_f + k}, \tag{B.6}$$

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

Substituting the result (B.6) into the relation for the total Ca^{+2} concentration (B.2) yields

$$[\mathrm{Ca}^{+2}]_t = [\mathrm{Ca}^{+2}]_f + \frac{B_t[\mathrm{Ca}^{+2}]_f}{[\mathrm{Ca}^{+2}]_f + k}. \tag{B.7}$$

We multiply by the denominator of the second term on the right hand side and collect terms with $[\mathrm{Ca}^{+2}]_f$ together to get a quadratic equation

$$[\mathrm{Ca}^{+2}]_f{}^2 + \left(k + B_t - [\mathrm{Ca}^{+2}]_t\right) [\mathrm{Ca}^{+2}]_f - k[\mathrm{Ca}^{+2}]_t = 0. \tag{B.8}$$

We find both roots of the quadratic equation and considering the physical constrains of the positiveness of concentration, we restrict the solution to a unique feasible root

$$[\mathrm{Ca}^{+2}]_f = \frac{1}{2}\left(-(k + B_t - [\mathrm{Ca}^{+2}]_t) + \sqrt{(k + B_t - [\mathrm{Ca}^{+2}]_t)^2 + 4k[\mathrm{Ca}^{+2}]_t}\right), \tag{B.9}$$

where the total concentration $[\mathrm{Ca}^{+2}]_t$ is described by 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:

$$\frac{\mathrm{d}[\mathrm{Ca}^{+2}]_t}{\mathrm{d}t} = f(\ldots), \tag{B.10}$$

$$[\mathrm{Ca}^{+2}]_f = \frac{1}{2}\left(-(k + B_t - [\mathrm{Ca}^{+2}]_t) + \sqrt{(k + B_t - [\mathrm{Ca}^{+2}]_t)^2 + 4k[\mathrm{Ca}^{+2}]_t}\right), \tag{B.11}$$

$$[\mathrm{Ca}^{+2}]_b = \frac{B_t[\mathrm{Ca}^{+2}]_f}{[\mathrm{Ca}^{+2}]_f + k}. \tag{B.12}$$

The numerical algorithm used for integrating such system of differential equation can be implemented in a number of different ways. For the purposes of BeatBox, the time stepping routine has to allow the flexibility of using any numerical integration method. So no particular time stepping should be present within the model code for calcium dynamics. We have noticed that some models include calcium dynamics using forward Euler. This is also the case of the TTP model, as we describe in following subsubsection.

Calcium Computation in TenTusscher-Panfilov Model

The Ca^{2+} dynamics within the TTP model are calculated for three compartments: sarcoplasmic reticulum (SR); dyadic subspace (SS), which is compartment in the proximity of the cellular membrane and SR; and bulk intra-cellular calcium.

The algorithm computing the calcium dynamics in the SR in the TTPs' code given as:

$$
\begin{aligned}
[\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} &= \frac{B_{\mathrm{SR}}[\mathrm{Ca}^{+2}]_{\mathrm{SR}}}{([\mathrm{Ca}^{+2}]_{\mathrm{SR}} + k_{\mathrm{SR}})}, && \text{(B.13)}\\
\Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} &= \Delta t\left(I_{\mathrm{up}} - I_{\mathrm{rel}} - I_{\mathrm{leak}}\right), && \text{(B.14)}\\
b_{\mathrm{SR}} &= B_{\mathrm{SR}} - [\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} - \Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} - [\mathrm{Ca}^{+2}]_{\mathrm{SR}} + k_{\mathrm{SR}}, && \text{(B.15)}\\
c_{\mathrm{SR}} &= k_{\mathrm{SR}}\left([\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} + \Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} + [\mathrm{Ca}^{+2}]_{\mathrm{SR}}\right), && \text{(B.16)}\\
[\mathrm{Ca}^{+2}]_{\mathrm{SR}} &= \frac{\sqrt{{b_{\mathrm{SR}}}^2 + 4c_{\mathrm{SR}}} - b_{\mathrm{SR}}}{2}. && \text{(B.17)}
\end{aligned}
$$

where $[\mathrm{Ca}^{+2}]_{\mathrm{CSQN}}$ represents buffered calcium $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ 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 $\Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$.

This method computes the $\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}$ by forward Euler formula embedded into (B.15) and (B.16) as

$$
\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}} = [\mathrm{Ca}^{+2}]_{\mathrm{CSQN}} + [\mathrm{Ca}^{+2}]_{\mathrm{SR}} + \Delta[\mathrm{Ca}^{+2}]_{\mathrm{SR}} \qquad \text{(B.18)}
$$

so that the higher order method can not be applied unless we reformulate the system in terms of pure dynamical equations.

Differential Equations for Calcium

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 using total concentrations of calcium in each compartment. This can be achieved by using the equations (B.10)–(B.12) with a specific set of variables and functions for each compartment.

The parameters for the calcium dynamics in sarcoplasmic reticulum are:

$$
\begin{aligned}
f(\dots) = \frac{\mathrm{d}\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SR}}}{\mathrm{d}t} &= I_{\mathrm{up}}([\mathrm{Ca}^{2+}]_i) - \\
&\quad I_{\mathrm{rel}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}}, O([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}})) - \\
&\quad I_{\mathrm{leak}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{2+}]_i), && \text{(B.19)}\\
B_t = B_{\mathrm{SR}} &= 10.0, && \text{(B.20)}\\
k = k_{\mathrm{SR}} &= 0.3, && \text{(B.21)}
\end{aligned}
$$

in the sub-space:

$$
f(\dots) = \frac{\mathrm{d}\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{SS}}}{\mathrm{d}t} = -\frac{V_{\mathrm{c}}}{V_{\mathrm{SS}}} I_{\mathrm{xfer}}([\mathrm{Ca}^{+2}]_{\mathrm{ss}}, [\mathrm{Ca}^{2+}]_i) +
$$

$$\begin{aligned} & \frac{V_{\mathrm{SR}}}{V_{\mathrm{SS}}} I_{\mathrm{rel}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}}, O([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{+2}]_{\mathrm{ss}})) - \\ & \frac{C_m}{2V_{\mathrm{SS}}F} I_{\mathrm{Ca}(L)}(d(\mathrm{V_m}), f_1(\mathrm{V_m}), f_2(\mathrm{V_m}), f_3([\mathrm{Ca}^{+2}]_{\mathrm{ss}})) \end{aligned} \tag{B.22}$$

