University of Alberta

INVESTIGATION OF ALGORITHMS FOR SOLVING THE ELECTRO-CARDIAC ACTIVITY

by

Soheila Aalami

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Applied Mathematics

Department of Mathematical and Statistical Sciences

Fall 2012
Edmonton, Alberta

TO MY MOTHER AND MY FATHER

Abstract

Mathematical models are used to simulate the behavior of the electrical activity of a single cell or multiple cells of the heart. Single cell models contain a system of ODEs (ordinary differential equations) while multi-cell models consist of a coupled system of ODEs and PDEs (partial differential equations). We present different algorithms to explore the efficiency of different solvers for simulating cardiac models. We use operator splitting methods to split the coupled system of equations into the ODE and PDE parts. Then, we examine different solvers for simulating each part separately. Experiments shows that solving the ODE part contributes significantly to the total work required for the simulation of the multi-cell models. Therefore, utilizing efficient solution methods for this part of the problem is a requirement. The goal of this research is investigating efficient algorithms for solving these mathematical models.

Acknowledgements

Without the support of many people, this research would not have been possible. I want to express my gratitude to my supervisor, Dr. Belhamadia who was helpful and offered assistance and guidance. Deepest gratitude are also due to the members of the supervisory committee without whose assistance I was not able to finish my thesis successfully. Thanks to my co-supervisor, Dr. Minev, for sharing his valuable time and for giving me helpful information. Thanks also to Dr. Hillen for his support and advice. Thanks to my husband, Mohsen, whose support has always been my source of strength and inspiration. Thanks to my parents and my sisters for their emotional support during this undertaking.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ODE	Ordinary differential equations
PDE	Partial differential equations
OS1	Operator splitting of degree one
OS2	Operator splitting of degree two
RK	Runge-Kutta method
ERK	Explicit Runge-Kutta
IRK	Implicit Runge-Kutta
DIRK	Diagonally implicit Runge-Kutta
SDIRK	Singly diagonally implicit Runge-Kutta
FE1	forward Euler of order one
ERK4	explicit Runge-Kutta of order four
IMP2	implicit Midpoint of order two
ESDIRK3	explicit first stage singly diagonally implicit Runge-Kutta of order three
ESDIRK32	embedded singly diagonally implicit Runge-Kutta
SDIRK4	singly diagonally implicit Runge-Kutta of order four

Chapter 1

Introduction

Mathematical modeling of cardiac electrical activity plays a crucial role in cardiovascular research. Many of the life-threatening heart problems are in fact related to disorder in the heart's electrical activity. Mathematical models can be used to simulate the heart activity and the effects of certain drugs designed to treat them. With the current technology in hand, the expected cost of development of a drug is often at the order of hundreds of millions of dollars [DiMasi et al., 2003]. One goal of mathematical modeling is to reduce this cost by reducing the number of physical experiments needed for designing a drug [Spiteri and Dean, 2008].

The main function of the heart is to pump the blood throughout the body. The heart consists of four chambers: right atria, left atria, right ventricle and left ventricle. The chambers are separated by atrioventricular valves. The atrioventricular valves are one way valves that allow blood to move from atria to ventricle [Katz, 2010]. Opening and closing of the valves delivers the blood around the body and back to the heart.

Electrical activity of the heart is initiated by the sinoatrial (SA) node, that

serves as a pacemaker, and propagated through the atria provoking contraction. Then the electrical signal reaches the atrioventricular (AV) node which is placed just above the ventricles. The AV node delays the electrical impulse for a brief period that allows the right and left atrium to finish emptying their blood into the two ventricles. After the delay, the electrical impulse travels through both ventricles which results in contraction of ventricles and blood is pumped into the pulmonary artery and aorta.

From a microscopic point of view, because of the difference in the net electrical charges of different ions in the cytoplasm of heart cells, a heart cell is negatively charged compared to its surroundings. This results in a potential difference across the cell membrane known as *transmembrane potential* [Sundnes, 2006]. In this thesis we study the transmembrane potential from a numerical point of view.

The mathematical models of the heart employ *ordinary differential equations* (ODEs) as well as *partial differential equations* (PDEs) to simulate the electrical activity. Models of the electrophysiology of a single cell consist of a system of ODEs only while a network of cells can be modeled via a coupled system of ODEs and PDEs [Shuaiby et al., 2011].

Simulation of the tissue scale electrophysiological models effectively is a challenging task. Indeed, the non-linearity and stiffness of the large ODE system make accurate simulations either difficult or not feasible. The goal of this research is to explore numerical algorithms, which are proper to solve both single cell and tissue scale electrophysiological models.

1.A Related Work

A large body of research has been done in the area of mathematical modeling of cardiac cells. The most notable contributions in this area include [Hodgkin and Huxley, 1952], [Noble, 1962], [DiFrancesco and Noble, 1985], [Luo and Rudy, 1991], [Luo and Rudy, 1994], [Courtemanche et al., 1998], and [Winslow et al., 1999]. Typically, the models consist of a system of nonlinear differential equations. Newer mathematical cardiac cell models capture more detailed and accurate cellular activities. Nevertheless, the accuracy usually comes at the cost of complexity of the model [MacLachlan et al., 2007]. The aim of this research is not to develop new cardiac cell models but to find efficient solutions to the existing ones.

Stiffness of the most advanced cardiac cell models makes it a challenging job to solve them efficiently. On the one hand, explicit ODE solvers require very small time steps to maintain stability, which result to inefficient solutions. On the other hand, efficient implementation of the implicit methods is a difficult task.

The research in the area of numerical simulation of cardiac cell models can be generally divided into the research on the single cell models and the research on tissue scale models. Since single cell models consist only of ODEs, the solutions to these models are supposedly simpler. Forward Euler is a common choice among the researchers to solve the single cell models, *e.g.*, see [Roth, 1995,Saleheen and Ng, 1998]. A more efficient alternative to forward Euler is the Rush and Larsen's method [Rush and Larsen, 1978], that is popularly used in the field of cardiac cell modeling, for instances see [Qu et al., 2000], [Jacquemet et al., 2003], and [Ten Tusscher et al., 2004]. Rush-Larsen method uses the

fact that while the ODE systems of the cell models are nonlinear, most of the ODEs become linear if some of the variables are assumed constants. As a result, an update formula can be derived using the analytical solution of the linear ODEs. The rest of the equations (the remaining nonlinear ones) can be solved using forward Euler. In [Spiteri and MacLachlan, 2003] the performance of Rush-Larsen method to solve the Luo-Rudy model is investigated. Usage of the methods other than forward Euler in combination with the Rush-Larsen method is also studied in a number of papers including [Sundnes et al., 2009]. In [Spiteri and Dean, 2008] the authors investigate the efficiency of implicit-explicit Runge-Kutta (IMEX-RK) splitting methods for the simulation of four cardiac electrophysiological models namely Luo-Rudy I, Courtemanche *et al.*, Winslow *et al.*, and Puglisi-Bers. In [Belhamadia et al., 2012] a nested implicit Runge-Kutta method of order 4 is employed to solve the Luo-Rudy I and Hund-Rudy models. Since the focus of the authors is on single-cell models, the results cannot be generalized to tissue-scale simulations.

Splitting methods are commonly used in tissue scale cardiac simulations in which coupled systems of ODEs and PDEs are studied [Sundnes et al., 2005, Lines et al., 2003, Santos et al., 2005]. Using operator splitting, one can split the coupled system into ODEs and PDEs. Then, each of the ODE and the PDE parts can be solved using an appropriate method. Experiments shows that solving the ODE part contributes significantly to the total work required for the simulation of the multi-cell models [Sundnes et al., 2001]. Therefore, most of the research that employ operator splitting methods focus on the ODE part. For instance in [Sundnes et al., 2001] the authors use a first and second order operator splitting method combined with an implicit Runge-Kutta solver (ESDIRK32) to solve the Winslow *et al.* model which contains 33

variables. A second-order operator splitting method for the monodomain model combined with Luo-Rudy I is studied in [Qu and Garfinkel, 1999]. In [Keener and Bogar, 1998], the authors use an implicit method for the PDEs, and an explicit method for the ODEs.

1.B Thesis Structure

The remainder of this thesis is structured into four chapters. In Chapter 2, the models of cardiac cells for both single cell and tissue scale models are presented. Chapter 3 reviews the numerical methods used in the thesis. The chapter describes the first and the second operator splitting methods as well as the ODE solvers used in the thesis. The numerical results of the thesis is presented in Chapter 4 and Chapter 5 is devoted to a conclusion.

Chapter 2

Cell Models

The mathematical models that simulate the electrical activity of the heart describe the behavior of ionic activity of either a single cell or multiple cells (*i.e.*, tissue scale models). Single cell models consist of a set of ODEs while tissue scale models consist of a coupled system of ODEs and PDEs. Monodomain (that contains a single PDE) and bidomain (that contains two PDEs) are two very well known tissue scale models. Theses two models can be coupled to system of ODE to accurately simulate the single cell behavior. The monodomain model is actually a simplification of the bidomain model. While the bidomain model is more accurate, monodomain is mostly used as it requires less computational effort. Because of the high computation demand of bidomain, the focus of this thesis is mainly on the monodomain model coupled with various single cell models. Figure 2.1 shows an overview of the cell models studied in this thesis.

In this chapter, we first describe several single cell models in Section 2.A. Then, in Section 2.B, we study the monodomain and bidomain models.

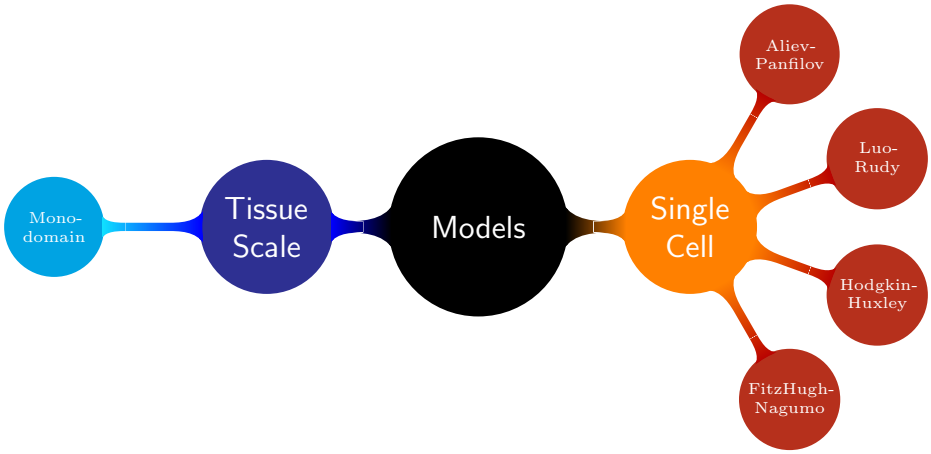


Figure 2.1: Single cell and tissue scale models studied in the thesis.

2.A Single Cell Models

In this section, we present different single cell models that is later used in Chapter 4 for the numerical results.

2.A.1 Hodgkin-Huxley Model

The Hodgkin-Huxley model was first introduced in 1952 by A.L. Hodgkin and A.F. Huxley [Hodgkin and Huxley, 1952] to describe the initiation and propagation of action potentials in the squid giant axon. In 1963, they received the Nobel Prize in Physiology or Medicine for this work. The model serves as the basis of the subsequent models for heart cells [Sundnes, 2006] and consists of four non-linear ODEs as follows,

$$\begin{cases} \dfrac{dV_m}{dt} & = \frac{-I_{ion}}{C_m} \\ \dfrac{dv_1}{dt} & = \alpha_m(1-v_1) - \beta_m v_1 \\ \dfrac{dv_2}{dt} & = \alpha_h(1-v_2) - \beta_h v_2 \\ \dfrac{dv_3}{dt} & = \alpha_n(1-v_3) - \beta_n v_3 \end{cases} \tag{2.1}$$

where,

$$\begin{aligned} \alpha_m &= \frac{-0.1(V_m+50)}{e^{-0.1*(V_m+50)}-1} \\ \beta_m &= 4e^{-(V_m+75)/18} \\ \alpha_h &= 0.07e^{-(V_m+75)/20} \\ \beta_h &= \frac{1}{e^{-0.1(V_m+45)}+1} \\ \alpha_n &= \frac{-0.01*(V_m+65)}{e^{-0.1*(V_m+65)}-1} \\ \beta_n &= 0.125e^{(V_m+75)/80} \\ I_{Na} &= g_{Na}\, v_1^3\, v_2\, (V_m - E_{Na}) \\ I_K &= g_K\, v_3^4\, V_m - E_K \\ I_L &= g_L(V_m - E_L) \\ I_{ion} &= I_{Na} + I_K + I_L\,. \end{aligned}$$

The following values is used for the constants: $E_R = -75$, $C_m = 1$, $E_{Na} = E_R + 115$, $E_K = E_R - 12$, $E_L = E_R + 10.613$, $g_{Na} = 120$, $g_K = 36$, and $g_L = 0.3$. A brief description of the variables follows.

- V_m: Transmembrane potential
- v_1, v_2, v_3: Gate variables

- I_{ion}: Total ionic current across the membrane ($\frac{\mu A}{cm^2}$)
- I_{Na}: Sodium current across the membrane ($\frac{\mu A}{cm^2}$)
- I_K: Potassium current across the membrane ($\frac{\mu A}{cm^2}$)
- I_L: Leakage current across the membrane ($\frac{\mu A}{cm^2}$)
- g_{Na}: Sodium conductance ($\frac{mS}{cm^2}$)
- g_K: Potassium conductance ($\frac{mS}{cm^2}$)
- g_{Na}: Sodium conductance ($\frac{mS}{cm^2}$)
- C_m: Membrane capacity per unit area ($\frac{\mu F}{cm^2}$)
- α_n and β_n: Rate constants which vary with voltage but not with time ($\frac{1}{mS}$)

Figure 2.2 shows the transmembrane potential over time produced by the Hodgkin-Huxley model.

