Some Stability Issues Of The Stationary Bidomain Model

Master Thesis

Zhiguang Jia

July 31, 2008
Acknowledgments

I am indebted to a great number of people who generously offered advise, encouragement, inspiration, and friendship during my master study. First and foremost I would like to thanks my supervisors Dr. Bjørn Fredrik Nielsen and Mr. Tomas Syrstad Ruud for their valuable time, guidance and feedback on both the technical and the theoretical aspects of this thesis. Thanks to my brother Dr. Zhiyang Jia and my sister in-law Weizhen Zhu, without their encouragement and support, I could not possibly have the opportunity to continue my study here at University of Oslo. I am also grateful to my friends who made me laugh in the tough time of writing the thesis. Finally I want to thank my parents, because of them I came to this world and become a person who I am today.
Contents

1 Introduction .......................................................... 1
   1.1 Motivation .................................................... 1
   1.2 Problem Description ........................................ 2
   1.3 Contributions ................................................ 2

2 Background .......................................................... 4
   2.1 Heart Electrical Activities ............................... 4
   2.2 The Bidomain Model ........................................ 6
      2.2.1 The Torso Model .................................... 7
      2.2.2 The Heart Model ................................... 9
      2.2.3 The Ionic Current Model ............................ 12
      2.2.4 The Complete Bidomain Model ...................... 12
   2.3 The Computational Model ................................ 13
   2.4 The Stationary Bidomain model .......................... 14
   2.5 Diffpack and The Simulator ................................ 15

3 Experiment Design .................................................. 18
   3.1 Selection of Heart Characteristics for Experiment .......... 19
   3.2 Variation Ranges ............................................. 20
      3.2.1 Range of Variation In Heart Fiber Orientation .... 20
      3.2.2 Range of Variation in Heart Conductivity .......... 21
      3.2.3 Range of Variation in Heart Geometry .............. 22
   3.3 Benchmark .................................................... 25
      3.3.1 Specification of $\sigma_i$ and $\sigma_e$ .................. 26
      3.3.2 Specification of Heart fiber orientations .......... 26
      3.3.3 Specification of Heart Geometry ..................... 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.4</td>
<td>Specification of Ischemia Region</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>A Measure of Relative Difference of The Body Surface Potentials</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>Experiment Setups</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Experiment Procedure</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>Software Tool Design</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Requirements</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Programming Paradigm</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Architecture Pattern</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Classes</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Inheritance</td>
<td>42</td>
</tr>
<tr>
<td>4.6</td>
<td>Workflow</td>
<td>43</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Main Workflow</td>
<td>43</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Experiment Workflow</td>
<td>45</td>
</tr>
<tr>
<td>4.7</td>
<td>Dataflow</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>Software Tool Implementation</td>
<td>52</td>
</tr>
<tr>
<td>5.1</td>
<td>Choice of the Programming Language: Python, Ruby and Perl</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Visualization Tools</td>
<td>54</td>
</tr>
<tr>
<td>5.3</td>
<td>GUI Toolkit</td>
<td>55</td>
</tr>
<tr>
<td>5.4</td>
<td>Documentation Tool</td>
<td>55</td>
</tr>
<tr>
<td>5.5</td>
<td>Implementation Details</td>
<td>56</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Affine Transformation</td>
<td>56</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Header File</td>
<td>58</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Visualizations on The GUI</td>
<td>59</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Simulation Result Extraction</td>
<td>61</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Preparation for Simulation</td>
<td>62</td>
</tr>
<tr>
<td>5.5.6</td>
<td>Automatic Load of Perturber</td>
<td>62</td>
</tr>
<tr>
<td>5.5.7</td>
<td>Abstract Super Classes</td>
<td>63</td>
</tr>
<tr>
<td>5.5.8</td>
<td>Experiment Exportation and Importation</td>
<td>64</td>
</tr>
<tr>
<td>5.5.9</td>
<td>Graphical Exceptions</td>
<td>64</td>
</tr>
<tr>
<td>5.5.10</td>
<td>From 2D to 3D</td>
<td>65</td>
</tr>
<tr>
<td>5.5.11</td>
<td>File Organization</td>
<td>66</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