$$B_t = B_{\mathrm{SS}} = 0.4 \tag{B.23}$$

$$k = k_{\mathrm{SS}} = 0.00025, \tag{B.24}$$

in the bulk intracellular space:

$$\begin{aligned} f(\ldots) = \frac{\mathrm{d}\overline{[\mathrm{Ca}^{+2}]}_{\mathrm{i}}}{\mathrm{d}t} = & - \frac{C_m}{2V_{\mathrm{c}}F} \left(I_{b\mathrm{Ca}}(\mathrm{V_m}) + I_{p\mathrm{Ca}}([\mathrm{Ca}^{2+}]_i) - 2I_{\mathrm{NaCa}}(\mathrm{V_m}, [\mathrm{Na}^+]_i, [\mathrm{Ca}^{2+}]_i) \right) - \\ & \frac{V_{\mathrm{SR}}}{V_{\mathrm{c}}} \left(I_{\mathrm{up}}([\mathrm{Ca}^{2+}]_i) - I_{\mathrm{leak}}([\mathrm{Ca}^{+2}]_{\mathrm{SR}}, [\mathrm{Ca}^{2+}]_i) \right) + \\ & I_{\mathrm{xfer}}([\mathrm{Ca}^{+2}]_{\mathrm{ss}}, [\mathrm{Ca}^{2+}]_i) \end{aligned} \tag{B.25}$$

$$B_t = B_i = 0.2 \tag{B.26}$$

$$k = k_i = 0.001. \tag{B.27}$$

where F is Faraday constant; V_{c}, V_{SR}, V_{SS} are volumes of intracellular space, sarcoplasmic reticulum and sub-space 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 for $I_{\mathrm{Ca}(L)}$ current $d(\mathrm{V_m})$, $f_1(\mathrm{V_m})$, $f_2(\mathrm{V_m})$, $f_3([\mathrm{Ca}^{+2}]_{\mathrm{ss}})$.

Comparison of Algorithms for Computation of Calcium Dynamics

The Figure B.1 shows the original (TTPs') 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 deviation from the exact solution. Because both methods only calculate approximate solution we compare them to each other, to demonstrate that any of them 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:

$$\epsilon = \left| \frac{A - D}{A} \right|, \tag{B.28}$$

where A is the value from the TTPs' code and D is the corresponding dynamical variable using the reformulated algorithm.

The size of computer memory is limited so only a finite number of significant digits can be saved. The trailing digits after the significant digits are truncated

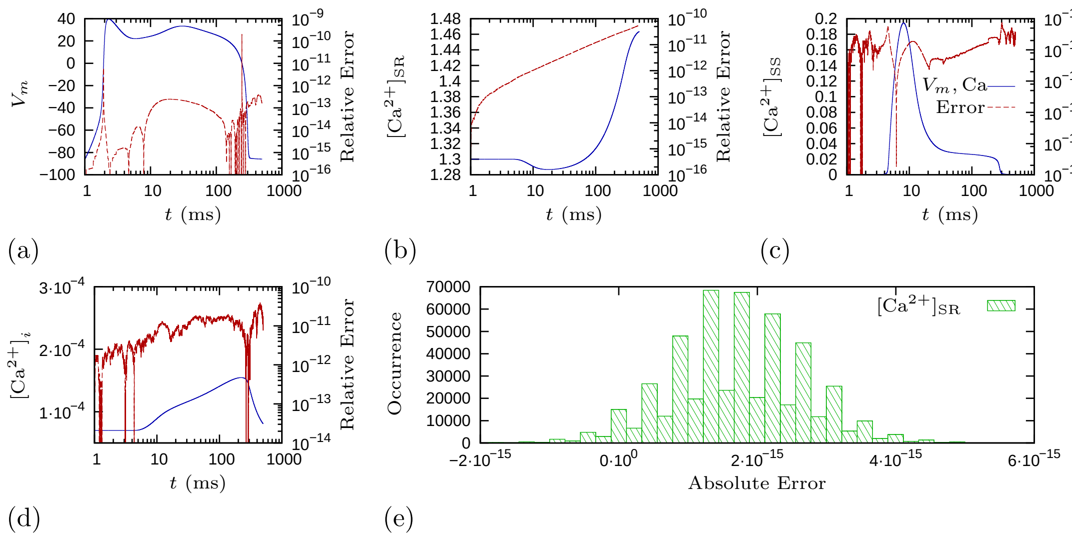


Figure B.1: Comparison of tenTusscher-Panfilov's and reformulated algorithm for computation of calcium dynamics: (a) Membrane potential $\mathrm{V_m}$, (b) calcium concentration in sarcoplasmic reticulum $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$, (c) sub-space $[\mathrm{Ca}^{+2}]_{\mathrm{ss}}$, (d) intracellular calcium $[\mathrm{Ca}^{2+}]_i$ (blue line shows simulated traces – left axis, red line shows the difference between results of TTPs' algorithm and reformulated code – right axis), (e) distribution of absolute differences in one time step between the TTPs' algorithm and reformulated code for $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$. Simulations were done using forward Euler method with time step of $\Delta t = 1\ \mu\mathrm{s}$.

which results in an approximation of original value and a numerical truncation error. When we use the approximated values for further calculations, the error can propagate onto other variables.

In the case of `double float` precision the number of significant digits is 15.

From the Figure B.1(a-d) we see, that the difference between TTPs' and updated algorithm is around 10^{-11}. Although this value of error seems rather high, we can show that such result is consistent.

The iterative solvers introduce small approximation error in each iteration. 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 TTPs' and updated algorithm side by side in one code. The calcium concentrations from updated algorithm D are used for the calculations of all the cellular compartments. Besides that we obtain the calcium concentrations using TTPs' algorithm based on calcium concentrations obtained by updated algorithm D in the previous time step, we denote those results by A^*.

The error in this experiment is than defined as the difference between the updated method and TTPs' method $(D - A^*)$. Figure B.1e shows the distribution of the values of error during the simulation. That the average error is skewed towards 10^{-15}.