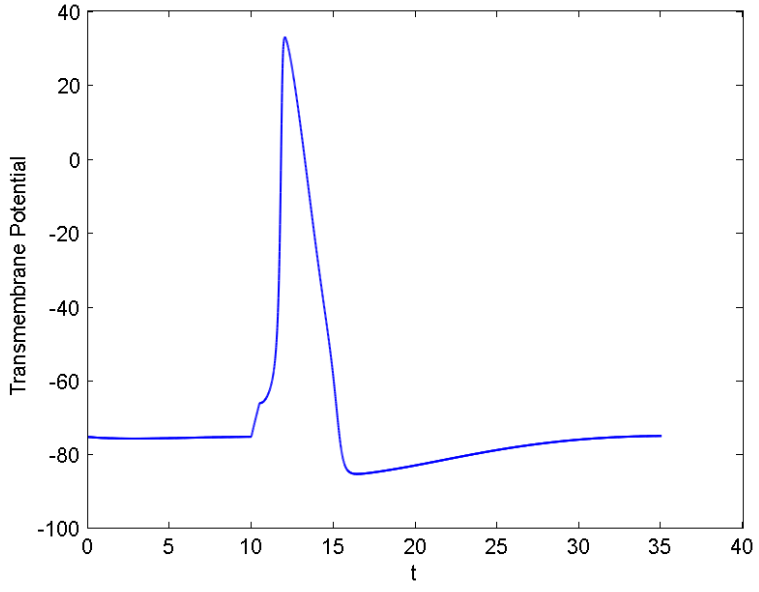


Figure 2.2: Transmembrane potential over time in the Hodgkin-Huxley model

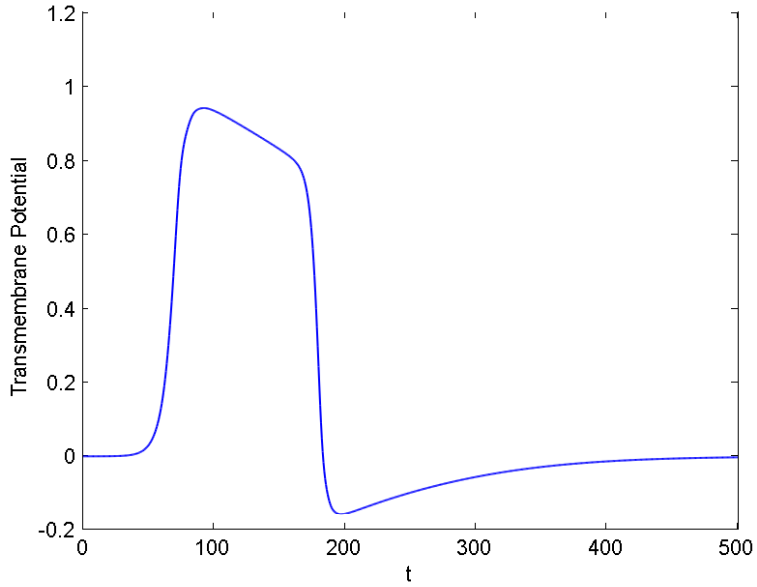


Figure 2.3: Transmembrane potential over time in the FitzHugh-Nagumo model

2.A.2 FitzHugh-Nagumo Model

FitzHugh-Nagumo is one of the simplest cardiac cell models that has only two variables. The model is a simplified version of the Hodgkin-Huxley model and consist of,

$$\begin{cases} \dfrac{dV_m}{dt} &= k_1 V_m (V_m - a)(1 - V_m) - k_2 w + I_{st} \\ \dfrac{dw}{dt} &= b(V_m - k_3 w), \end{cases} \tag{2.2}$$

where V_m is the transmembrane potential, w is the dimensionless recovery variable, I_{st} is the stimulus current, and a, b, k_1, k_2, and k_3 are the parameters of the model [Sundnes, 2006]. Figure 2.3 shows the transmembrane potential produced by the FitzHugh-Nagumo model.

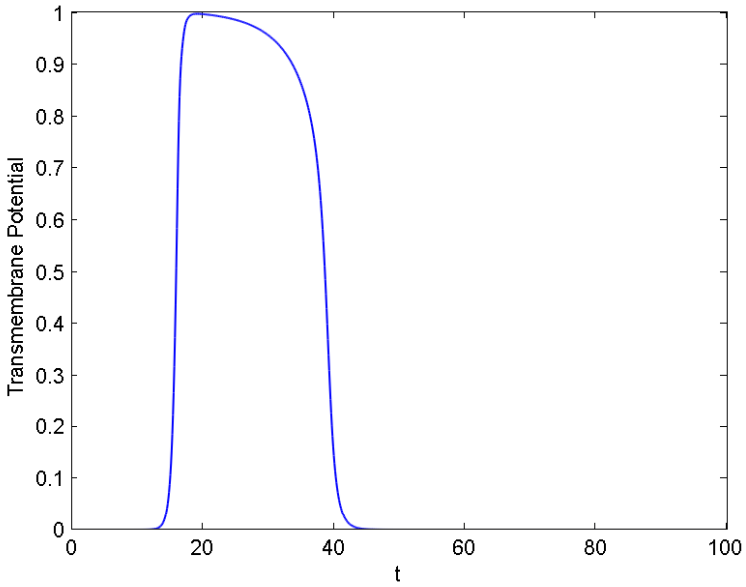


Figure 2.4: Transmembrane potential over time in the Aliev-Panfilov model

2.A.3 Aliev-Panfilov Model

Aliev and Panfilov proposed their model in 1996. The Aliev-Panfilov model reproduces more realistic shapes of the cardiac action potential [Belhamadia et al., 2009] and consists of the following equations [Aliev and Panfilov, 1996],

$$\begin{cases} \dfrac{dv}{dt} &= \left(\epsilon + \frac{\mu_1 v}{\mu_2 + V_m}\right)\Big(- v - kV_m\left(V_m - a - 1\right)\Big) \\ \dfrac{dV_m}{dt} &= kV_m(V_m - a)(1 - V_m) - V_m v \end{cases} \tag{2.3}$$

in which, V_m is the transmembrane potential, v is the dimensionless recovery variable, and a, k, ϵ, μ_1, μ_2 are parameters to the model. An example of the evolution of V_m over time for this model is given in Figure 2.4.

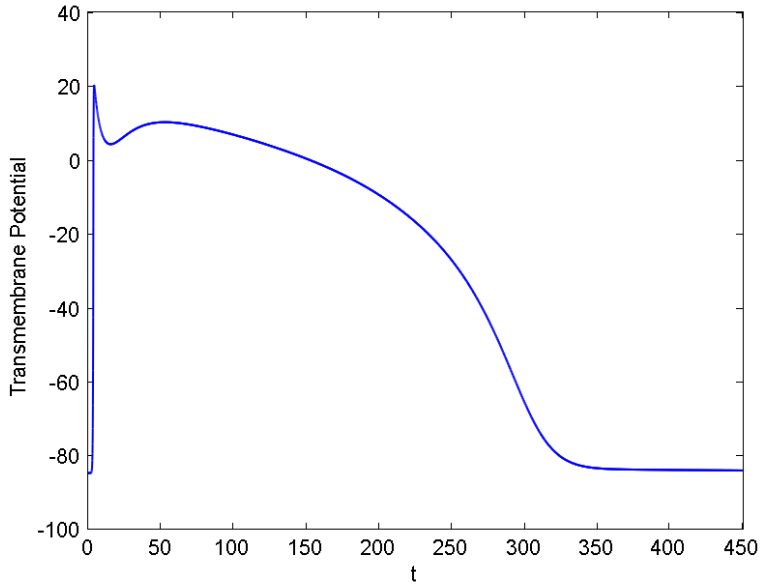


Figure 2.5: Transmembrane potential over time in the Luo-Rudy model

2.A.4 Luo-Rudy I Model

The original version of the Luo-Rudy model, known as Luo-Rudy I model, is a model of guinea pig ventricular action potentials and was introduced in 1991 [Luo and Rudy, 1991]. Compared to the other models we described above, this model gives a more detailed description of the ionic currents across the membrane. For an individual cardiac cell we have that the transmembrane potential V_m, typically measured in mV, satisfies $\frac{dV_m}{dt} = \frac{-1}{C_m}(I_{ion} + I_{st})$ in which, C_m is the membrane capacitance, I_{ion} is the total transmembrane ionic current, and I_{st} is the stimulus current [Spiteri and Dean, 2008].

The Luo-Rudy I model itself consists of 8 nonlinear ODEs. More detail is provided in Appendix A where the C++ code is presented. An example of the evolution of V_m over time for this model is given in Figure 2.5.

2.B Tissue Scale Models

To model the electrical activity across a network of cells, a single cell model should be coupled with one or more PDEs. In this research, we are specifically interested in a model known as the *monodomain* model that can be described as,

$$\begin{cases} \dfrac{dv}{dt} = f(u,v), & x \in H \\ \chi C \dfrac{du}{dt} + \chi I_{ion}(u,v) = \nabla.(\sigma_I \nabla u), & x \in H \\ n.\nabla u = 0, & x \in \partial H \\ n.\nabla v = 0, & x \in \partial H \end{cases} \tag{2.4}$$

where the variables are defined as follows,

- u: transmembrane potential
- v: vector of gate variables
- H: physical domain of interest
- ∂H: boundary of H
- n: an outward pointing normal vector of the boundary
- σ_I: symmetric conductivity tensor
- C: Capacitance
- χ: Membrane area to volume ratio
- The functions $I_{ion}(u,v)$ and $g(u,v)$ depend on the ionic model

Note that u and v are functions of time and space. As in [Sundnes et al., 2001], we use the following values for the monodomain parameters in our simulations: $C = 1\mu F/cm^2$, $\chi = 2000cm^{-1}$, $\sigma_I = 1.3514mS/cm$.

As mentioned earlier, the monodomain model must be coupled with one of the single cell models to have a complete tissue scale model. For instance monodomain coupled with Hodgkin-Huxley is as follows,

$$\begin{cases} \dfrac{dV_m}{dt} &= \frac{-I_{ion}}{C_m} + \nabla.(\sigma_I \nabla u), \qquad x \in H \\ \dfrac{dv_1}{dt} &= \alpha_m(1 - v_1) - \beta_m v_1 \\ \dfrac{dv_2}{dt} &= \alpha_h(1 - v_2) - \beta_h v_2 \\ \dfrac{dv_3}{dt} &= \alpha_n(1 - v_3) - \beta_n v_3 \end{cases} \tag{2.5}$$

The variables are as already described in Section 2.A.1.

Chapter 3

Numerical Methods

Since the electrical activity models of the heart deals with a coupled system of ODEs and PDEs, in this chapter, we first describe several methods to solve the ODE systems and then we will present the methods that can solve the coupled systems of ODEs and PDEs.

In one part of the experiment, we compare several Runge-Kutta ODE solvers to simulate the single cell models. Specifically, we use the following ODE solvers: forward Euler of order one (FE1), explicit Runge-Kutta of order four (ERK4), implicit Midpoint of order two (IMP2), explicit first stage singly diagonally implicit Runge-Kutta of order three (ESDIRK3), embedded singly diagonally implicit Runge-Kutta (ESDIRK32), singly diagonally implicit Runge-Kutta of order four (SDIRK4), embedded Dormant-Prince 45.

In the other part, we use operator splitting methods of the first and second order to solve the tissue scale (*i.e.*, multi-cell) models. In operator splitting methods, we split the mathematical equations into two parts. The first part consists of solving the ODEs with an appropriate solver and the second consists of solving the PDE with implict methods.

3.A Numerical Methods for ODEs

It is well-known that the solution of the ODEs contributes significantly to the total work of the simulation [Sundnes et al., 2001]. Therefore, the focus of our research is to explore the efficiency of different ODE solvers for the problems. To that end, we try several ODE solvers of different orders and accuracies. The models we study in this thesis usually contain stiff equations. There are different definitions of stiffness in the literature. One typical definition is based on the ratio of the smallest and the largest negative real parts of the eigenvalue of the Jacobian matrix. It is commonly assumed that no real part is positive. More formally, suppose we have a system $x' = f(x)$ and $x \in \mathbb{R}^n$. The system is called stiff at $x = x_0$ if:

(1) The system at $x = x_0$ is stable, *i.e.*, all eigenvalues, λ_k , of the Jacobian matrix $J = Df(x_0)$ have negative real parts.

(2) The ratio,

$$\mathcal{L} = \frac{\max_k |Re\lambda_k|}{\min_k |Re\lambda_k|} \tag{3.1}$$

is sufficiently large. The ratio $\mathcal{L}$ is known as the *stiffness index*. A larger ratio results in a stiffer problem. Another common definition is that certain numerical methods for solving the equations are numerically unstable unless extremely small step sizes are considered. In our research, we consider the latter definition.

The widely used Runge-Kutta(RK) class is used in this research. However, because of the stiffness of the ODEs and the poor stability properties of the explicit solvers we mainly focus on implicit RK methods [Butcher, 2008, Hairer and Wanner, 2004]. A general s stage RK method has the form [Sundnes,

2006, Hairer and Wanner, 2004, Hairer et al., 1993],

$$y_n = y_{n-1} + \Delta t_n \sum_{i=1}^{s} b_i K_i, \tag{3.2}$$

where for $i = 1, 2, ..., s$,

$$K_i = f(t_{n-1} + \Delta t_n c_i, y_{n-1} + \Delta t_n \sum_{j=1}^{s} a_{ij} K_j), \tag{3.3}$$

which can be summarized via the Butcher tableau,

$$\begin{array}{c|c} c & \mathcal{A} \\ \hline & b^T \end{array} = \begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array} \tag{3.4}$$

A RK method is called an explicit RK (ERK) if $\mathcal{A}$ is strictly lower triangular; otherwise it is called an implicit RK (IRK). If an IRK method has a matrix $\mathcal{A}$ that is lower triangular, then it is called a diagonally IRK (DIRK) method. If we additionally have all a_{ii} to be equal, it is called singly DIRK (SDIRK) method [Sundnes et al., 2001, Hairer et al., 1993]. As an example, Table 3.1 shows the Butcher tableau of the ERK4 method, that is an explicit method of order 4.

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	1/3	1/3	1/6

Table 3.1: Butcher tableau of ERK4

As another example, the Butcher tableau of SDIRK4 is shown in Table 3.2. SDIRK methods are often considered efficient when solving stiff systems [Hairer et al., 1993]. A SDIRK4 is a 5-stages method of order 4 and the Butcher tableau of this method is presented in Table 3.2.

c_1	λ				
c_2	a_{21}	λ			
c_3	a_{31}	a_{32}	λ		
c_4	a_{41}	a_{42}	a_{43}	λ	
c_5	a_{51}	a_{52}	a_{53}	a_{54}	λ
	b_1	b_2	b_3	b_4	b_5

Table 3.2: SDIRK4 Butcher tableau

The coefficients a_{ij} , b_i and are provided in [Hairer et al., 1993].