The bidomain model is a mathematical model which has been used to study the heart electrical activity for many years. However, its complexity sometimes limits its usage in some scientific research areas. To deal with this problem, the stationary bidomain model is derived from the bidomain model as an alternative by utilizing the known value of the resting transmembrane potential. The stationary model has been used by some scientists for different research purposes, for example the researches done in and . Through a series of experiments MacLachlan et al concluded in that subendocardial ischemia is indeed locatable by ST segment shift above the ischemic region on both the epicardial and torso surface. Later Nielsen et al concluded in paper that the position and size of the ischemia region can be roughly recovered from both the pure observation data(resting potential on the body surface) and the observation data with Gaussian noise added. In both articles the experiments are conducted based on a specific set of physiology and anatomy characteristics. However, the inter-individual variation in those characteristics is not taken into account. How the stationary model behaves with respect to those variations is critical, especially to the inverse problem addressed in . This thesis aims to shed some lights on this problem.
1.2 Problem Description

This thesis focuses on two basic tasks:

1. Investigate the stability (or sensitivity) of the stationary model with respect to the inter-individual variations in the physiology and anatomy characteristics to a certain extent, and try to find out what aspects of these characteristics affect the model’s behavior most. The investigation is done through a series of experiments.

2. Develop a software tool to help to perform the investigation. The tool should be able to serve as a simple framework for possible future investigations of the similar kind.

In order to accomplish the investigation, some of the heart physiology and anatomy characteristics have to be selected together with their variation range to be put in the experiments. The selection of values within these ranges is also an important issue as only a finite set of values within these ranges can be set in the experiments while in reality the heart characteristics could take any value within the variation ranges. Developing a software tool is the other task of this thesis. Considering possible future investigations of the similar kind, this tool must be carefully designed and implemented.

1.3 Contributions

In this thesis I investigate the stability of the stationary bidomain model with respect to the inter-individual variations of the physiological and anatomical characteristics to a certain extent. A program with graphic user interface has been developed. With the help from this program, a series of experiments have been conducted. The variation ranges of the heart characteristics are set according to the relevant informations reported in the literature. I find that:

1. The variations in the heart conductivities, the fiber orientations, and the heart position and orientation only affects the values of the body surface potentials. It is also observed that neither the variation in the
heart position nor the variation in the orientation affects the potential
distribution on the heart surface. It shows that the stationary model is
stable with respect to the inter-individual variation of the physiological
and anatomical characteristics experimented. Such stability indicates
that the body surface potential variations due to the inter-individual
characteristic variations are predictable. However, the experiment re-
sults also show that the variation in the position of the ischemia region
affects the body surface potentials most. It not only affects the values
of the potentials, but the distribution pattern of the potential values.

2. A software tool with a graphic user interface is successfully developed.
The experiments are conducted and analyzed automatically. The vi-
sualization of the physiology and anatomy characteristics, the analysis
result, and the simulation result are displayed on the graphical user
interface. The tool can be extended to support the experiments based
on the heart characteristics not involved in this thesis without chang-
ing any existing code. With the help of the good structure and well
documented APIs, it is easy to add new components or customize, even
replace any current component of the program. This feature makes it
easy to extend to support investigations of the similar kind.
Chapter 2

Background

2.1 Heart Electrical Activities

The main functionality of the heart is to pump blood periodically to the body and the lungs. The heart has four chambers: left atrium, right atrium, left ventricle and right ventricle. The atria are located on the top of the heart; the ventricles are located on the bottom of the heart. In each period the impulse created by atrionector results in contraction of the atria which pump the blood into the ventricles, then the pulse passes to the ventricles through atrio-ventricular node, resulting in ventricles’s contraction to pump the blood into the body and the lungs, see Figure 2.1 for the impulse conduction and 2.2 for the blood circulation in the heart. Both figures are from [12].