As seen from the relative error $[\mathrm{Ca}^{+2}]_{\mathrm{SR}}$ on Figure B.1 panel (c) the error reach values about 10^{-11} after 100 ms with 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 error around 10^{-10}. We have to bear in mind, that this a rough estimation, as the two simulation are not exactly identical, however, it can give us an idea about the behaviour of the solution.

The error is probably caused by a difference between the computation of free calcium concentration between the two algorithms in one time step. In the TTPs' algorithm the increment of calcium due to inter-compartmental fluxes $\Delta[\mathrm{Ca}^{+2}]$ is added directly to the free calcium concentration $[\mathrm{Ca}^{+2}]_f$. In the approach used in the updated code, the $\Delta[\mathrm{Ca}^{+2}]$ increase the total calcium concentration. The free calcium concentration is then found from the total calcium concentration by an algebraic equation resulting from the steady-state approximation of the reaction of calcium buffering.

Implementation of TenTusscher-Panfilov in `ionic` Format

To use the BeatBox device for solving gate variables using Rush-Larsen, we have to convert the `rhs` module to `ionic` format. The specifications of the `ionic` module is described in Subsection 5.4.2. 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 section.

The dynamical variables are then divided into groups 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 other variables 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:

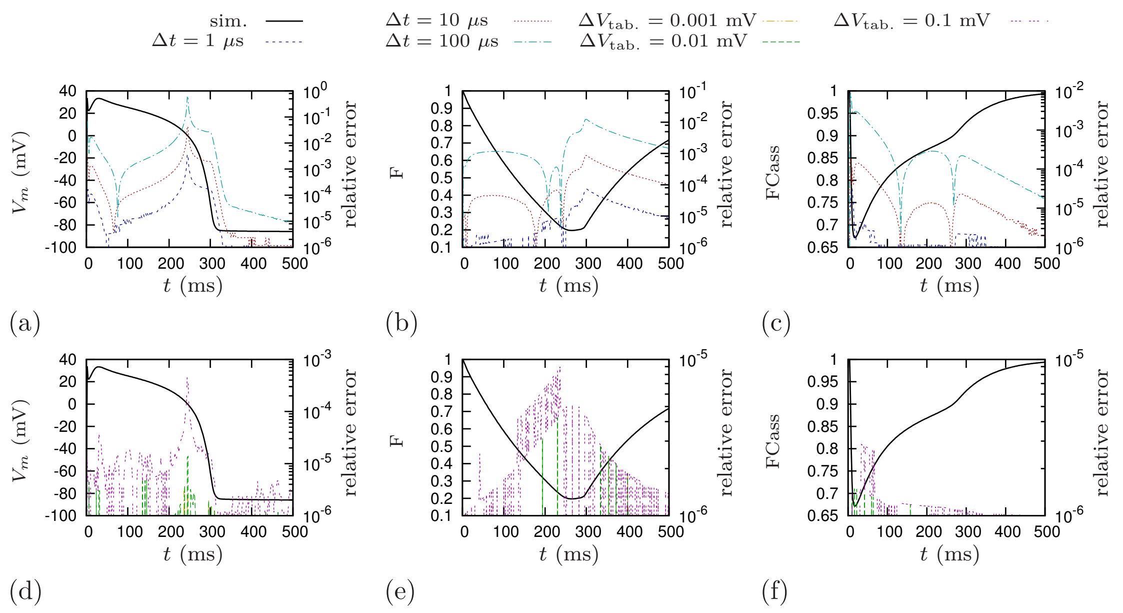


Figure B.2: Convergence of the solution in time-step (top row) and voltage-step in tabulation (bottom row) for tenTusscher-Panfilov (2006)[31] model. The panels (a), (d) show the membrane voltage $\mathrm{V_m}$ computed as “other” variable; the panels (b), (e) show F gate of the I_{Ca} current computed 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 $\Delta t = 0.1\ \mu\mathrm{s}$, 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).

- $\mathrm{V_m}$ — 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 sub-space;
- $[\mathrm{Na}^{+}]_i$ — sodium concentration;
- $[\mathrm{K}^{+}]_i$ — potassium concentration.

The dynamics of the calcium refer to the total calcium concentration rather than the free calcium concentration as in the authors' code.

Figure B.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 gating variables in rushlarsen device are computed using Rush-Larsen method which provides higher accuracy of the solution compared to the rhs modules.

In the euler device the increment found during the computation of the right-hand sides of the differential equations is added to the corresponding dynamical states at the end of the time step. The rushlarsen device solving ionic modules updates the state of tabulated and non-tabulated gates and “other” variables separately (as described in Table 5.2) after each of the parts has been computed.

This different approach leads to small discrepancies in the result. The figure shows results computed using the `order=tgo`.

Appendix C

rushlarsen Source Code

Listing C.1: *ionic.h*

```
/**
 * Experimental module, not for distribution.
 * 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 */
```

```c
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 (number of columns in the table) */
  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 (steady-state) 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
```

Listing C.2: *channel.h*

```c
/* 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;
```

```c
#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 (must be reflected in TransRatesMat) */
} 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;
```