Adaptive step-size methods are also important class of the ODE solvers. In these methods, the step-size is adjusted according to the rate of the change of the solution, *i.e.*, if the solution is changing rapidly a smaller step-size is used while a larger step-size is used for the parts of the solution that slowly change [LU, 2011].

An embedded RK method is an adaptive method that uses two ordinary Runge-Kutta methods, one with order p and one with order $p-1$, to estimate the local truncation error of a single RK step. The Butcher tableau of an embedded RK method is given by,

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ c_p & a_{p1} & \cdots & a_{pp} \\ \hline & b_1 & \dots & b_p \\ & b_1^* & \dots & b_p^* \end{array} \tag{3.5}$$

Given upper and lower local error bounds denoted by e_U and e_L and upper and lower bounds on the step size denoted by h_U and h_L, in every iteration of the method the following steps are followed,

1. Compute the values y_{n+1} and y^*_{n+1} using,

$$y_{n+1} = y_n + \sum_{i=1}^{p} b_i K_i \tag{3.6}$$

$$y^*_{n+1} = y_n + \sum_{i=1}^{p} b^*_i K_i \tag{3.7}$$

   where K_i's are given by Eq. (3.3).

2. Compute the local error estimate $e_{n+1} = |y_{n+1} - y^*_{n+1}|$.

3. If e_{n+1} is within the bound e_L and e_U, propagate one of the values y_{n+1} or y^*_{n+1} (the one with the higher order) as the answer of the iteration and continue to the next iteration.

4. If $e_{n+1} > e_U$ reject the iteration, choose a smaller step size[1], and recompute the current iteration from the beginning.

5. If $e_{n+1} < e_L$ accept the iteration, choose a larger step size[2], and continue with the next iteration.

In summary, Table 3.3 shows the list of ODE solvers used in this research with a brief description.

[1] In our code, the smaller step size is computed as $h_{\text{new}} = h_{\text{old}} \times 0.8 \left(\frac{e_U}{e_{n+1}}\right)^{1/4}$ and we always ensure that h_{new} is within h_L and h_U.

[2] In our code, the larger step size is computed as $h_{\text{new}} = h_{\text{old}} \times 0.8 \left(\frac{e_L}{e_{n+1}}\right)^{1/4}$ and we always ensure that h_{new} is within h_L and h_U.

Method	Description
FE1	Explicit RK of order 1
ERK4	Explicit RK of order 4
IMP2	Fully implicit method of order 2 [LU, 2011]
ESDIRK3	Explicit first stage, singly diagonally implicit RK of order 3 [Van Zuijlen and Bijl, 2005]
SDIRK4	Singly diagonally implicit RK of order 4
ESDIRK23A	A third order singly diagonally implicit RK with adaptive step size
Dormant-Prince45	An adaptive explicit RK method with order 5

Table 3.3: List of ODE solvers used in this research

3.A.1 Implicit versus Explicit Solvers

An ODE solver is called *explicit* if the value of y_{n+1} is given directly from known quantities and previous values of y_n. In the other case, in which y_{n+1} is found by solving a system of (nonlinear) equations, the method is called *implicit* (because y_{n+1} is given implicitly). Because of poor stability of explicit methods, the focus of this thesis is mainly on implicit methods. To solve the nonlinear equations, Newton iterations is commonly used as the standard procedure [Sundnes et al., 2001]. To do so, after computing the initial value of k_i^0, the following formulas are used to compute the rest of the values,

$$(I - \Delta t\gamma J)\Delta k_i^m = -k^m + f\left(t_n + \Delta t c_i, y_n + \Delta t \sum_{j=1}^{i-1} a_{ij} k_i^m + \gamma k_i^m\right) \tag{3.8}$$

$$k_i^{m+1} = k_i^m + \Delta k_i^m, \tag{3.9}$$

for $i = 2, 3, \ldots$ until the termination criteria is satisfied. The matrix J is the Jacobian of the right hand function $f(.)$. In standard Newton method, this

matrix should be computed for every iteration by,

$$J = \frac{\partial f}{\partial y}(t_n + \Delta t c_i, y_n + r\Delta t), \tag{3.10}$$

where r is the collection of terms in Eq. (3.8). However, since computation of J using the above formula is computationally costly, the following approximation is used in my experiments,

$$J \approx \frac{\partial f}{\partial y}(t_n, y_n). \tag{3.11}$$

3.B Numerical Methods for PDEs

In this section, we present the numerical methods that we employ to solve the PDE part of the coupled system. To simplify the presentation consider the boundary-initial value problem,

$$\begin{cases} u_t = cu_{xx}, & 0 < x < 1, t > 0 \\ u(0,t) = u(1,t) = 0 & \text{(boundary conditions)} \\ u(x,0) = f(x) & \text{(initial condition)} \end{cases} \tag{3.12}$$

where $u = u(x,t)$ and c is a constant. To solve this problem numerically, x and t are discretized such that,

$$x_j = j\Delta x, \qquad j = 0, 1, 2, ... \tag{3.13}$$

and

$$t_n = n\Delta t, \qquad n = 0, 1, 2, ... \tag{3.14}$$

If we use the backward difference at time t_{n+1} and a second-order central difference for the space derivative at position x_j we have the following scheme known as backward Euler,

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2} \tag{3.15}$$

We can obtain u_j^{n+1} by solving the following system,

$$(1 + 2\lambda)u_j^{n+1} - \lambda u_{j-1}^{n+1} - \lambda u_{j+1}^{n+1} = u_j^n \tag{3.16}$$

where $\lambda = \frac{\Delta t}{\Delta x^2}$.

If we use the central difference at time $t_{n+1/2}$ and a second-order central difference for the space derivative at position x_j we have the scheme,

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{1}{2}\left(\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2} + \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}\right) \tag{3.17}$$

known as the Crank-Nicolson method. We can obtain u_j^{n+1} by solving the following system,

$$(1 + 2r)u_j^{n+1} - ru_{j-1}^{n+1} - ru_{j+1}^{n+1} = (1 - 2r)u_j^n + ru_{j-1}^n + ru_{j+1}^n \tag{3.18}$$

where $r = \frac{\Delta t}{2\Delta x^2}$. This system has the matrix form of $Au = b$ [LeVeque, 2007]. We use direct method (LU) to solve this system.

3.C Numerical Methods for the Monodomain Model

We consider two type of numerical methods, known as operator splitting and semi-implicit finite difference methods, to solve the coupled system.

3.C.1 Operator Splitting Methods

With operator splitting technique, a coupled system of ODEs and PDEs can be split into smaller parts which are easier to solve. For most of the cases, instead of using one single method to solve the whole system of equations, it would be more efficient to use different numerical methods for different parts [Sundnes et al., 2001]. There are several different operator splitting techniques but in this research we are interested in first order operator splitting (OS1) and second order operator splitting (OS2) methods [Sundnes, 2006].

OS1 is a two-step method with a first order accuracy while OS2 is a three step method with a second order accuracy. OS1 and OS2 are general techniques to solve the system of coupled PDEs and ODEs but we describe the methods in the context of the monodomain problem only. Using this method for the monodomain problem, for OS1 we have the following steps [Sundnes, 2006],

1. Solve the ODEs

$$\chi C \frac{du}{dt} = -\chi I_{ion}(u, v),\ u(t_n) = u^n$$

$$\frac{dv}{dt} = f(u, v),\ v(t_n) = v^n$$

for $t_n < t \leq t_n + \Delta t$. The solutions of this step are denoted by $\tilde{u}$ and

$v(t_n + \Delta t)$

2. Solve the linear PDE

$$\chi C \frac{du}{dt} = \nabla.(\sigma_I \nabla u),\ u(t_n) = \tilde{u}$$

for $t_n < t \leq t_n + \Delta t$. The resulting solution $u(t_n + \Delta t)$ is denoted by u^{n+1}

in which Δt is the time step.

For the OS2 method, the steps are as follows,

1. Solve the ODEs

$$\chi C \frac{du}{dt} = -\chi I_{ion}(u, v),\ u(t_n) = u^n$$

$$\frac{dv}{dt} = f(u, v),\ v(t_n) = v^n$$

for $t_n < t \leq t_n + \frac{1}{2}\Delta t$. The solutions of this step are denoted by $\tilde{u}^n$ and $\tilde{v}^n$.

2. Solve the linear PDE

$$\chi C \frac{du}{dt} = \nabla.(\sigma_I \nabla u),\ u(t_n) = \tilde{u}^n$$

for $t_n < t \leq t_n + \Delta t$. The resulting solution is denoted by $\bar{u}^{n+1}$

3. Solve the system

$$\chi C \frac{du}{dt} = -\chi I_{ion}(u, v),\ u(t_n + \frac{1}{2}\Delta t) = \bar{u}^{n+1}$$

$$\frac{dv}{dt} = f(u,v),\ v(t_n + \frac{1}{2}\Delta t) = \tilde{v}^n$$

for $t_n + \frac{1}{2}\Delta t \leq t_n + \Delta t$. The resulting solutions are u^{n+1} and v^{n+1} at $t = t_n + \Delta t$.

in which Δt is the time step.

For some simple problems the subproblems can be solved analytically but for most of the cases, we have to solve the subproblems numerically.

3.C.2 Semi-implicit Methods

In order to verify the correctness of our implementation for the operator splitting methods, the semi-implicit method is used for comparison. The semi-implicit method is useful for some particular cases where the PDE is coupled with only one ODE. Given a non-linear PDE of the form,

$$\begin{cases} u_t = \nabla^2 u + f(u,v) \\ v_t = g(u,v), \end{cases} \tag{3.19}$$

with the Dirichlet boundary condition of,

$$\begin{cases} u(x_0,t) = h_1 \\ u(x_N,t) = h_2 \end{cases} \tag{3.20}$$

We consider discretization of Eq. (3.19) which replace the solution of $u(t)$ with the approximate solution of $u^n \approx u(t_n)$ at times $t_0 < t_1 < \ldots < t_n < \ldots$ and $t_n = n\Delta t$ for a constant Δt. A semi-implicit method deals with $\nabla^2 u$ implicitly and with $f(u,v)$ explicitly, *i.e.*, at time t_n we have the following

scheme,

$$\begin{cases} \dfrac{u^{n+1}-u^n}{\Delta t} = \nabla^2 u^{n+1} + f(u^n, v^n) \\ \dfrac{v^{n+1}-v^n}{\Delta t} = g(u^n, v^n) \end{cases} \tag{3.21}$$

If we also partition the domain in space using a mesh $x_0, \dots x_N$ we have the following scheme,

$$\begin{cases} \dfrac{u_i^{n+1}-u_i^n}{\Delta t} = \dfrac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + f(u_i^n, v_i^n) \qquad i = 0 \dots N \\ \dfrac{v_i^{n+1}-v_i^n}{\Delta t} = g(u_i^n, v_i^n) \end{cases} \tag{3.22}$$

Letting $\lambda = \frac{\Delta t}{\Delta x^2}$ we can rewrite Eq. (3.23) as,

$$\begin{cases} -\lambda u_{i+1}^{n+1} + (1+2\lambda)u_i^{n+1} - \lambda u_{i-1}^{n+1} = \Delta t f(u_i^n, v_i^n) + u_i^n \qquad i = 0 \dots N \\ v_i^{n+1} - v_i^n = g(u_i^n, v_i^n)\Delta t \end{cases} \tag{3.23}$$

The linear system of the first equation in Eq. (3.23) is,

$$\begin{cases} i = 1 & -(\lambda u_2^{n+1} - (1+2\lambda)u_1^{n+1} + \lambda u_0^{n+1}) = \Delta t f(u_1^n, v_1^n) + u_1^n \\ i = 2 & -(\lambda u_3^{n+1} - (1+2\lambda)u_2^{n+1} + \lambda u_1^{n+1}) = \Delta t f(u_2^n, v_2^n) + u_2^n \\ \quad \vdots & \\ i = N-1 & -(\lambda u_N^{n+1} - (1+2\lambda)u_{N-1}^{n+1} + \lambda u_{N-2}^{n+1}) = \Delta t f(u_{N-1}^n, v_{N-1}^n) + u_{N-1}^n \end{cases} \tag{3.24}$$

with boundary conditions of $u_0^{n+1} = h_1$ and $u_N^{n+1} = h_2$. The above system has

the matrix form of $A\mathbf{u} = F$ where $\mathbf{u}$, A, and F are defined as follows,

$$\mathbf{u} = \begin{bmatrix} u_0 \\ \vdots \\ u_N \\ v_0 \\ \vdots \\ v_N \end{bmatrix} \in \mathbb{R}^{2N+2} \tag{3.25}$$

$$A = \begin{pmatrix} 1 & & & & & & \\ -\lambda & 1+2\lambda & -\lambda & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & -\lambda & 1+2\lambda & -\lambda & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \\ & & & & & & & & 1 \end{pmatrix} \in \mathbb{R}^{2N+2\times 2N+2} \tag{3.26}$$

$$F = \begin{bmatrix} h_1 \\ \Delta t f(u_1^n, v_1^n) + u_1^n \\ \Delta t f(u_2^n, v_2^n) + u_2^n \\ \vdots \\ \Delta t f(u_{N-1}^n, v_{N-1}^n) + u_{N-1}^n \\ h_2 \\ \Delta t g(u_0^n, v_0^n) + v_0^n \\ \Delta t g(u_1^n, v_1^n) + v_1^n \\ \vdots \\ \Delta t g(u_N^n, v_N^n) + v_N^n \end{bmatrix}^T \in \mathbb{R}^{2N+2} \tag{3.27}$$

Chapter 4

Results

In this chapter, simulation results of several different single cell models as well as the results of monodomain model for 1D and 2D are presented. To solve the single cell models, which only consist of nonlinear ODEs, several different ODE solvers that were introduced in Chapter 3 are used. To solve the monodomain model, we use first and second order operator splitting techniques, for both 1D and 2D models, to split the model into the ODE and PDE parts. Backward Euler and Crank-Nicolson methods are used to solve the PDE part. To investigate the efficiency of ODE solvers, we compare the result of various implicit and explicit Runge-Kutta ODE solvers. Specifically, we use the following ODE solvers: forward Euler of order one (FE1), explicit Runge-Kutta of order four (ERK4), implicit Midpoint of order two (IMP2), explicit first stage singly diagonally implicit Runge-Kutta of order three (ESDIRK3), embedded singly diagonally implicit Runge-Kutta (ESDIRK32), singly diagonally implicit Runge-Kutta of order four (SDIRK4). In addition, we use embedded Dormant-Prince 45 to generate the reference solutions of ODEs. To verify the correctness of our implementations, we use a number of test problems, most of which have

known analytical solutions.