The contraction is triggered by the action potential: an special pulse-lide wave of voltage. When the action potential value achieves a certain threshold, the heart muscle cells respond with their contractions. The action potential is normally initiated by capacitive currents which are generated when a wave of depolarization approaches a region of the resting area of the heart [30, Chap 13]. When the cells are at rest, the potential inside cells are negative, comparing to the potential outside cells. When depolarizing, the inside cell potential will increase to be positive or approximately zero because of the influx of positive charged current, the resulting potential then activates the cells. When repolarizing the intra-cellular potential will be restored to it’s resting level. As the body is a conductor, the changes in the potential can be
2.1. HEART ELECTRICAL ACTIVITIES

Figure 2.1: Conduction of Impulse in the Heart

(a) Blood enters both atria (b) When the ventricles are (c) Delivering the blood to simultaneously from the full, the contract simultaneously the pulmonary and systemic pulmonary and systemic and pulmonaryously. circuits. When full, the atria pump their blood into the ventricles.

Figure 2.2: Blood Circulation in the Heart
detected on the body surface. This makes it possible to record the electrical activity of the heart through measuring the potential values on the body surface. Electrocardiogram (ECG) is one example of such application.

### 2.2 The Bidomain Model

As one of the most important organs, the heart has been continuously studied for over two thousand years since the days of Aristotle\[63\]. Although the study has given a huge amount of knowledge about the cardiac processes happening on cellular or sub-cellular level, few is known on how these processes interact to form the basic heart function because of the complexity of the heart. To cope with this problem, the mathematical modeling was introduced into the cardiac research field. The mathematical modeling is the process in which the scientists try to express the reality with the help of the mathematical model. In essence, the mathematical model is the mathematical representation of the reality involving some degree of approximations\[10\, Chap 1.2.2\]. The approximations can be achieved by idealizing and simplifying the original problem\[29\], while the representation is formulated by combining the known observations, the fundamental natural laws and mathematical techniques. Scientists expect that the mathematical model can reveal the critical properties of the reality it represents.

The bidomain model first introduced by Tung \[68\] is one of the mathematical models of the heart electrical activities. It consists of a group of time and space dependent ordinary differential equations (ODEs) and partial differential equations (PDEs). The primary unknowns of this model are the potentials in the extra cellular domain and the transmembrane potentials. As shown in Fig 2.3 the domain of body is divided into 2 parts: the heart part and the torso part. Together with the surrounding medium they form two boundaries:

1. The interface between the torso and the surrounding medium, denoted by $\partial T$.
2. The interface between the heart and the surrounding torso, denoted by $\partial H$. 
2.2. THE BIDOMAIN MODEL

Figure 2.3: The sketch over domains of the heart H and the torso T. \( \partial T \) denotes the interface between body and it’s surrounding medium; \( \partial H \) denotes the interface between the heart and the torso.

Because of its importance, the ionic current is also involved in the model. The complete bidomain model is then formed by coupling the torso electrical activity model, the heart electrical activity model and the ionic current model. Both the heart and the torso are modeled as volume conductor, the potential at each point in both domain is viewed as a quantity average over a small volume around the point. The derivation of the complete bidomain model in \([63]\) is sketched in following sections.