Listing C.3: *rushlarsen.c*

```c
/**
 * Experimental module, not for distribution.
 * Rush-Larsen solver for cardiac excitability kinetic models.
 * Utilizes a special kinetics format, describing HH-type gates separately.
 */

/* This device aims to add Matrix Rush Larsen solver for Markov chain (MC). It is a modification of rushlarsen device. */

#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 steppedn by RL, if zero 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);

  real *tr, *tr_transp;                            /* transposed transition rates matrix of INa */
  real *evec_right, *evec_left, *evals;          /* to eigenvalues and eigenvectors in simple format */
  real *exp_evals_ht, *evec_right_eval;          /* auxilary arrays for MRL computations */

  /* 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 (this is 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 models 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);          /* the inverse of increment of V in the table */
  DEVICE_ARRAY(real,du/*[no]*/);        /* array of right-hand sides for the 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 markov chain values */
  real *dmarkov,*dmarkov_cur;                   /* subvector of markov chain 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 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);            /* the inverse of increment of V in the table */
  DEVICE_ARRAY(real,du/*[no]*/);          /* array of right-hand sides for the 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 tabulated 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 */
/* TODO: since now it will contains only mcfe and tabmrl, perahps should be numeric */
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␣layer␣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));        /* creates space for mrl although might not be needed */
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 */
                  /* TODO: if the scale is logarithmic, should it still be (it + 0.5)? */
                  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));
                    }
                  /* else */
                  /*   { */
                  /*      ABORT("Unknown code of which_exp_mc = %d.", S->which_exp_mc); */
                  /*   } */


                  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);
    }


  /* If the ionic model allocated initial state, possibly */
  /* finalize it by calculating stationary values of the gates */
#if 0
  if (S->u) {
    ACCEPTI(equilibrate_gates,0,0,1);
    V=S->u[S->I.V_index];
    values=S->adhoc;

    if (!(S->I.fddt(S->u,nv,values,ntab,S->I.p,S->I.var,S->du,no,S->nalp,S->nbet,nn))) {
      URGENT_MESSAGE("error␣calculating␣fddt(%s)␣at␣parse␣time:␣u=",ionic);
      for(iv=0;iv<nv;iv++) URGENT_MESSAGE("␣%lg",u[iv]);
      ABORT("\n");
    } /*  if !fddt... */
    for (in=0;in<nn;in++) {
      alp=S->nalp[in];
      bet=S->nbet[in];
      S->u[no+in]=alp/(alp+bet);
    } /* for in */
    if (!(S->I.ftab)(V,values,ntab))
      EXPECTED_ERROR("error␣calculating␣ftab(%s)␣for␣V=%g\n",S->ionic,V);
    for (it=0;it<nt;it++) {
      calcab(values+it,values+nt+it,ht);
      alp=values[it];
      bet=values[nt+it];
      S->u[no+nn+it]=alp/(alp+bet);
    } /* for it */
  } else {
    if (find_key("equilibrate_gates=",w)) {
      MESSAGE("\n/*␣Warning:␣equilibrate_gate␣parameter␣specified␣for␣a␣model␣which␣"
              "␣does␣not␣define␣a␣standard␣state.␣"
              "The␣parameter␣will␣be␣ignored.␣*/\n");
    } /* if findkey */
  } /* if S->u else */
#endif  /* 0 -- comment */

  /* 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␣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␣*/");

#if 0
    for (step=0;step<S->rest;step++) {
      V=u[S->I.V_index];
      iV=floor((V-Vmin)*S->one_o_dV);
      ASSERT(iV>=0);
      ASSERT(iV<nV);
      values=S->tab+iV*ntab;
      if (!(S->I.fddt(u,nv,values,ntab,S->I.p,S->I.var,S->du,no,S->nalp,S->nbet,nn))) {
        URGENT_MESSAGE("error␣calculating␣fddt(%s)␣at␣parse␣time,␣iteration␣%d:␣u=",ionic,step);
        for(iv=0;iv<nv;iv++) URGENT_MESSAGE("␣%lg",u[iv]);
        ABORT("\n");
      } /*  if !fddt... */
      for (iv=0;iv<nv;iv++) {
        if (!isfinite(S->du[iv])) {
          URGENT_MESSAGE("\nfddt␣returned␣NAN␣(not-a-number):\n");
          for (jv=0;jv<nv;jv++)
            URGENT_MESSAGE("%c%g",jv?',':'(',S->du[jv]);
          URGENT_MESSAGE(")\nin␣response␣to␣input␣vector:\n");  \
          for (jv=0;jv<nv;jv++)
            URGENT_MESSAGE("%c%g",jv?',':'(',S->u[jv]);
          ABORT(")\n");
        } /* if !isfinite */
      } /* for iv */
      for (in=0;in<nn;in++) {
        alp=S->nalp[in];
        bet=S->nbet[in];
        u[no+in]=alp/(alp+bet); /* would it be better to do RL step? */
      } /* for in */
      for (it=0;it<nt;it++) {
        alp=values[it];
        bet=values[nt+it];
        u[no+nn+it]=alp/(alp+bet); /* would it be better to do RL step? */
      } /* for it */
      for (io=0;io<no;io++) {
        u[io]+=S->ht*S->du[io];
        if (!isfinite(u[io])) {
          URGENT_MESSAGE("\nNAN␣(not-a-number)␣at␣parse␣time,␣iteration␣%d␣component␣%d\n",step,io);
          URGENT_MESSAGE("This␣happened␣after␣an␣increment␣by␣%lg*",S->ht);
          for(jo=0;jo<nv;jo++) URGENT_MESSAGE("%c%lg",jo?',':'(',S->du[jo]);
          URGENT_MESSAGE(")\n");
          return 0;
        }
      } /* for im */
      for (im=0;im<nmv;im++) {
        u[no+nn+nt+im]+=S->ht*S->du[no+nn+nt+im];
        if (!isfinite(u[no+nn+nt+im])) {
          URGENT_MESSAGE("\nNAN␣(not-a-number)␣at␣parse␣time,␣iteration␣%d␣component␣%d:␣v=%d",step,im);
          URGENT_MESSAGE("This␣happened␣after␣an␣increment␣by␣%lg*",S->ht);
          URGENT_MESSAGE(")\n");
          return 0;
        }
      } /* for im */
    } /* for step */
#endif
#if 1
```

```
    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;
#endif

    /* 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

Eigenvalue Computation

For the MRL method, we need to solve eigenvalue problem. Although, cost of the computation of eigenvalue problem is much higher than when using simple forward Euler method, the benefit from the time step increase might overweight the costs. If the transition rates matrix depend on a single control variable, the MRL operator matrices can be computed only once for a grid of control variable and tabulated.

For our purposes we need to solve eigenvalues of relatively small systems (e.g. transition rates matrix of I_{Na} is of the dimension 9×9). To solve eigenvalue problem it is recommend to use canned subroutines[36].

There are a number of libraries containing subroutines to find eigenvalues and eigenvectors[37, 3]. Due to the copyright conditions of BeatBox, we need to choose a library compatible with a GNU GPL license.

There is a large number of software libraries which can be used to find eigenvalues and eigenvectors (see Table D.1 for some examples). All this packages require BLAS and many also require LAPACK.

LAPACK is a popular package, which can solve the eigenvalue problem. LAPACK is written in Fortran, but has C language interface LAPACKE.

Table D.1: Numerical libraries for solving eigenvalue problem (based on [3])

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

D.1 Linear Algebra Package LAPACK

D.1.1 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 (e.g. C) the implementation of LAPACK functions is not straightforward due different design assumption between the languages (e.g. structures of data, ways scalar values are passed to the functions). The C functions have to use an interface in order to benefit from LAPACK functionality. A popular LAPACK C interface called lapacke is included in LAPACK package. So, the C code can use LAPACK function through lapacke interface, and be linked against lapacke precompiled lapacke libraries at during the build.

The LAPACK library depends on subroutines of Basic Algebra Subprograms (BLAS). Rather 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 supposed to be optimised for the particular system and as the efficiency of LAPACK depends very much on the efficiency of BLAS implementation.

We have implemented the code for eigenvalue computation of I_{Na} Markov chain model shown in the next subsection.

D.1.2 Standalone LAPACK Code for Eigenvalue Computation

This section shows the code for computation of eigenvalues and eigenvectors of I_{Na} Markov chain models. 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
}
```