Recall that most of the computation cost is consumed to solve the ODE part of the coupled systems. Therefore, the study is mostly focused on solving the ODE part. As expected, the results show that implicit ODE solvers, specially for stiffer ODEs, give a much more accurate result. In addition, implicit ODE solvers have a more stable behavior for stiff ODEs.

To evaluate the accuracy of suggested methods, we use e_∞ and e_2 errors defined as follows,

$$e_\infty = \max |V_{\text{ref}} - V_{\text{m}}| \,. \tag{4.1}$$

$$e_2 = \sqrt{\int_{t_i}^{t_f} |V_{\text{ref}} - V_{\text{m}}|^2} \,. \tag{4.2}$$

where V_{ref} and V_{m} are the reference solution and obtained numerical solution vectors respectively and are reinterpolated at N equally spaced points between t_i and t_f.

4.A Numerical Results for ODEs

In this section, we present the result of applying the ODE solvers described in Section 3.A to several single cell models. To verify the correctness of the algorithms of our ODE solvers, we first apply the algorithms to a number of test problems. Then, we present the results of the single cell models.

4.A.1 Test Problems for ODE Solvers

The ODE solvers are verified using several test problems. However, for brevity, only the results of two of them are presented here. The first test problem is an analytically solvable ODE. The second one is the well-known Van der Pol equations. A brief description of each test problem follows.

Problem 1: The first test problem is a function of one variable and consist of the following equations,

$$\begin{aligned} y' &= -200(y - \cos(t)), \qquad 0 \leq t \leq 1.5, \\ y(0) &= 0, \end{aligned} \tag{4.3}$$

with an exact solution of,

$$y(t) = \frac{200(-200e^{-200t} + \sin(t) + 200\cos(t))}{400001}\,. \tag{4.4}$$

Figure 4.1 shows the plot of different ODE solvers applied to the test problem 1. The plot shows that, compared to the other methods, FE1 has poor stability properties. Table 4.1 shows the details of the simulation. Note that the error ratios follow the expected order of the methods.

Problem 2:

For the second test problem, we consider the well known Van der Pol equations. The equations are offered by the Dutch physicist Balthasar Van der Pol in 1920 as a description for the circuit of a vacuum tubes.

Method	**Time Steps**	**Newton Iterations**	e_∞	$\frac{e_\infty[dt]}{e_\infty[dt/2]}$
FE1	1.50×10^{-3}	-	3.00×10^{-7}	
	7.50×10^{-4}	-	1.50×10^{-7}	1.998
	3.75×10^{-4}	-	7.55×10^{-8}	1.997
ERK4	1.50×10^{-3}	-	1.94×10^{-10}	
	7.50×10^{-4}	-	1.13×10^{-11}	17.10
	3.75×10^{-4}	-	7.10×10^{-13}	15.95
IMP2	1.50×10^{-3}	2000	2.22×10^{-8}	
	7.50×10^{-4}	4000	5.55×10^{-9}	4.0001
	3.75×10^{-4}	7998	1.38×10^{-9}	4.0000
ESDIRK3	1.50×10^{-3}	5043	4.13×10^{-11}	
	7.50×10^{-4}	10072	5.89×10^{-12}	7.344
	3.75×10^{-4}	20117	7.35×10^{-13}	8.025
SDIRK4	1.50×10^{-3}	9933	1.67×10^{-11}	
	7.50×10^{-4}	18051	1.13×10^{-12}	14.81
	3.75×10^{-4}	34596	0.75×10^{-13}	15.20

Table 4.1: The result of test problem 1 solved with different ODE solvers

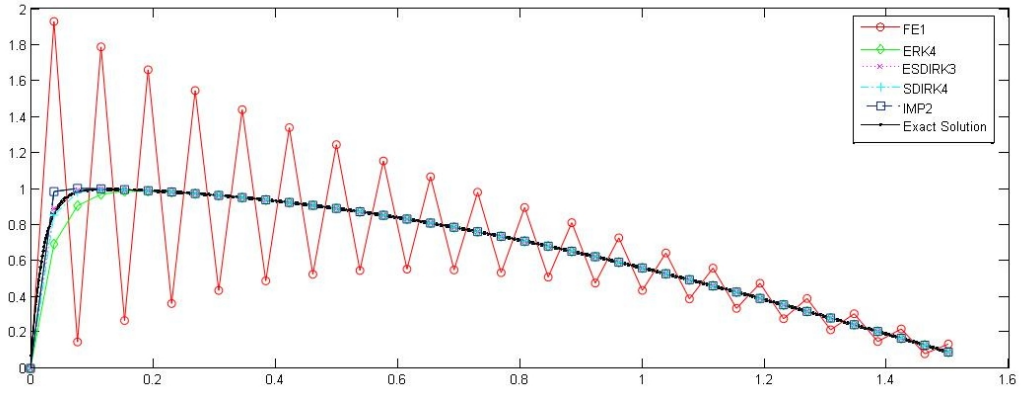


Figure 4.1: Comparison of different ODE solvers for test problem 1: $y' = -200(y - \cos(t)),\ y(0) = 0,\ 0 \leq t \leq 1.5, dt = 1.5 \times 10^{-3}$

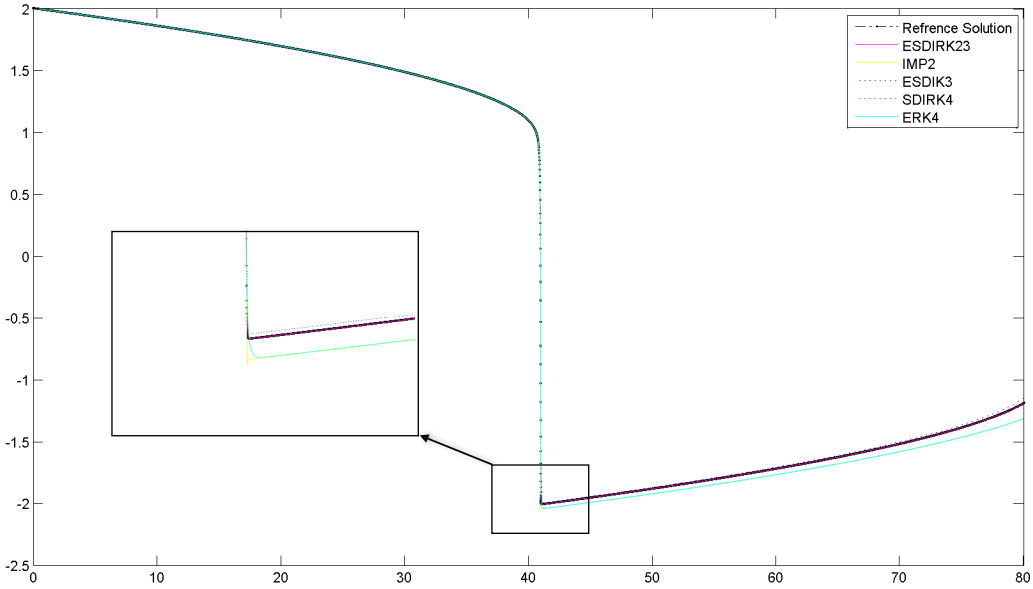


Figure 4.2: Comparison of different ODE solvers for test problem 2: Van der Pol equations for $dt = 1.25 \times 10^{-2}$

Method	**Time Steps**	**Newton Iterations**	e_∞	$\frac{e_\infty[dt]}{e_\infty[dt/2]}$
IMP2	1.25×10^{-2}	12912	1.487×10^{-1}	
	6.25×10^{-3}	25716	3.722×10^{-2}	3.99
	3.12×10^{-3}	51336	9.292×10^{-3}	4.00
ESDIRK3	1.25×10^{-2}	28566	5.043×10^{-2}	
	6.25×10^{-3}	52972	5.332×10^{-3}	9.4591
	3.12×10^{-3}	94777	6.405×10^{-4}	8.3249
SDIRK4	1.25×10^{-2}	37938	1.633×10^{-3}	
	6.25×10^{-3}	70445	1.674×10^{-4}	13.8512
	3.12×10^{-3}	135760	1.079×10^{-5}	15.5163
ESDIRK23A	$[10^{-3}, 10^{-2}]^*$	35595	5.560×10^{-3}	
	$[10^{-3}, 10^{-2}]^*$	36360	5.661×10^{-4}	-
	$[10^{-3}, 10^{-2}]^*$	37777	4.501×10^{-4}	-

Table 4.2: The result of test problem 2 solved with different ODE solvers. *Note that ESDIRK23A is an adaptive method. Therefore, the table shows the range of Time Steps used. The chosen tolerances for the entries of the ESDIRK23A is set to 10^{-2}, 10^{-4}, and 10^{-6} respectively.

$$
\begin{aligned}
y_1' &= y_2 & y_1(0) &= 2 \\
y_2' &= \mu(1 - y_1^2)y_2 - y_1 & y_2(0) &= 0\,.
\end{aligned}
\tag{4.5}
$$

We choose $0 \leq t \leq 80$ and $\mu = 50$ in our experiment. Figure 4.2 and Table 4.2 show the result of different ODE solvers applied to problem 2. The reference solution is obtained by applying an adaptive Dormant-Prince 45 with very small tolerance. To ensure the precision of the result, two solutions with different tolerances of 10^{-14} and 10^{-15} are obtained. The errors between the solutions are as follows: $e_\infty = 3.81465 \times 10^{-9}$ and $e_2 = 3.89881 \times 10^{-9}$. The higher precision solution (*i.e.*, the one with tolerance 10^{-15}) is then chosen as the reference solution. Note that the error ratios of Table 4.2 converge to expected order of the methods. Comparing the results of SDIRK4 and ESDIRK23A from Table 4.2 it can be seen that ESDIRK23A, which is an adaptive method, can obtain almost the same error value with fewer number of Newton iterations.

4.A.2 Result of Single Cell Models

In this section, we present the result of applying different ODE solvers to solve the Luo-Rudy I model. The reference solutions are generated by applying an adaptive Dormant-Prince 45 with very small tolerances. Two solutions with different tolerances of 10^{-16} and 10^{-17} are obtained where the e_∞ error between the solutions is 4.84034×10^{-8}. Then, the solution with the smaller tolerance is selected as the reference solution.

Table 4.3 show the results of the Luo-Rudy I model. The results show that

Method	**Time Steps**	**Newton Iterations**	e_∞	$\frac{e_\infty[dt]}{e_\infty[dt/2]}$
IMP2	1.00×10^{-1}	9408	1.90	
	5.00×10^{-2}	18059	0.47	4.05
	2.50×10^{-2}	36072	0.11	3.98
ESDIRK3	1.00×10^{-1}	22881	1.63	
	5.00×10^{-2}	44844	0.18	9.00
	2.50×10^{-2}	88085	0.02	8.64
SDIRK4	1.00×10^{-1}	31733	9.19×10^{-2}	
	5.00×10^{-2}	58750	7.69×10^{-3}	18.38
	2.50×10^{-2}	113456	5.98×10^{-4}	15.49
ERK4	1.00×10^{-1}	-	-	
	5.00×10^{-2}	-	-	-
	2.50×10^{-2}	-	-	-
	1.25×10^{-2}	-	2.30×10^{-4}	-
	6.25×10^{-3}	-	1.33×10^{-5}	17.23
ESDIRK23A	$[10^{-3}, 10^{-1}]^*$	22872	2.19×10^{-2}	
	$[10^{-3}, 10^{-1}]^*$	23053	2.92×10^{-3}	-
	$[10^{-3}, 10^{-1}]^*$	24222	1.71×10^{-4}	-

Table 4.3: The result of simulation for the Luo-Rudy I model. ERK4 is not stable for time steps $> 1.25 \times 10^{-2}$. *Note that ESDIRK23A is an adaptive method. Therefore, the table shows the range of Time Steps used. The chosen tolerances for the entries of the ESDIRK23A is set to 10^{-1}, 10^{-2}, and 10^{-4} respectively.

for ERK4, which is an explicit method, we need to choose very small time steps to have a stable output. The table also shows that SDIRK4 and ESDIRK23A give the most accurate results compared to other methods, between which ESDIRK23A need fewer Newton iterations.

4.B Result of One Dimensional Monodomain Model

The results of one dimensional monodomain model are presented in this section. Both OS1 and OS2 methods, as well as semi-implicit method are used to solve coupled systems of ODE and PDE in one dimension. Note that semi-implicit method is only used in the case that the system has one ODE. To verify the correctness of our programs, we first test our programs against two test problems with exact solutions. Then, we present the results of the 1D monodomain model using ionic models. To compute the errors, we used two functions, $\|\mathbf{e}\|_2$ and $\|\mathbf{e}\|_\infty$, which are defined by Eq. (2.12) and Eq. (2.13) in Appendix B.

4.B.1 Test Problems for Coupled Systems in One Dimension

(a) Varying dx for $dt = 10^{-4}$

dx	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
1/8	9.623×10^{-2}	-
1/16	2.846×10^{-2}	3.381
1/32	8.270×10^{-3}	3.441

(b) Varying dt for $dx = 10^{-3}$

dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dt]}{\|\mathbf{e}\|_2[dt/2]}$
1/8	1.719	-
1/16	6.875×10^{-1}	2.500
1/32	3.079×10^{-1}	2.232

Table 4.4: The results of test problem 4 using OS1 and IMP2 for solving the ODE part.

Test Problem 4: The test problem consists of the following system of equations,

$$
\begin{aligned}
u_t &= u_{xx} + (2t-4)(\frac{v}{5})^{1/3} \qquad (4.6)\\
v_t &= 30\,tu^3 \\
u(x,0) &= e^{2x}, \qquad 0 \le x \le 1 \\
u_x(1,t) &= 2e^{2+t^2}, \quad 0 \le t \le 1 \\
u_x(0,t) &= 2e^{t^2}, \qquad 0 \le t \le 1,
\end{aligned}
$$

with an exact solution of $u(x,t) = e^{2x+t^2}$, $v(x,t) = 5(e^{2x+t^2})^3$. The results of applying OS1, OS2, and semi-implicit to the test problem are shown in Tables 4.4 to 4.6 respectively. Figure 4.3 also shows the plots of $\|\mathbf{e}\|_2$ over dt for test problem 4. The results show that the order of the outputs follow what we expect in theory. It confirms the correctness of our implementation of the PDE model.