2.2.1 The Torso Model

The Maxwell’s equations for volume conductor give the relation between electric field \( E \) and magnetic field \( B \):

\[
\nabla \times E + \frac{\partial B}{\partial t} = 0
\]

(2.1)

For each moment the fields can be treated as being *quasi-static*. This assumption gives rise of equation\([62]\):

\[
E = -\nabla u
\]

(2.2)
in which \( u \) denotes the electric field potential. If denote conductivity of the medium by \( M \), the current \( J \) is given by

\[
J = ME
\]

Inserting (2.2) gives:

\[
J = -M \nabla u \tag{2.3}
\]

By assuming no sinks and sources in the medium and no electric charges build up in any point, the net current leaving a small volume \( S \) must be zero. It gives:

\[- \int_V \nabla \cdot J \, dV = 0 \tag{2.4}\]

It must hold for any arbitrary volume, implies

\[
\nabla \cdot J = 0 \tag{2.5}
\]

Inserting (2.3) gives

\[
\nabla \cdot (M \nabla u) = 0 \tag{2.6}
\]

Naturally medium surrounding the human body can be assumed to be electric insulating, leading to that the normal component of current is zero on the body surface, giving rise of equation

\[
n \cdot J = 0
\]

By inserting (2.3) it gives

\[
n \cdot \nabla u = 0 \tag{2.7}
\]

On the interface between the heart and the torso a natural condition is that the electric potential on the interface is equivalent to the electric potential on the heart surface, which is known for the torso model as the source of electrical activity is located in the heart. The model of electrical activity in the torso can be formulated as:

\[
\nabla \cdot (M_T \nabla u_T) = 0, \quad x \in T \tag{2.8}
\]

\[
n \cdot M_T \nabla u_T = 0, \quad x \in \partial T \tag{2.9}
\]

\[
u_T = u_{\partial H}, \quad x \in \partial H \tag{2.10}
\]
2.2. THE BIDOMAIN MODEL

where $T$ denotes the body domain; $H$ denotes the heart domain; $u_{\partial H}$ denotes the electric potential on the heart surface and $u_{\partial T}$ denotes the potential on the body surface.

2.2.2 The Heart Model

The heart is divided into two separate domains: the intra-cellular domain and the extra-cellular domain. The intra-cellular domain and the extra-cellular domain are separated by the cell membrane. The extracellular domain is naturally continuous, the intra-cellular domain is also continuous as the neighboring cells are connected through gap junctions embedded on the membrane. Moreover there are two assumptions involved in modeling process:

1. The intra-cellular domain is completely isolated by cell membrane

2. Both domains are assumed to be quasi-static.

Because of the assumption of quasi-static condition the relation between current $J_i$ and $J_e$ and potential $u_i, u_e$ are ohmic

\begin{align}
J_i &= - M_i \nabla u_i \quad (2.11) \\
J_e &= - M_e \nabla u_e \quad (2.12)
\end{align}

where $M_i$ is the conductivity in the intra-cellular domain; $M_e$ is the conductivity in the extracellular domain. Unless there are extraneous sources of charge the total charge must be conserved. On the cell membrane there might be some charged ions accumulation occurred in each domain. In [63] the argument for conservation is that “because of the small thickness of the membrane, and accumulating of charge on one side of the membrane immediately attracts an opposite charge on the other side of the membrane.” There could be other arguments based on the biological property of cells:

1. Proactive transportation of ions [31, 2.5].

2. Electrolytes balancing mechanism.
This conservation states that the variation of the total accumulation over time \( t \) at any point is zero, which gives:

\[
\frac{\partial}{\partial t}(q_i + q_e) = 0
\]

where \( q_i \) is the charge in the intra-cellular domain; and \( q_e \) is the charge in the extracellular domain. By rearranging terms it gives:

\[
\frac{\partial q_i}{\partial t} = -\frac{\partial q_e}{\partial t}
\]  

(2.13)

The net current at a point in each domain is equal to the sum of the rate of the charge accumulation at this point and the ionic current leaving this point. This gives

\[
-\nabla \cdot J_i = \frac{\partial q_i}{\partial t} + \chi I_{ion}
\]  

(2.14)

\[
-\nabla \cdot J_e = \frac{\partial q_e}{\partial t} - \chi I_{ion}
\]  