D.2 GNU Scientific Library (GSL)

D.2.1 Overview of GSL

GNU Scientific Library is a collection of routines for numerical computing[38]. The GSL is released under GNU GPL license. The GNU GPL contains a copyleft clause, which restrict the library from the use in non-free (proprietary) software, i.e. any program using GSL library must be a free software. BeatBox is a free software and so the use of GSL is authorised.

D.2.2 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 "ttp2006mrl_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 "ttp2006mrl_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");
    }

      /************/
  /* 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;
}
```

D.2.3 Including GSL to BeatBox

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 for portability of the code on 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 Makefiles. 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 reference path to the GSL libraries as follows:

```
beatbox_CPPFLAGS += -I$(GSL)
beatbox_SEQ_CPPFLAGS += -I$(GSL)/
beatbox_LDADD += -lgsl -lgslcblas
```

The complete build of GSL library takes several minutes, although many of the function so provided, 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 secondary function has to be included the binary, although the specific function is actually never used. That soon becomes complicated system, where we would need to include the complete library to satisfy all the dependencies.

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.

Appendix E

Second Order MRL

E.0.4 Extensions of Rush-Larsen Method to Higher Order

Perego-Veneziani method

double check if that is correct especialy for non-gating variables, check the variables names

Perego, Veneziani (2009)[39] suggest a higher order extension of Rush-Larsen scheme. Here we rewrite the method for the gate model in a situation, where the membrane voltage (or other variables the gates depend) are computed using ano

rewrite the general ODE (2.97) element wise as

$$\frac{\mathrm{d}x^i}{\mathrm{d}t} = f^i(t, \vec{x}(t)) = a^x(t, \vec{x}(t))x^i + b^x(t, \vec{x}(t)). \tag{E.1}$$

For gate model (2.104) gets form

$$\frac{\mathrm{d}w}{\mathrm{d}t} = a^w(t, \vec{x}(t))w + b^w(t, \vec{x}(t)) \tag{E.2}$$

where $a^w = -(\alpha + \beta)$ and $b^w = \alpha$. The Rush-Larsen scheme then becomes

$$w_{n+1} = \exp({a^w}_n \Delta t)\left(w_n + \frac{{b^w}_n}{{a^w}_n}\right) - \frac{{b^w}_n}{{a^w}_n} \tag{E.3}$$

where Δt is the time step, $w_n \approx w(t_n)$, coefficients ${a^w}_n = a^w(t_n, \vec{x}_n)$ and ${b^w}_n = b^w(t_n, \vec{x}_n)$. To increase the accuracy we estimates the coefficients ${a^w}_{n+1/2}$ and ${b^w}_{n+1/2}$ which approximate the values in the middle of the interval $[t_n, t_{n+1}]$. Those values are inferred from values at previous t_{n-1}, current t_n, and predicted value at the next step which is found in two iteration. First, the predictor step computes the

Table E.1: Coefficients of the numerical schemes

	c_{-1}	c_0	c_1
FE	0	1	0
$M(\theta)$	$\frac{\theta}{2}+\frac{1}{4}$	$1-\theta$	$\frac{\theta}{2}-\frac{1}{4}$
AB2	0	$\frac{3}{2}$	$-\frac{1}{2}$
CN	$\frac{1}{2}$	$\frac{1}{2}$	0
AM3	$\frac{5}{12}$	$\frac{8}{2}$	$-\frac{1}{12}$

value of variables, which the coefficients depend on as

$$\bar{x}_{n+1}{}^{i} = \exp(\bar{a^{x}}_{n+1/2}\Delta t)\left(x_{n}{}^{i} + \frac{\bar{b^{x}}_{n+1/2}}{\bar{a^{x}}_{n+1/2}}\right) - \frac{\bar{b^{x}}_{n+1/2}}{\bar{a^{x}}_{n+1/2}} \tag{E.4}$$

where the coefficients only consider current and past time steps as

$$\bar{a^{x}}_{n+1/2} = \bar{c}_0 a^{x}{}_{n} + \bar{c}_1 a^{x}{}_{n-1}, \tag{E.5a}$$
$$\bar{a^{x}}_{1/2} = (\bar{c}_0 + \bar{c}_1) a^{x}{}_{0}, \tag{E.5b}$$
$$\bar{b^{x}}_{n+1/2} = \bar{c}_0 b^{x}{}_{n} + \bar{c}_1 b^{x}{}_{n-1}, \tag{E.5c}$$
$$\bar{b^{x}}_{1/2} = (\bar{c}_0 + c_1) b^{x}{}_{0}. \tag{E.5d}$$

The predicted values for the $\bar{a^{w}}_{n+1/2}$ and $\bar{b^{w}}_{n+1/2}$ are corrected as

$$\hat{a^{w}}_{n+1/2} = c_{-1} a^{w}(t, \vec{\bar{x}}_{n+1}) + c_0 \bar{a^{w}}_{n} + c_1 \bar{a^{w}}_{n-1}, \tag{E.6a}$$
$$\hat{a^{w}}_{1/2} = c_{-1} a^{w}(t, \vec{\bar{x}}_{n+1}) + (c_0 + c_1) \bar{a^{w}}_{0}, \tag{E.6b}$$
$$\hat{b^{w}}_{n+1/2} = c_{-1} b^{w}(t, \vec{\bar{x}}_{n+1}) + c_0 \bar{b^{w}}_{n} + c_1 \bar{b^{w}}_{n-1}, \tag{E.6c}$$
$$\hat{b^{w}}_{1/2} = c_{-1} b^{w}(t, \vec{\bar{x}}_{1}) + (c_0 + c_1) \bar{b^{w}}_{0}. \tag{E.6d}$$