(a) Varying dx for $dt = 10^{-3}$

dx	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
1/8	9.700×10^{-2}	-
1/16	2.918×10^{-2}	3.324
1/32	9.260×10^{-3}	3.151

(b) Varying dt for $dx = 10^{-3}$

dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dt]}{\|\mathbf{e}\|_2[dt/2]}$
1/4	1.027	-
1/8	3.441×10^{-1}	2.984
1/16	1.059×10^{-1}	3.249

Table 4.5: The results of test problem 4 using OS2 and IMP2 for solving the ODE part.

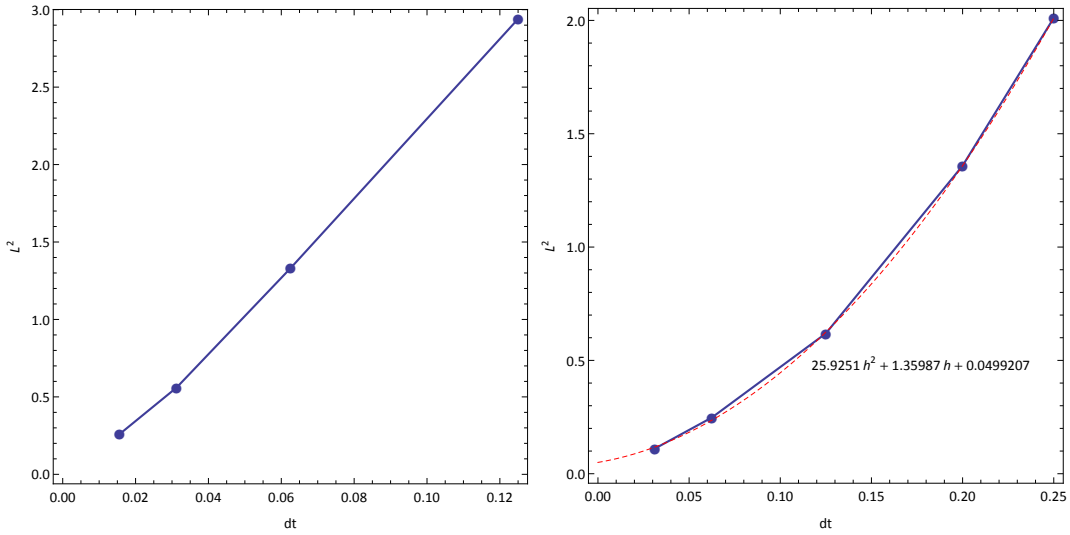
(a) Varying dx for $dt = 10^{-4}$

dx	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
1/8	9.630×10^{-2}	-
1/16	2.845×10^{-2}	3.384
1/32	8.310×10^{-3}	3.423

(b) Varying dt for $dx = 10^{-3}$

dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dt]}{\|\mathbf{e}\|_2[dt/2]}$
1/8	1.719	-
1/16	6.873×10^{-1}	2.500
1/32	3.083×10^{-1}	2.229

Table 4.6: The results of test problem 4 using Semi-implicit method.



(a) Result of applying OS1

(b) Result of applying OS2. Dashed curve shows the least-square error fit of a polynomial of degree 2.

Figure 4.3: Plots of $\|\mathbf{e}\|_2$ over dt for test problem 4.

Test Problem 5: The test problem consists of the following system of equations,

$$
\begin{aligned}
u_t &= u_{xx} + 4\,t(v+1) + u - 1 \qquad (4.7)\\
v_t &= -\frac{1}{2}(u-1)\\
u(x,0) &= 2\sin x + 1, \qquad 0 \le x \le \frac{\pi}{4}\\
u_x(\frac{\pi}{4},t) &= 2\cos(\frac{\pi}{4} + t^2), \quad 0 \le\ t\ \le \frac{\pi}{4}\\
u_x(0,t) &= 2\cos t^2, \qquad 0 \le\ t\ \le \frac{\pi}{4},
\end{aligned}
$$

with an exact solution of $u(x,t) = 2\sin(x+t^2)+1$ and $v(x,t) = \cos(x+t^2)-1$.

(a) Varying dx for $dt = \frac{\pi}{40000}$

dx	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
$\pi/16$	9.642×10^{-3}	-
$\pi/32$	2.644×10^{-3}	3.646
$\pi/64$	7.072×10^{-4}	3.738

(b) Varying dt for $dx = \frac{\pi}{4000}$

dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dt]}{\|\mathbf{e}\|_2[dt/2]}$
$\pi/16$	1.462×10^{-1}	-
$\pi/32$	5.110×10^{-2}	2.861
$\pi/64$	2.080×10^{-2}	2.456

Table 4.7: Results of test problem 5 using OS1 and IMP2 for solving the ODE part.

Tables 4.7 and 4.8 show the results of test problem 5. The results of test problem show that the order of the outputs follow what we expect in theory. It confirms the correctness of our implementation of the PDE model. In addition, OS2 gives more accurate results compared to OS1.

(a) Varying dx for $dt = \frac{\pi}{40000}$

dx	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
$\pi/16$	9.459×10^{-3}	-
$\pi/32$	2.468×10^{-3}	3.832
$\pi/64$	5.849×10^{-4}	4.219

(b) Varying dt for $dx = \frac{\pi}{4000}$

dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dt]}{\|\mathbf{e}\|_2[dt/2]}$
$\pi/16$	4.891×10^{-2}	-
$\pi/32$	1.419×10^{-2}	3.44
$\pi/64$	4.393×10^{-3}	3.23

Table 4.8: Results of test problem 5 using OS2 and IMP2 for solving the ODE part.

4.B.2 Results of One Dimensional Monodomain Coupled with Luo-Rudy I

We have already described the monodomain and Luo-Rudy I model in Sections 2.A.4 and 2.B. In this section, we present the result of our simulations for the one dimensional monodomain model coupled with Luo-Rudy I. Table 4.9 shows the results of monodomain combined with Luo-Rudy I model using different methods. The parameters of the model are the same as the ones described in Section 2.B and the initial condition is assumed to be,

$$u(x,0) = \begin{cases} -84.5 & x \geq 0.3 \\ 20 & x < 0.3 \end{cases} \tag{4.8}$$

The reference solution for the Luo-Rudy I model is generated using IMP2 method with $dt = 10^{-5}$ and $dx = 2 \times 10^{-3}$. To solve the PDE part, backward Euler and Crank-Nicolson methods are used. For the ODE part, the results of

IMP2, ESDIRK3, and SDIRK4 are presented. Please note that for the ODE part, we also tried two explicit methods (namely Forward Euler and ERK4). However, since they don't give a stable output for the time steps of Table 4.9, we don't present the result here.

The results show that IMP2, which is a fully implicit method, gives the most accurate results compared to other methods we tried [1]. Furthermore, if we use Crank-Nicolson instead of backward Euler, we receive better results.

Table 4.10 shows the timing result of backward Euler and CrankNicolson methods for the Aliev-Panfilov model. Simulations are run on an Intel Core Due CPU with 3 gigabytes of RAM. Table 4.11 shows the timing result for the Luo-Rudy model with the same configuration.

4.C Result of Two Dimensional Monodomain Model

The results of two dimensional monodomain model using different ionic models are presented in this section. Both OS1 and OS2 methods are used to solve coupled systems of ODE and PDE in two dimensions. For the ODE part, we tried IMP2, ESDIRK3, SDIRK4, FE1, and ERK4 methods. However, the results with FE1 and ERK4 are not presented as they do not produce a stable solution with a large value of time step. For the PDE part backward Euler is used. The results show that IMP2, which is a fully implicit method, gives the most accurate results compared to other ODE solvers we tried. We first test our programs against an analytical solution. Then, we use complex ionic

[1] Note that we only examined non-adaptive methods in this section.

(a) OS1 with backward Euler

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.05	0.005	6.771×10^{-2}
ESDIRK3	0.05	0.005	7.141×10^{-2}
SDIRK4	0.05	0.005	7.053×10^{-2}

(b) OS2 with backward Euler

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.05	0.005	6.300×10^{-2}
ESDIRK3	0.05	0.005	7.129×10^{-2}
SDIRK4	0.05	0.005	6.970×10^{-2}

(c) OS1 with Crank-Nicolson

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.05	0.005	1.508×10^{-2}
ESDIRK3	0.05	0.005	1.733×10^{-2}
SDIRK4	0.05	0.005	1.697×10^{-2}

(d) OS2 with Crank-Nicolson

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.05	0.005	1.157×10^{-2}
ESDIRK3	0.05	0.005	1.252×10^{-2}
SDIRK4	0.05	0.005	1.244×10^{-2}

Table 4.9: Results of monodomain combined with Luo-Rudy I model using different methods

PDE Solver	dt	dx	Total Time (S)	PDE Time (S)
backward Euler	0.1	0.5	28.41	3.97
CrankNicolson	0.1	0.5	30.84	5.05
backward Euler	0.1	0.25	70.47	22.45
CrankNicolson	0.1	0.25	72.55	23.96

Table 4.10: Comparison of the timing of backward Euler and CrankNicolson methods for the Aliev-Panfilov model.

PDE Solver	dt	dx	Total Time (S)	PDE Time (S)
backward Euler	0.1	0.05	50.41	2.57
CrankNicolson	0.1	0.05	58.84	2.55
backward Euler	0.1	0.025	135.43	17.32
CrankNicolson	0.1	0.025	141.21	17.26

Table 4.11: Comparison of the timing of backward Euler and CrankNicolson methods for the Luo-Rudy model.

models. To compute the errors, we used two functions, $\|\mathbf{e}\|_2$ and $\|\mathbf{e}\|_\infty$, which are defined by Eq. (2.14) and Eq. (2.15) in Appendix B.

4.C.1 Test Problem for Coupled Systems in Two Dimensions

Test Problem 6: The test problem consists of the following system of equations,

$$\begin{cases} u_t &= u_{xx} + u_{yy} - 7(\frac{v}{11})^{1/5} \\ v_t &= 55u^5 \end{cases} \tag{4.9}$$

with the following boundary conditions,

$$\begin{aligned} u(x,y,0) &= e^{2u+2y} \\ u_x(0,y,t) &= 2e^{2y+t} \\ u_x(1,y,t) &= 2e^{2+2y+t} \\ u_y(x,0,t) &= 2e^{2x+t} \\ u_y(x,1,t) &= 2e^{2x+t+2}\,. \end{aligned}$$

The problem has an exact solution of $u(x, y, t) = e^{2x+2y+t}$ and $v(x, y, t) = 11(e^{2x+2y+t})^5$. The results of the test problem for OS1 and OS2 methods are shown in Tables 4.12 and 4.13 respectively, which verifies the correctness of the programs. Additionally, OS2 provides more accurate results than OS1, which is the expected behavior.

(a) Varying $dx = dy$ for $dt = 10^{-5}$

$dx = dy$	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
$1/16$	6.41×10^{-2}	-
$1/32$	1.76×10^{-2}	3.64
$1/64$	4.97×10^{-3}	3.54

(b) Varying dt for $dx = dy = 10^{-2}$

dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
$1/8$	2.93×10^{0}	-
$1/16$	1.51×10^{0}	1.94
$1/32$	7.46×10^{-1}	2.02

Table 4.12: The result of test problem 6 using OS1 method and IMP2 for solving the ODE part.

In Sections 4.C.2 to 4.C.4, we present the result of Aliev-Panfilov, Hodgkin-Huxley, and Luo-Rudy I combined with the monodomain model in two dimensions. The parameters of the model are the same as the ones described in Section 2.B.

4.C.2 Results of 2D monodomain coupled with Aliev-Panfilov

We have already described the monodomain and Aliev-Panfilov model in Sections 2.A.3 and 2.B. In this section we present the result of our simulations

(a) Varying $dx = dy$ for $dt = 10^{-3}$

$dx = dy$	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
1/16	7.030×10^{-2}	-
1/32	1.830×10^{-2}	3.84
1/64	4.710×10^{-3}	3.88

(b) Varying dt for $dx = dy = 10^{-2}$

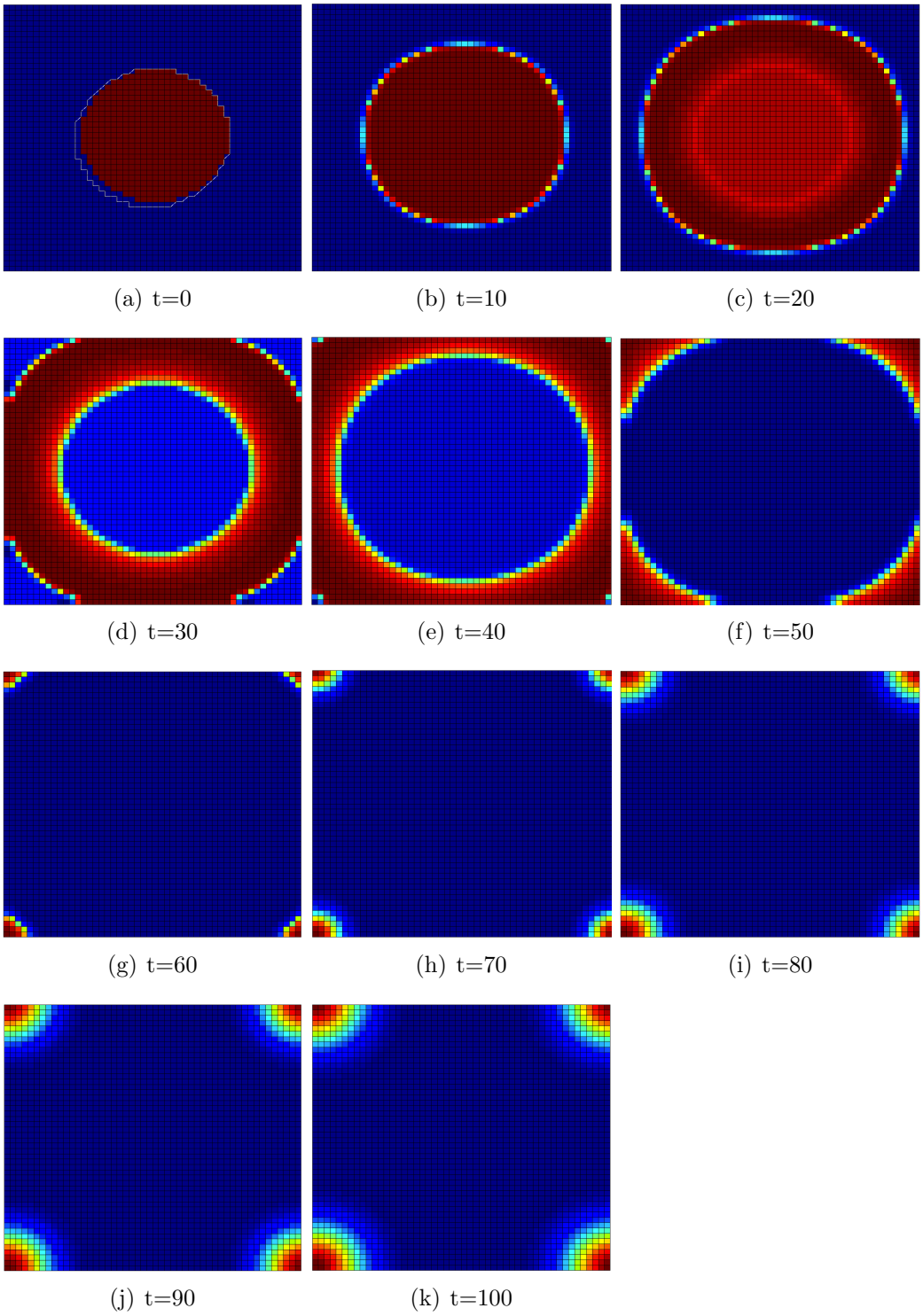
dt	$\|\mathbf{e}\|_2$	$\frac{\|\mathbf{e}\|_2[dx]}{\|\mathbf{e}\|_2[dx/2]}$
1/8	1.7900	-
1/16	4.610-1	3.88
1/32	1.480-1	3.11

Table 4.13: The result of test problem 6 using OS2 method and IMP2 for solving the ODE part.

for the two dimensional monodomain model coupled with Aliev-Panfilov. The initial condition of the simulations of this section is as follows,

$$u(x, y, 0) = \begin{cases} 0 & \sqrt{(x-50)^2 + (y-50)^2} \geq 25 \\ 1 & \sqrt{(x-50)^2 + (y-50)^2} < 25 \end{cases} \tag{4.10}$$

Table 4.14 shows the result of OS1 and OS2 methods applied to the two dimensional monodomain model combined with Aliev-Panfilov model. Compared with ESDIRK3 and SDIRK4, IMP2 gives more accurate results for both OS1 and OS2 methods. Note that, FE1 and ERK4 do not provide a stable output for the given Δt and Δx. Figure 4.4 also shows the evolution of transmembrane potential over time for the 2D monodomain model combined with Aliev-Panfilov. The reference solution is generated using the OS2 method with $dt = 0.00025$ and $dx = 0.001$. The ODE part is solved with IMP2 while the PDE part is solved using the backward Euler method.



(a) t=0 (b) t=10 (c) t=20

(d) t=30 (e) t=40 (f) t=50

(g) t=60 (h) t=70 (i) t=80

(j) t=90 (k) t=100

Figure 4.4: Evolution of transmembrane potential over time for the 2D monodomain model combined with Aliev-Panfilov model.

(a) Results of OS1

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.04	0.05	1.229×10^{-1}
ESDIRK3	0.04	0.05	1.467×10^{-1}
SDIRK4	0.04	0.05	1.342×10^{-1}

(b) Results of OS2

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.04	0.05	1.176×10^{-1}
ESDIRK3	0.04	0.05	1.377×10^{-1}
SDIRK4	0.04	0.05	1.241×10^{-1}

Table 4.14: The result of 2D monodomain combined with Aliev-Panfilov model

4.C.3 Results of 2D monodomain coupled with Hodgkin-Huxley

We have already described the monodomain and Hodgkin-Huxley model in Sections 2.A.1 and 2.B. In this section, we present the result of our simulations for the two dimensional monodomain model coupled with Hodgkin-Huxley. The initial condition of the simulations of this section is as follows,

$$u(x, y, 0) = \begin{cases} -75 & \sqrt{(x-100)^2 + (y-100)^2} \geq 30 \\ 20 & \sqrt{(x-100)^2 + (y-100)^2} < 30 \end{cases} \tag{4.11}$$

Table 4.15 shows the result of OS1 and OS2 methods applied to the two dimensional monodomain model combined with Hodgkin-Huxley model. Compared with ESDIRK3 and SDIRK4, IMP2 gives more accurate results for both OS1 and OS2 methods. As mentioned earlier, FE1 and ERK4 do not provide a stable output for the given Δt and Δx. Figure 4.5 also shows the evolution of transmembrane potential over time for the 2D monodomain model

combined with Hodgkin-Huxley. The reference solution is generated using the OS2 method with $dt = 7 \times 10^{-4}$ and $dx = 10^{-3}$. The ODE part is solved with IMP2 while the PDE part is solved using the backward Euler method.

(a) Results of OS1

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.0175	0.01	8.324×10^{-2}
ESDIRK3	0.0175	0.01	9.154×10^{-2}
SDIRK4	0.0175	0.01	8.872×10^{-2}

(b) Results of OS2

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.0175	0.01	7.915×10^{-2}
ESDIRK3	0.0175	0.01	8.513×10^{-2}
SDIRK4	0.0175	0.01	8.252×10^{-2}

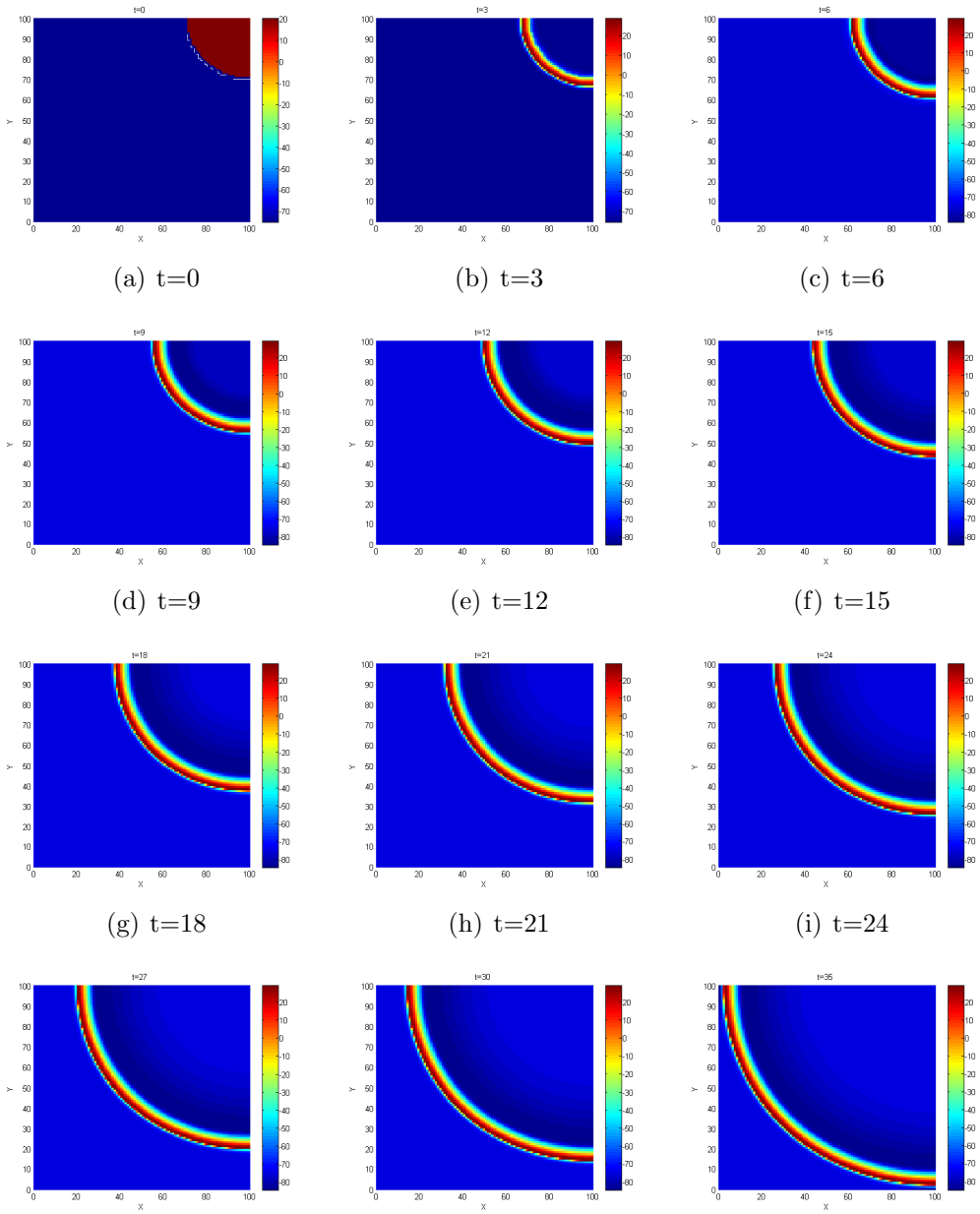
Table 4.15: The result of 2D monodomain combined with Hodgkin-Huxley model

4.C.4 Results of 2D monodomain coupled with Luo-Rudy I

We have already described the monodomain and Luo-Rudy I model in Sections 2.A.4 and 2.B. In this section, we present the result of our simulations for the two dimensional monodomain model coupled with Luo-Rudy I. The initial condition of the simulations of this section is as follows,

$$u(x, y, 0) = \begin{cases} -84 & \sqrt{(x-100)^2 + (y-100)^2} \geq 30 \\ 20 & \sqrt{(x-100)^2 + (y-100)^2} < 30 \end{cases} \tag{4.12}$$

Table 4.16 shows the result of OS1 and OS2 methods applied to the two



(a) t=0 (b) t=3 (c) t=6

(d) t=9 (e) t=12 (f) t=15

(g) t=18 (h) t=21 (i) t=24

(j) t=27 (k) t=30 (l) t=35

Figure 4.5: Evolution of transmembrane potential over time for the 2D monodomain model combined with Hodgkin-Huxley model.

dimensional monodomain model combined with Luo-Rudy I. Compared with ESDIRK3 and SDIRK4, IMP2 gives more accurate results for both OS1 and OS2 methods. As mentioned earlier, FE1 and ERK4 do not provide a stable output for the given Δt and Δx. The reference solution is generated using OS2 with $dt = 0.00125$ and $dx = 0.005$. The ODE part is solved with IMP2 while the PDE part is solved using the backward Euler method.

(a) Results of OS1

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.05	0.01	1.883×10^{-1}
ESDIRK3	0.05	0.01	1.987×10^{-1}
SDIRK4	0.05	0.01	1.932×10^{-1}

(b) Results of OS2

ODE Solver	dt	dx	$\|\mathbf{e}\|_2$
IMP2	0.05	0.01	1.773×10^{-1}
ESDIRK3	0.05	0.01	1.811×10^{-1}
SDIRK4	0.05	0.01	1.802×10^{-1}

Table 4.16: The result of 2D monodomain combined with Luo-Rudy I model

4.D Spiral Waves

In this section, we present the result of applying the semi-implicit method to a special case of FitzHugh-Nagumo model to obtain spiral waves. Spiral waves are believed to initiate the ventricular fibrillation. Ventricular fibrillation is a severely abnormal heart rhythm that is produced by one or many spiral propagation waves of the excitation cardiac wall [Bourgault et al., 2003, Belhamadia, 2008]. Figure 4.6 shows the two point simulation of the spiral waves using monodomain model combined with the modified FitzHugh-Nagumo model with

the following equations,

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla . D \nabla u + c_1 u(u-a)(1-u) - c_2 uv + I_s \\ \frac{\partial v}{\partial t} &= b(u - dv), \end{aligned} \tag{4.13}$$

with Neumann boundary condition $\frac{\partial u}{\partial n} = 0$. In the above system, u is the transmembrane potential, v is the recovery variable, n is a vector normal to the boundary, and I_s is the stimulus current. The parameters are chosen as $a = 0.13$, $b = 0.013$, $c_1 = 0.26$, $c_2 = 0.1$, and $d = 1$ similar to [Rogers and McCulloch, 1994]. Two electrical stimuli are applied to the cell at the beginning of the simulation and at time $570ms$. The amplitude of both of the stimuli are $I_s = 30\mu A/cm^2$. The figure shows that a point stimulus is applied in the center of the domain. Later, it follows by another point stimulus at the center. For more details see about the spiral waves see [Pertsov et al., 1993].

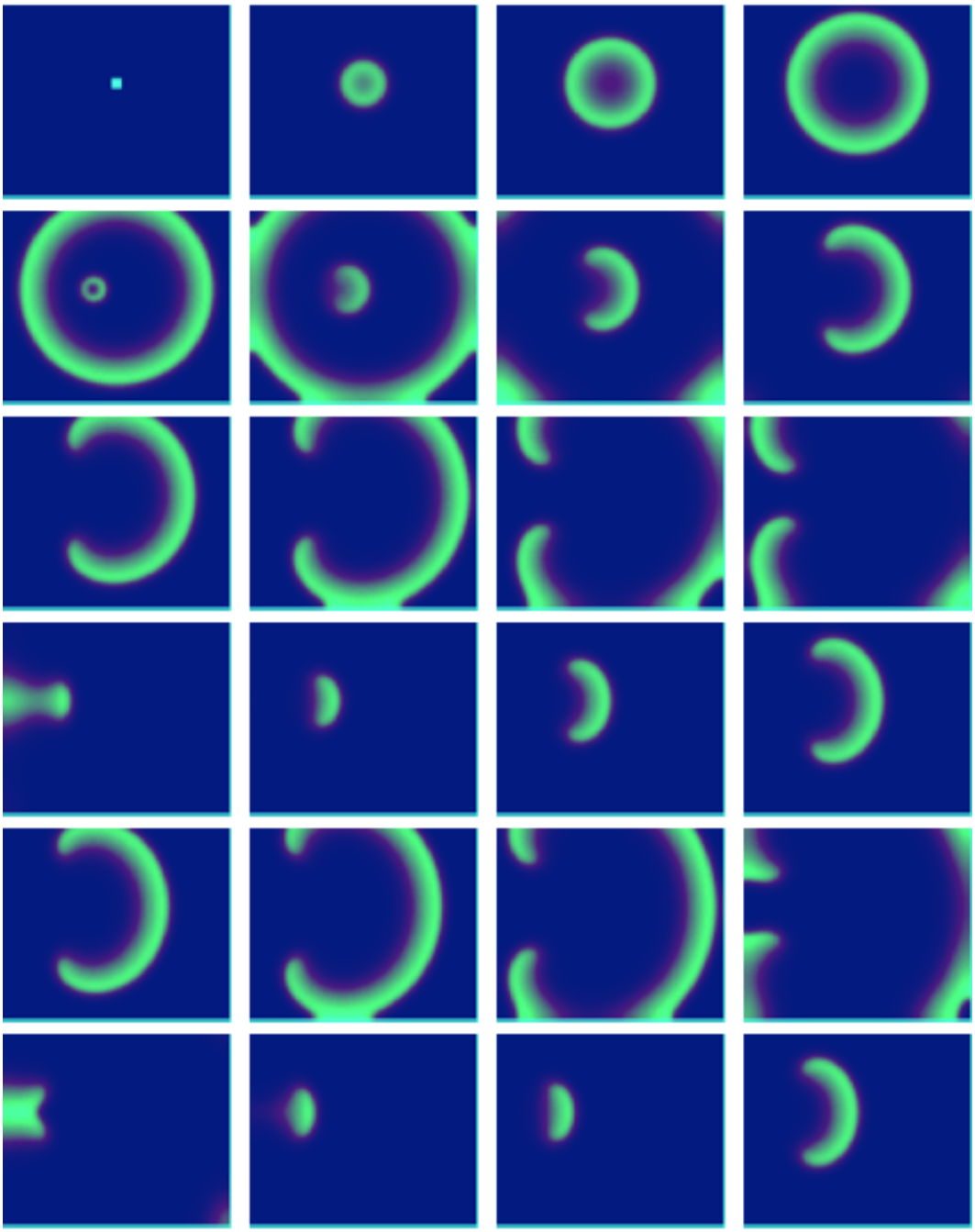


Figure 4.6: Spiral waves for the FitzHugh-Nagumo model for $t = 0$ to 3750ms. The flow is from top to bottom and from left to right.