Finally, the value at the next time step is obtained as

$$w_{n+1} = \exp(\hat{a^{w}}_{n+1/2}\Delta t)\left(w_{n} + \frac{\hat{b^{w}}_{n+1/2}}{\hat{a^{w}}_{n+1/2}}\right) - \frac{\hat{b^{w}}_{n+1/2}}{\hat{a^{w}}_{n+1/2}}. \tag{E.7}$$

The constants c_{-1}, c_0 and c_1 are chosen from the values shown in the Table E.1, where the FE stands for forward Euler, AB2 stands for Adams-Bashforth, CN for Crank-Nicolson, AM3 for Adams-Moulton and $M(\theta)$ reflects the basic requirements for all other constants set.

Sundnes et al. method

Sundnes et al. (2009)[40] consider general system as defined in (2.97). Linearising the system they obtain a system for each of i differential equations as

$$\frac{\mathrm{d}x^i}{\mathrm{d}t} = f^i(\eta) + (x_i - \eta^i)\frac{\partial}{\partial x^i} f^i(\eta) = \mathcal{A}^i + (x^i - \eta^i)\mathcal{B}^i \tag{E.8}$$

where $\mathcal{A}^i = f^i(\eta)$ and $\mathcal{B}^i = \partial f^i(\eta)/\partial x^i$ and $\mathcal{B}^i$ is found using numerical differentiation which reads as

$$\mathcal{B}^i = \frac{f^i(\eta^1, \dots, \eta^{i-1}, \eta^i + \delta, \eta^{i+1}, \dots, \eta_k) - f^i(\eta)}{\delta}. \tag{E.9}$$

Solving (E.8) gives

$$x^i = \eta^i + \frac{\mathcal{A}^i}{\mathcal{B}^i}\left(\exp(\mathcal{B}^i \Delta t) - 1\right). \tag{E.10}$$

The numerical scheme is implemented in two steps. First, the solution at $\eta = x_n$ is obtained up to $t_{n+1/2}$, i.e. for $\Delta t/2$

$$x^i_{n+1/2} = x^i_n + \frac{\mathcal{A}^i}{\mathcal{B}^i}\left(\exp(\mathcal{B}^i(t_{n+1/2} - t_n)) - 1\right) \tag{E.11}$$

In the second step, the solution is found for $\eta = \bar{x}_{(i)}$ as:

$$x^i = x^i_n + \frac{\bar{a^w}}{\bar{b^w}}\left(\exp(\bar{b^w}\Delta t) - 1\right) \tag{E.12}$$

where $\bar{a^w} = f^i(\bar{x}_{(i)})$, $\bar{b^w} = \partial f^i(\bar{x}_{(i)})/\partial x^i$ and

$$\bar{x}_{(i)} = (x^1_{n+1/2}, \dots, x^{i-1}_{n+1/2}, x^i_n, x^{i+1}_{n+1/2}, \dots, x^k_{n+1/2}) \tag{E.13}$$

This means, that all the $\bar{a^w}$ and $\bar{b^w}$ have to be computed for each $\bar{x}_{(i)}$.

E.1 Application of Second Order Methods to Markov chain

The schemes in previous sections were developed for a general system. Applying those to a Markov chain allow us to proceed with assumptions, which simplify the process. This is because the transition rates matrix of Markov chains are linear and in our case, dependent only on the membrane voltage $\mathrm{V_m}$:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = A(\mathrm{V_m})u \tag{E.14}$$

E.1.1 Perego-Veneziani method

We start the from the Matrix Rush-Larsen scheme, which is then given by

$$u^{n+1} = \exp(A^n \Delta t) u^n \tag{E.15}$$

where the transition rates matrix $A^n = A(\mathrm{V_m}(t_n))$ is constant for the duration of one time step.

Perego-Veneziani suggested a method for the second order solution, where the coefficients in the Rush-Larsen formulas are "frozen" at half-step, rather then at its beginning. This gives the following formula for the MRL scheme:

$$u^{n+1} = \exp(A^{n+1/2} \Delta t) u^n \tag{E.16}$$

we have to find the $A^{n+1/2} = A(t_{n+1/2})$, which is the transition rates matrix at the half step. This is found using the following formula:

$$A^{n+1/2} = c_{-1} A^{n+1} + c_0 A^n + c_1 A^{n-1} \tag{E.17}$$

where the constants c are chosen from the table E.1 and the $A^{n+1} = A(\hat{\mathrm{V_m}}^{n+1})$.

The membrane voltage at the next step $\hat{\mathrm{V_m}}^{n+1}$ is found by a predictor step

$$\hat{\mathrm{V_m}}^{n+1} = \mathrm{V_m}^n + \Delta t(\bar{c}_0 I^n + \bar{c}_1 I^{n-1}) \tag{E.18}$$

where the $\bar{c}$ correspond to the coefficients c from the table E.1, whose $c_{-1} = 0$. The $i^n = -\sum_j I_j$

Alternative estimate for $\hat{\mathrm{V_m}}^{n+1}$

The alternative approach could estimate the $\hat{\mathrm{V_m}}^{n+1}$ using by using the value of $\dot{\mathrm{V}}_\mathrm{m}$ as:

$$\hat{\mathrm{V_m}}^{n+1} = \mathrm{V_m}^n + \Delta t \dot{\mathrm{V}}_\mathrm{m}^n \tag{E.19}$$

where

$$\dot{\mathrm{V}}_\mathrm{m}^n = (\mathrm{V_m}^n - \mathrm{V_m}^{n-1})/\Delta t, \qquad \dot{\mathrm{V}}_\mathrm{m}^0 = 0 \tag{E.20}$$

E.1.2 Application of Perego-Veneziani to MRL

The AP was initialised by setting the voltage up to the -35 mV. The results (exept FE 1) were obtained using tabulation.

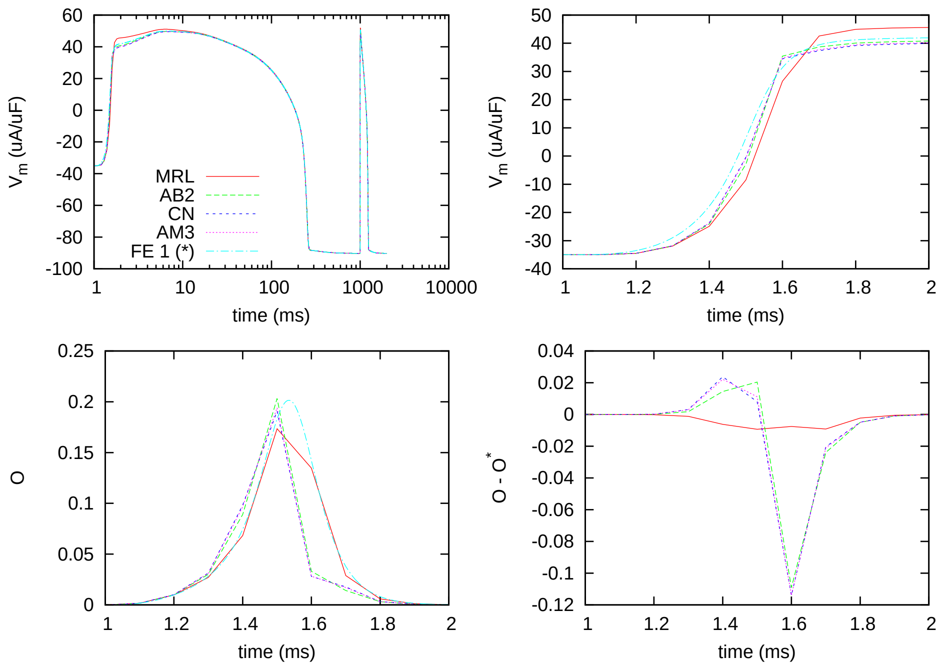