Chapter 5

General Discussion and Conclusions

Mathematical models of electrophysiology of the heart can generally be grouped into single cell and tissue scale models. A single cell model consist of a system of ODEs. There are many different single cell models with different degrees of loyalty to the real world model and different levels of computation intensiveness. However, it is known that the models that capture more details of the heart ionic activity are more complex and require more computational power to solve.

Tissue scale models, in addition to a set of ODEs, contains one or more PDEs. Two well known tissue scale models, namely monodomain and bidomain are commonly studied in the literature. Nevertheless, because of high computational power required by the bidomain model, the main focus of this thesis was on the monodomain model.

On the one hand, because of complexity and stiffness of the equations, solving the models is a challenging task and requires an enormous amount of computational power. On the other hand, accurate solutions to the electro-

physiological models of the heart can improve the heart disease detection and prevention. Therefore, having efficient solutions for these models translates into more effective and less expensive ways to prevent and cure heart diseases.

In this thesis, we explored the Runge-Kutta family of ODE solvers to find out which of the methods best fit for solving the electrophysiological models of the heart. To effectively employ the ODE and PDE solvers, we used operator splitting methods of order one and two to split a coupled system of equations into the ODE and PDE parts. Then, we examined different ODE and PDE solvers to find out which method works better. As a general rule of thumb, we can say operator splitting method of order two works better than the order one counterpart. Furthermore, implicit ODE solvers outperform the explicit solvers. Among different implicit ODE solvers we tried, IMP2 usually produces the most accurate result. In some cases, explicit ODE solvers are not able to produce a stable output or require very small time steps to be stable. The latter case means explicit methods require a longer time to solve a problem compared to the implicit methods. Since, most of the computation cost is consumed to solve the ODE part, using implicit methods of order 2 results in faster and more accurate solutions. For the PDE part, in the cases that we tried two solvers, Crank-Nicolson performs better than backward Euler.

Bibliography

[Aliev and Panfilov, 1996] Aliev, R. and Panfilov, A. (1996). A simple two-variable model of cardiac excitation. *Chaos, Solitons & Fractals*, 7(3):293–301.

[Belhamadia, 2008] Belhamadia, Y. (2008). A time-dependent adaptive remeshing for electrical waves of the heart. *Biomedical Engineering, IEEE Transactions on*, 55(2):443–452.

[Belhamadia et al., 2009] Belhamadia, Y., Fortin, A., and Bourgault, Y. (2009). Towards accurate numerical method for monodomain models using a realistic heart geometry. *Mathematical biosciences*, 220(2):89–101.

[Belhamadia et al., 2012] Belhamadia, Y., Fortin, A., and Bourgault, Y. (2012). Nested implicit runge-kutta method for simulating cardiac cell model. *International Journal of Bioelectromagnetism*, 14(2):40–45.

[Bourgault et al., 2003] Bourgault, Y., Ethier, M., and LeBlanc, V. (2003). Simulation of electrophysiological waves with an unstructured finite element method. *ESAIM: Mathematical Modelling and Numerical Analysis*, 37(04):649–661.

[Butcher, 2008] Butcher, J. (2008). *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 2nd edition.

[Courtemanche et al., 1998] Courtemanche, M., Ramirez, R., and Nattel, S. (1998). Ionic mechanisms underlying human atrial action potential properties: insights from a mathematical model. *American Journal of Physiology-Heart and Circulatory Physiology*, 275(1):H301–H321.

[DiFrancesco and Noble, 1985] DiFrancesco, D. and Noble, D. (1985). A model of cardiac electrical activity incorporating ionic pumps and concentration changes. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 307(1133):353–398.

[DiMasi et al., 2003] DiMasi, J., Hansen, R., and Grabowski, H. (2003). The price of innovation: new estimates of drug development costs. *Journal of health economics*, 22(2):151–185.

[Hairer et al., 1993] Hairer, E., Nørsett, S., and Wanner, G. (1993). *Solving ordinary differential equations: Nonstiff problems*. Springer.

[Hairer and Wanner, 2004] Hairer, E. and Wanner, G. (2004). *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer; 2nd edition.

[Hodgkin and Huxley, 1952] Hodgkin, A. and Huxley, A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*.

[Iserles, 2008] Iserles, A. (2008). *A first course in the numerical analysis of differential equations*. Cambridge University Press.

[Jacquemet et al., 2003] Jacquemet, V., Virag, N., Ihara, Z., Dang, L., Blanc, O., Zozor, S., VESIN, J., Kappenberger, L., and Henriquez, C. (2003). Study of unipolar electrogram morphology in a computer model of atrial fibrillation. *Journal of cardiovascular electrophysiology*, 14:S172–S179.

[Katz, 2010] Katz, A. (2010). *Physiology of the Heart*. Lippincott Williams & Wilkins.

[Keener and Bogar, 1998] Keener, J. and Bogar, K. (1998). A numerical method for the solution of the bidomain equations in cardiac tissue. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 8(1):234.

[LeVeque, 2007] LeVeque, R. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics.

[Lines et al., 2003] Lines, G., Buist, M., Grottum, P., Pullan, A., Sundnes, J., and Tveito, A. (2003). Mathematical models and numerical methods for the forward problem in cardiac electrophysiology. *Computing and visualization in science*, 5(4):215–239.

[LU, 2011] LU, Y. Y. (2011). *Numerical Methods for Differential Equations*. Department of Mathematics, City University of Hong Kong. Avaiable at `http://math.cityu.edu.hk/~mayylu/ma3514/3514.pdf`.

[Luo and Rudy, 1991] Luo, C. and Rudy, Y. (1991). A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. *Circulation research*, 68(6):1501–1526.

[Luo and Rudy, 1994] Luo, C. and Rudy, Y. (1994). A dynamic model of the cardiac ventricular action potential. i. simulations of ionic currents and concentration changes. *Circulation research*, 74(6):1071–1096.

[MacLachlan et al., 2007] MacLachlan, M., Sundnes, J., and Spiteri, R. (2007). A comparison of non-standard solvers for odes describing cellular reactions in the heart. *Computer Methods in Biomechanics and Biomedical Engineering*, 10(5):317–326.

[Noble, 1962] Noble, D. (1962). A modification of the hodgkinhuxley equations applicable to purkinje fibre action and pacemaker potentials. *The Journal of physiology*, 160(2):317–352.

[Pertsov et al., 1993] Pertsov, A., Davidenko, J., Salomonsz, R., Baxter, W., and Jalife, J. (1993). Spiral waves of excitation underlie reentrant activity in isolated cardiac muscle. *Circulation research*, 72(3):631–650.

[Qu and Garfinkel, 1999] Qu, Z. and Garfinkel, A. (1999). An advanced algorithm for solving partial differential equation in cardiac conduction. *Biomedical Engineering, IEEE Transactions on*, 46(9):1166–1168.

[Qu et al., 2000] Qu, Z., Weiss, J., and Garfinkel, A. (2000). From local to global spatiotemporal chaos in a cardiac tissue model. *Physical Review E*, 61(1):727.

[Rogers and McCulloch, 1994] Rogers, J. and McCulloch, A. (1994). A collocation-galerkin finite element model of cardiac action potential propagation. *Biomedical Engineering, IEEE Transactions on*, 41(8):743–757.

[Roth, 1995] Roth, B. (1995). A mathematical model of make and break electrical stimulation of cardiac tissue by a unipolar anode or cathode. *Biomedical Engineering, IEEE Transactions on*, 42(12):1174–1184.

[Rush and Larsen, 1978] Rush, S. and Larsen, H. (1978). A practical algorithm for solving dynamic membrane equations. *Biomedical Engineering, IEEE Transactions on*, (4):389–392.

[Saleheen and Ng, 1998] Saleheen, H. and Ng, K. (1998). A new three-dimensional finite-difference bidomain formulation for inhomogeneous anisotropic cardiac tissues. *Biomedical Engineering, IEEE Transactions on*, 45(1):15–25.

[Santos et al., 2005] Santos, R., Plank, G., Bauer, S., and Vigmond, E. (2005). Preconditioning techniques for the bidomain equations. *Domain Decomposition Methods in Science and Engineering*, pages 571–580.

[Shuaiby et al., 2011] Shuaiby, S., Hassan, M., and El-Melegy, M. (2011). Modeling and simulation of the action potential in human cardiac tissues using finite element method. *Journal of Communications and Computer Engineering*, 2(3):21–27.

[Spiteri and Dean, 2008] Spiteri, R. and Dean, R. (2008). On the Performance of an Implicit–Explicit Runge–Kutta Method in Models of Cardiac Electrical Activity. *Biomedical Engineering, IEEE Transactions on*, 55(5):1488–1495.

[Spiteri and MacLachlan, 2003] Spiteri, R. and MacLachlan, M. (2003). An efficient non-standard finite difference scheme for an ionic model of cardiac action potentials. *Journal of Difference Equations and Applications*, 9(12):1069–1081.

[Sundnes, 2006] Sundnes, J., e. a. (2006). *Computing the electrical activity in the heart*, volume 1. Springer Verlag.

[Sundnes et al., 2009] Sundnes, J., Artebrant, R., Skavhaug, O., and Tveito, A. (2009). A second-order algorithm for solving dynamic cell membrane equations. *Biomedical Engineering, IEEE Transactions on*, 56(10):2546–2548.

[Sundnes et al., 2001] Sundnes, J., Lines, G., and Tveito, A. (2001). Efficient solution of ordinary differential equations modeling electrical activity in cardiac cells. *Mathematical biosciences*, 172(2):55–72.

[Sundnes et al., 2005] Sundnes, J., Lines, G., and Tveito, A. (2005). An operator splitting method for solving the bidomain equations coupled to a volume conductor model for the torso. *Mathematical biosciences*, 194(2):233–248.

[Ten Tusscher et al., 2004] Ten Tusscher, K., Noble, D., Noble, P., and Panfilov, A. (2004). A model for human ventricular tissue. *American Journal of Physiology-Heart and Circulatory Physiology*, 286(4):H1573–H1589.

[Van Zuijlen and Bijl, 2005] Van Zuijlen, A. and Bijl, H. (2005). Implicit and explicit higher order time integration schemes for structural dynamics and fluid-structure interaction computations. *Computers & structures*, 83(2):93–105.

[Winslow et al., 1999] Winslow, R., Rice, J., Jafri, S., Marban, E., O'Rourke, B., et al. (1999). Mechanisms of altered excitation-contraction coupling in canine tachycardia-induced heart failure, ii: model studies. *Circulation research*, 84(5):571.