Figure E.1: Second order accuracy based on voltage at a half step at time step $\Delta t = 100\ \mu$s. Lines represents the following solvers: first order Matrix Rush-Larsen (MRL) – red lines; second order Adams-Bashforth (AB2) – green long-dashed lines, Crank-Nicolson (CN) – blue dashed lines, Adams-Moulton (AM3) – magenta dotted lines; and forward Euler with $\Delta t = 1\ \mu$s.The top row shows membrane voltage $\mathrm{V_m}$ for 2APs (left) and zoom for the first ms (right); bottom row shows open state occupancy in the first ms (left) and its difference from FE 1 (marked with *) (right bottom panel).

Based on voltage at half step

The first implementation uses the value of the voltage at half step:

$$\hat{\mathrm{V_m}}^{n+1} = \mathrm{V_m}^n + \Delta t(\bar{c}_0 i^n + \bar{c}_1 i^{n-1}) \tag{E.21}$$

$$\mathrm{V_m}^{n+1/2} = c_{-1}\hat{\mathrm{V_m}}^{n+1} + c_0 \mathrm{V_m}^n + c_1 \mathrm{V_m}^{n-1} \tag{E.22}$$

then $A^{n+1/2} = A(\mathrm{V_m}^{n+1/2})$. Where $\bar{c}_0 = 1.5$ and $\bar{c}_1 = -0.5$ and c_{-1}, c_0, c_1 correspond to a specific method in the table E.1.

The results for simulations with time step $\Delta t = 100\ \mu$s are shown on Figure E.1. At lower time steps, the results are qualitatively identical. The results are compared to the forward Euler at $\Delta t = 1\ \mu$s, as it best approximates the exact solution; and to Matrix Rush-Larsen as it is the method we aim to improve by second order accuracy. The solution using MRL leads to faster upstroke of AP and higher maximum voltage than the FE 1.

Based on transition rates matrix at half step

This implementation calculates the transition rates matrix at half step as follows:

$$\hat{\mathrm{V}_\mathrm{m}}^{n+1} = {\mathrm{V}_\mathrm{m}}^{n} + \Delta t(\bar{c}_0 i^n + \bar{c}_1 i^{n-1}) \tag{E.23}$$

$$A^{n+1/2} = c_{-1} A(\hat{\mathrm{V}_\mathrm{m}}^{n+1}) + c_0 A({\mathrm{V}_\mathrm{m}}^{n}) + c_1 A({\mathrm{V}_\mathrm{m}}^{n-1}) \tag{E.24}$$

where $\bar{c}_0 = 1.5$ and $\bar{c}_1 = -0.5$; and c_{-1}, c_0, c_1 correspond to a specific method in the table E.1.

Figure **??** shows results of the simulations. The APs are initiated by setting the membrane voltage to -35 mV. The simulations were performed with time step $\Delta t = 100\ \mu$s. At lower time steps the solutions of CN and visually overlap with the solution for AB2 in their vicinity.

The solution for AB2 (green lines) shows faster upstroke and decay. The open state occupancy reach nonphysical negative values in this case. The CN and AM3 leads to faster upstroke but after 0.5 ms approximate FE 1.

E.1.3 Sundnes et al. scheme

The system writes as:

$$\frac{\mathrm{d}w}{\mathrm{d}t} = f(w) \tag{E.25}$$

Which splits to:

$$\frac{\mathrm{dV_m}}{\mathrm{d}t} = -\frac{1}{C}\sum_j I_j(\mathrm{V_m}, u, c_j, m_j^*) = i(\mathrm{V_m}, u, c_j, m_j^*) \tag{E.26}$$

$$\frac{\mathrm{d}m_i}{\mathrm{d}t} = \alpha_i(\mathrm{V_m})(1-m_i) - \beta_i(\mathrm{V_m})m_i, \qquad i = 1, \dots, k \tag{E.27}$$

$$\frac{\mathrm{d}c_j}{\mathrm{d}t} = g(c_j, m_j^*, \mathrm{V_m}), \qquad j = 1, \dots, l \tag{E.28}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} = A(\mathrm{V_m})u \tag{E.29}$$

We linearise the system using (E.8) and solve analytically to get the solution

$$w_i = \eta_i + \frac{\alpha_i}{\beta_i}\left(\exp(\beta_i(t - t_n)) - 1\right) \tag{E.30}$$

where $\eta_i = w_i(t_n)$ in the predictor step, which leads to the solution $w_i(t_{n+1/2})$. This result is then corrected by substituting

$$\eta_i = \bar{w}_{(i)} = (w_1^{n+1/2}, \dots, w_{i-1}^{n+1/2}, w_i^n, w_{i+1}^{n+1/2}, \dots, w_k^{n+1/2})$$

As the Markov chain model is already in a linear form, we can proceed directly to the analytic solution which yield the scheme:

$$u^{n+1} = \exp(A(\mathrm{V_m}^{n+1/2})\Delta t)u^n \tag{E.31}$$

where $\mathrm{V_m}^{n+1/2}$ is obtained from the linearisation of equation (E.26) and substituting $\eta = \mathrm{V_m}^n$ to (E.30):

$$\mathrm{V_m}^{n+1/2} = \mathrm{V_m}^n + \frac{\alpha_i}{\beta_i}\Big(\exp(\beta_i(t_{n+1/2} - t_n)) - 1\Big) \tag{E.32}$$

where $\alpha_i = i(\mathrm{V_m}^n, u^n, c_j{}^n, m_j^{*n})$ and $\beta_i = \frac{\partial i(\mathrm{V_m}^n, u^n, c_j, m_j^{*n})}{\partial \mathrm{V_m}}$ is obtained using numerical integration

$$\beta_i = \frac{i(\mathrm{V_m}^n + \delta, u^n, c_j{}^n, m_j^{*n}) - i(\mathrm{V_m}^n, u^n, c_j{}^n, m_j^{*n})}{\delta} \tag{E.33}$$

and where δ is a small number.

Bibliography

[1] 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.

[2] 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.