Appendices

Appendix A

Cell Models

1.A Luo-Rudy I Model

1.B Source Code of the Models

```
#ifndef LUORUDY_H_INCLUDED
#define LUORUDY_H_INCLUDED

#include "XArray.h"
#include <math.h>

XArray<double> luo_rudy_model(double t, XArray<double>
   y)
{
    const double Kplusi = 145;
    const double Kpluse = 5.4;
    const double Naplusi = 18;
    const double Napluse = 140;
    const double gNa = 23;
    const double gKp = 1.83 * 0.01;
    const double gb = 3.921 * 0.01;
    const double Cm = 1;
    const double Eb = -59.87;
    const double R = 8314;
    const double T = 310;
    const double F = 96484.6;
    const double PR = 1.833 * 0.01;
    const double ENa = ((R * T) / (F)) * log(Napluse /
```

```
        Naplusi);

    double alphax = (5 * (0.0001) * ((exp(0.083 * (y
      (+0) + 50))) / (exp(0.057 * (y(+0) + 50)) + 1)))
      ;
    double betax = (1.3 * (0.001)) * ((exp(-0.06 * (y
      (+0) + 20))) / (exp(-0.04 * (y(+0) + 20)) + 1));
    double alpham = (0.32 * (y(+0) + 47.13)) / (1 -
      exp(-0.1 * (y(+0) + 47.13)));
    double betam = 0.08 * exp(-y(+0) / 11);
    double alphad = (0.095) * ((exp(-0.01 * (y(+0) -
      5))) / (exp(-0.072 * (y(+0) - 5)) + 1));
    double betad = (0.07 * ((exp(-0.017 * (y(+0) + 44)
      )))) / (exp(0.05 * (y(+0) + 44)) + 1);
    double alphaf = 0.012 * ((exp(-0.008 * (y(+0) +
      28))) / (exp(0.15 * (y(+0) + 28)) + 1));
    double betaf = 0.0065 * ((exp(-0.02 * (y(+0) + 30)
      )) / (exp(-0.2 * (y(+0) + 30)) + 1));

    double alphah = 0;
    double betah = 0;
    double alphaj = 0;
    double betaj = 0;

    if(y(+0) >= -40)
    {
        alphah = 0;
        betah = 1 / (0.13 * (1 + exp(-(y(+0) + 10.66)
          / 11.1)));
        alphaj = 0;
        betaj = (0.3 * exp(-2.535 * (1.0 / 10000000) *
           y(+0))) / (1 + exp(-0.1 * (y(+0) + 32)));
    }
    else
    {
        alphah = 0.135 * exp((-80 - y(+0)) / 6.8);
        betah = (3.56 * exp(0.079 * y(+0))) + (3.1 *
          (100000) * exp(0.35 * y(+0)));
        alphaj = (((-1.2714 * (100000) * exp(0.2444 *
          y(+0))) - ((3.474 * (1.0 / 100000) *
                    exp(-0.04391 * y(+0))))) * (y(+0)
                      + 37.78)) / (1 + exp(0.311 * (y
```

```
                    (+0) + 79.23)));
51          betaj = (0.1212 * exp(-0.01052 * y(+0))) / (1
              + exp(-0.1378 * (y(+0) + 40.14)));
52      }
53
54      double Esi = 7.7 - (13.0287 * log(y(+7)));
55      double Isi = 0.09 * y(+4) * y(+5) * (y(+0) - Esi);
56      double gK1 = 0.6047 * sqrt(Kpluse / 5.4);
57      double EK1 = (R * T / F) * log(Kpluse / Kplusi);
58      double EK = (R * T / F) * log(((Kpluse) + (PR *
          Napluse)) / (Kplusi + (PR * Naplusi)));
59
60      double Xi = 0;
61
62      if(y(+0) <= -100)
63      {
64          Xi = 1;
65      }
66      else
67      {
68          Xi = (2.837) * (exp(0.04 * (y(+0) + 77)) - 1)
              / ((y(+0) + 77) * exp(0.04 * (y(+0) + 35)));
69      }
70
71      double gK = 0.282 * sqrt(Kpluse / 5.4);
72      double IK = gK * y(+6) * Xi * (y(+0) - EK);
73      double gamma = exp(0.06175 * (y(+0) - EK1 -
          594.31));
74      double alphaK1 = 1.02 / (1 + exp(0.2385 * (y(+0) -
          EK1 - 59.215)));
75      double betaK1 = ((0.49124 * exp(0.08032 * (y(+0) -
          EK1 + 5.476))) + gamma) /
76                              (1 + exp(-0.5143 * (y(+0)
                                  - EK1 + 4.753)));
77      double K1infinity = alphaK1 / (alphaK1 + betaK1);
78      double IK1 = gK1 * K1infinity * (y(+0) - EK1);
79      double EKp = EK1;
80      double Kp = 1 / (1 + exp((7.488 - y(+0)) / 5.98));
81      double IKp = gKp * Kp * (y(+0) - EKp);
82      double Ib = gb * (y(+0) - Eb);
83      double INa = gNa * pow(y(+1), 3) * y(+2) * y(+3) *
          (y(+0) - ENa);
```

```
84      double Iion = INa + Isi + IK + IK1 + IKp + Ib;
85
86      XArray<double> df(8);
87
88      df(0) = -(Iion) / Cm;
89      df(1) = (alpham * (1 - y(+1))) - (betam * y(+1));
90      df(2) = (alphah * (1 - y(+2))) - (betah * y(+2));
91      df(3) = (alphaj * (1 - y(+3))) - (betaj * y(+3));
92      df(4) = (alphad * (1 - y(+4))) - (betad * y(+4));
93      df(5) = (alphaf * (1 - y(+5))) - (betaf * y(+5));
94      df(6) = (alphax * (1 - y(+6))) - (betax * y(+6));
95      df(7) = (-0.0001 * Isi) + (0.07 * (0.0001 - y(+7))
          );
96
97      return df;
98  }
99
100 double luo_rudy_initu(double x)
101 {
102     return (x >= 0.3) ? -84.5 : 20;
103 }
104
105 XArray<double> luo_rudy_initv(double x)
106 {
107     XArray<double> r(7);
108
109     if(x >= 0.3)
110     {
111         r(0) = 0.00167;
112         r(1) = 0.928;
113         r(2) = 1;
114         r(3) = 0.00298;
115         r(4) = 1;
116         r(5) = 0.00602;
117         r(6) = 0.000178;
118     }
119     else
120     {
121         r(0) = 0.0001;
122         r(1) = 0.0001;
123         r(2) = 0.0001;
124         r(3) = 0.0001;
```

```
125         r(4) = 0.0001;
126         r(5) = 0.0001;
127         r(6) = 0.0001;
128     }
129 
130     return r;
131 }
132 
133 #endif // LUORUDY_H_INCLUDED
134 
135 /*********************************************/
136 
137 XArray<double> aliev_panfilov_AD_v1(double t, XArray<
      double> y)
138 {
139     const double k = 8;
140     const double a = 0.15;
141     const double u1 = 0.2;
142     const double u2 = 0.3;
143     const double epsilon0 = 0.002;
144     int dims[] = {2};
145     XArray<double> df(1, dims);
146 
147     double epsilon = epsilon0 + ((u1 * y(1)) / (y(0) +
          u2));
148     df(0) = -(k * y(0) * (y(0) - a) * (y(0) - 1)) - (y
          (0) * y(1));
149     df(1) = epsilon * ((-y(1)) - (k * y(0) * (y(0) - a
          - 1)));
150     return df;
151 }
152 
153 /*********************************************/
154 
155 XArray<double> fitzhugh_nagumo_AD_v1(double t, XArray<
      double> y)
156 {
157     double c1 = 1;
158     double a = 0.25;
159     double c2 = 1;
160     double b = 0.0016875;
161     double c3 = 0.01;
```

```
162     XArray<double> df = y;
163     df(0) = (c1 * y(0) * (y(0) - a) * (1 - y(0))) - (
          c2 * y(1));
164     df(1) = ((b * y(0)) - (c3 * y(1)));
165     return df;
166 }
167
168 /*****************************************/
169
170 #ifndef XARRAY_H_INCLUDED
171 #define XARRAY_H_INCLUDED
172
173 #include <string>
174 #include <sstream>
175 #include <vector>
176 #include <assert.h>
177
178 using namespace std;
179
180 template <class T = double>
181 class XArray
182 {
183     // Fields (keep data)
184     int index_helper[10];
185     // cells of the array
186     vector<T> table;
187     // dimensions of the array
188     vector<int> dims;
189
190 public:
191     XArray(){}
192
193     XArray(unsigned int n, int *d)
194     {
195         dims.resize(n);
196         int size = 1;
197         for (unsigned int i = 0; i < n; i++) {
198             size *= d[i];
199             dims[i] = d[i];
200         }
201         table.resize(size);
202     }
```

```
203
204         XArray(unsigned int d1)
205     {
206         dims.resize(1);
207         dims[0] = d1;
208         table.resize(d1);
209     }
210
211         XArray(unsigned int d1, unsigned int d2)
212     {
213         dims.resize(2);
214         dims[0] = d1;
215         dims[1] = d2;
216         table.resize(d1 * d2);
217     }
218
219         XArray(unsigned int d1, unsigned int d2,
              unsigned int d3)
220     {
221         dims.resize(3);
222         dims[0] = d1;
223         dims[1] = d2;
224         dims[2] = d3;
225         table.resize(d1 * d2 * d3);
226     }
227
228     XArray(const XArray<T>& xa)
229     {
230         this->table = xa.table;
231         this->dims = xa.dims;
232     }
233
234         int dim(int i)
235     {
236         return dims[i];
237     }
238
239         int num_dims()
240         {
241             return dims.size();
242         }
243
```

```
244     T& operator()(int i)
245     {
246         index_helper[0] = i;
247         return get_helper(1, index_helper);
248     }
249
250     T& operator()(int i, int j)
251     {
252         index_helper[0] = i;
253         index_helper[1] = j;
254         return get_helper(2, index_helper);
255     }
256
257     T& operator()(int i, int j, int k)
258     {
259         index_helper[0] = i;
260         index_helper[1] = j;
261         index_helper[2] = k;
262         return get_helper(3, index_helper);
263     }
264
265     XArray<T> operator*(double m)
266     {
267         XArray<T> r = *this;
268         for (unsigned int i = 0; i < table.size(); i
              ++) {
269             r.table[i] *= m;
270         }
271
272         return r;
273     }
274
275     XArray<T> operator+(const XArray<T> &that)
276     {
277         assert(this->dims.size() == that.dims.size());
278         for (unsigned int i = 0; i < dims.size(); i++)
               {
279             assert(this->dims[i] == that.dims[i]);
280         }
281
282         XArray<T> r = *this;
283         for (unsigned int i = 0; i < table.size(); i
```

```
            ++) {
284             r.table[i] += that.table[i];
285         }
286
287         return r;
288     }
289
290         private:
291     T& get_helper(unsigned int n, int *indices)
292     {
293         assert(n == dims.size());
294
295         int multiplier = 1;
296         int index = 0;
297
298         for (unsigned int i = 0; i < n; i++) {
299             //cerr << "index " << i << " out of range.
                  Expected [0, " << dims[i] - 1
300             //     << "] found " << indices[i] << endl
                  ;
301             assert(indices[i] >= 0 && indices[i] <
                  dims[i]);
302             index += indices[i] * multiplier;
303             multiplier *= dims[i];
304         }
305
306         return table[index];
307     }
308 };
309
310 template <class T>
311 ostream &operator<<(ostream &stream, XArray<T> xa)
312 {
313     int d = xa.num_dims();
314
315     if(d == 1)
316     {
317         int n = xa.dim(0);
318         for(int i = 0; i < n; i++)
319         {
320             stream << xa(i);
321             if(i < n - 1)
```

```
322                 {
323                     stream << ", ";
324                 }
325             }
326         }
327         else
328         {
329             stream << "XArray[";
330             for(int i = 0; i < d; i++)
331             {
332                 stream << xa.dim(i);
333                 if(i < d - 1)
334                 {
335                     stream << "x";
336                 }
337             }
338             stream << "]";
339         }
340 
341         return stream;
342 }
343 
344 #endif // XARRAY_H_INCLUDED
```

Appendix B

Definitions

2.A Computation of Continuous and Discrete Norms

Consider a problem in which the solution is a function $u(x)$ over the interval $a \leq x \leq b$. Assume a method approximate the solution by $\hat{u}(x)$. The error is given by,

$$e(x) = \hat{u}(x) - u(x) \tag{2.1}$$

The magnitude of the error can be measured using the appropriate p-norm,

$$\|e\|_p = \left(\int_a^b |e(x)|^p dx \right)^{1/p} \tag{2.2}$$

For two dimensions the p-norm is defined analogously as,

$$\|e\|_p = \left(\int_a^b \int_c^d |e(x,y)|^p dx dy \right)^{1/p} \tag{2.3}$$

for $a \leq x \leq b$ and $c \leq y \leq d$.

Some methods such as finite difference methods, instead of producing a function as the solution, produce a set of values U_i at grid points $x_i = a + ih$ for $i = 0, \ldots, N$ (*i.e.*, $U_i \approx u(x_i)$). Let the vector of errors $\mathbf{e} = (e_1, \ldots, e_N)$ be defined by,

$$e_i = U_i - u(x_i) \tag{2.4}$$

The usual vector norms for e_i would grow unboundedly with the increase of the number of grid points and therefore the following norms for grid functions are used,

$$\|\mathbf{e}\|_1 = h \sum_{i=0}^{N} |e_i| \tag{2.5}$$

with h as the scaling factor. Note that the scaling factor h scales the sum by $1/N$ as the number of point increases. Without the scaling factor h, the error grows unboundedly when $N \to \infty$. Therefore, $\|\mathbf{e}\|_i$ is the average value of $\mathbf{e}$ over the interval. Similarly, the p-norm $\|\mathbf{e}\|_p$ is defined as,

$$\|\mathbf{e}\|_p = \left(h \sum_{i=0}^{N} |e_i|^p \right)^{\frac{1}{p}} \tag{2.6}$$

Note that Eq. (2.6) is the discretization of Eq. (2.2). For $p = 2$ we have,

$$\|\mathbf{e}\|_2 = \sqrt{h \sum_{i=0}^{N} |e_i|^2} \tag{2.7}$$

While $h^{1/p} \to 1$ as $p \to \infty$, the ∞-norm is defined as,

$$\|\mathbf{e}\|_\infty = \max_{1 \le i \le N} |e_i| \tag{2.8}$$

In two dimensions, the p-norm is analogously defined as,

$$\|\mathbf{e}\|_p = \left(\Delta x \Delta y \sum_i \sum_j |e_{ij}|^p \right)^{\frac{1}{p}} \tag{2.9}$$

with the special cases of,

$$\|\mathbf{e}\|_2 = \sqrt{\Delta x \Delta y \sum_i \sum_j |e_{ij}|^2} \tag{2.10}$$

and

$$\|\mathbf{e}\|_\infty = \max_{\substack{1 \le i \le N \\ 1 \le j \le M}} |e_{ij}| \tag{2.11}$$

In the above equations we have $x_i = a + i\Delta x$ for $i = 0, \ldots, N$, $y_i = c + j\Delta y$ for $j = 0, \ldots, M$, and $e_{ij} = U_{ij} - u(x_i, y_j)$. For higher dimensions the p-norm is defined analogously.

Please note that since we use finite difference methods in this thesis, we use the error definitions for the grid functions. More specifically, we use the 2-norm and ∞-norm for the models throughout the thesis. For the models with one dimension in space we use the following error functions,

$$\|\mathbf{e}\|_2 = \sqrt{\Delta t \Delta x \sum_i \sum_j |e_{ij}|^2} \tag{2.12}$$

and

$$\|\mathbf{e}\|_\infty = \max_{\substack{1 \le i \le N \\ 1 \le j \le M}} |e_{ij}| \tag{2.13}$$

where x and t denote the variables in space and time and M and N are defined as above. While for two dimensional models we use,

$$\|\mathbf{e}\|_2 = \sqrt{\Delta t \Delta x \Delta y \sum_i \sum_j \sum_k |e_{ijk}|^2} \tag{2.14}$$

and

$$\|\mathbf{e}\|_\infty = \max_{\substack{1 \le i \le N_1 \\ 1 \le j \le N_2 \\ 1 \le k \le M}} |e_{ijk}| \tag{2.15}$$

where x and y denote the variables in space and t denotes the time. It is assumed that N_1 and N_2 are the number of grid points along x and y directions.