[3] Jack Dongarra. Freely available software for linear algebra (may 2013). web page, May 2013. Table on Sparse Eigenvalue Solvers; retrieved 26 January 2015.

[4] 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.

[5] Vladimir E. Bondarenko. A compartmentalized mathematical model of the β1-adrenergic signaling system in mouse ventricular myocytes. *PLoS One*, 9(2):e89113, 2014.

[6] Bertil Hille. *Ionic Channels of Excitable Membranes*. Sinauer Associates Inc, second edition, 1992.

[7] F. Bezanilla and C. M. Armstrong. Inactivation of the sodium channel. I. Sodium current experiments. *J Gen Physiol*, 70(5):549–566, Nov 1977.

[8] C. M. Armstrong and F. Bezanilla. Inactivation of the sodium channel. II. gating current experiments. *J Gen Physiol*, 70(5):567–590, Nov 1977.

[9] G. W. Beeler and H. Reuter. Reconstruction of the action potential of ventricular myocardial fibres. *J Physiol*, 268(1):177–210, Jun 1977.

[10] 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.

[11] 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.

[12] 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.

[13] Gonzalo Ferreira, Eduardo Ríos, and Nicolás Reyes. Two components of voltage-dependent inactivation in $Ca_v1.2$ channels revealed by its gating currents. *Biophys J*, 84(6):3662–3678, Jun 2003.

[14] A. Fabiato. Time and calcium dependence of activation and inactivation of calcium-induced release of calcium from the sarcoplasmic reticulum of a skinned canine cardiac purkinje cell. *J Gen Physiol*, 85(2):247–289, Feb 1985.

[15] A. Zahradníková and I. Zahradník. A minimal gating model for the cardiac calcium release channel. *Biophys J*, 71(6):2996–3012, Dec 1996.

[16] J. Keizer and L. Levine. Ryanodine receptor adaptation and Ca^{2+}-induced Ca^{2+} release-dependent Ca^{2+} oscillations. *Biophys J*, 71(6):3477–3487, Dec 1996.

[17] CA Villalba-Galea, BA Suarez-Isla, M Fill, and AL Escobar. Kinetic model for ryanodine receptor adaptation. *BIOPHYSICAL JOURNAL*, 74(2, Part 2):A58, FEB 1998.

[18] M. D. Stern, L. S. Song, H. Cheng, J. S. Sham, H. T. Yang, K. R. Boheler, and E. Ríos. Local control models of cardiac excitation-contraction coupling. A possible role for allosteric interactions between ryanodine receptors. *J Gen Physiol*, 113(3):469–489, Mar 1999.

[19] T. R. Shannon, K. S. Ginsburg, and D. M. Bers. Potentiation of fractional sarcoplasmic reticulum calcium release by total and free intra-sarcoplasmic reticulum calcium concentration. *Biophys J*, 78(1):334–343, Jan 2000.

[20] A. N. Tikhonov. Systems of differential equations containing small parameters in the derivatives. *Mat. Sb. (N.S.)*, 31(73)(3):575–586, 1952.

[21] Neil Fenichel. Geometric singular perturbation theory for ordinary differential equations. *Journal of Differential Equations*, 31(1):53 – 98, 1979.

[22] V.N. Biktashev. Envelope equations for modulated non-conservative waves. *IUTAM Symposium Asymptotics, Singularities and Homogenisation in Problems of Mechanics*, 5(1):11, 2003.

[23] S. Rush and H. Larsen. A practical algorithm for solving dynamic membrane equations. *IEEE Trans Biomed Eng*, 25(4):389–392, Jul 1978.

[24] 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.

[25] G. M. Faber and Y. Rudy. Action potential and contractility changes in [na+](i) overloaded cardiac myocytes: A simulation study. *Biophysical Journal*, 78(5):2392–2404, May 2000.

[26] Gregory M Faber, Jonathan Silva, Leonid Livshitz, and Yoram Rudy. *Source code of Faber et al. (2007) cellular model*.

[27] BeatBox developers. Beatbox home page http://empslocal.ex.ac.uk/people/staff/vnb262/software/beatbox/. online, accessed 2015.

[28] BeatBox developers. Beatbox documentation, 2014.

[29] R. McFarlane. *High-Performance Computing for Computational Biology of the Heart*. PhD thesis, University of Liverpool, 2010.

[30] BeatBox developers. Source code of beatbox package. online, accessed 2015.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] William T. Vetterling Brian P. Flannery William H. Press, Saul A. Teukolsky. *Numerical Recipes; The Art of Scientific Computing*. Cambridge University Press, 2007.

[37] C. Wilkinson, H.H.; Reish. *Handbook for Automatic Computation*. Springer, 1971.

[38] M. Galassi et al. *GNU Scientific Library Reference Manual*. 2009.

[39] 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.

[40] 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.