(2.15)

where \( I_{ion} \) is the ionic current across the membrane measured per unit area of cell membrane, the constant \( \chi \) represents the area of cell membrane per unit volume, thus \( \chi I_{ion} \) represents the ionic current per unit volume. The positive direction of ionic current is set to be pointing from intra-cellular domain to the extra-cellular domain. Thus combining (2.11), (2.12), (2.13), (2.14) and (2.15) it follows:

\[
\nabla \cdot (M_i \nabla u_i) + \nabla \cdot (M_e \nabla u_e) = 0
\]

(2.16)

The transmembrane potential \( v \) defined as \( v = u_i - u_e \) can be expressed as

\[
v = \frac{q}{\chi C_m}
\]  

(2.17)

where

\[
q = \frac{q_i - q_e}{2}
\]  

(2.18)

and \( C_m \) is the capacitance of the cell membrane. By taking the time derivative of (2.17) and inserting (2.18) it gives:

\[
\chi C_m \frac{\partial v}{\partial t} = \frac{1}{2} \frac{\partial (q_i - q_e)}{\partial t}
\]  

(2.19)
2.2. THE BIDOMAIN MODEL

Combining (2.18) and (2.13) gives
\[ \frac{\partial q_i}{\partial t} = -\frac{\partial q_e}{\partial t} = \chi C_m \frac{\partial v}{\partial t} \] (2.20)

Inserting this into (2.14) gives
\[ -\nabla \cdot J_i = \chi C_m \frac{\partial v}{\partial t} + \chi I_{ion} \] (2.21)

Inserting (2.11) gives
\[ \nabla \cdot (M_i \nabla u_i) = \chi C_m \frac{\partial v}{\partial t} + \chi I_{ion} \] (2.22)

By replacing \( u_i \) with \( u_e + v \), the variable \( u_i \) can be eliminated from the equation (2.16) and equation (2.22). A rearrangement of terms gives
\[ \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) = \chi C_m \frac{\partial v}{\partial t} + \chi I_{ion} \] (2.23)
\[ \nabla \cdot (M_e \nabla u_e) = \nabla \cdot (M_t \nabla u_t) \] (2.24)

The assumption of isolated domains states that only the extra-cellular domain has direct contact with the torso. This fact indicates that the normal component of the current in the intra-cellular domain on the heart surface is zero, thus the normal component of the current flowing from the heart surface into the torso must be the normal component of the extra-cellular domain current flowing on the heart surface only. Those can be expressed
\[ n \cdot (M_i \nabla u_i) = 0 \]
\[ n \cdot (M_e \nabla u_e) = n \cdot (M_t \nabla u_t) \]

Again by replacing \( u_i \) with \( u_e + v \), it gives:
\[ n \cdot (M_i \nabla v + M_i \nabla u_e) = 0 \] (2.25)
\[ n \cdot (M_e \nabla u_e) = n \cdot (M_t \nabla u_t) \] (2.26)

Another boundary condition which can also be derived from assumption of isolated domains is that the potential on the boundary of the heart and the torso must be continuous, which change the (2.10) to
\[ u_T = u_e \quad x \in \partial H \] (2.27)
2.2.3 The Ionic Current Model

The ionic current plays the crucial role in the heart functionality. There have been lot of models proposed for the cell’s electrophysiology\cite{63, Chap 2.4}. In general the state \( s \) of the ionic current over time \( t \) is the function of the ionic current itself, the transmembrane potential \( v \) and the time \( t \), it can be expressed as

\[
\frac{\partial s}{\partial t} = F(s, v, t)
\]  

(2.28)

This expression can be seen as the general form of the models proposed so far, further details about the ionic current model refer to \cite{63} and references therein.

2.2.4 The Complete Bidomain Model

By coupling the torso model, the heart model and the ionic current model it gives the complete bidomain model as:

\[
\frac{\partial s}{\partial t} = F(s, v, t), \quad x \in H
\]  

(2.29)

\[
\nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) = \frac{\partial v}{\partial t} + I_{\text{ion}}, \quad x \in H
\]  

(2.30)

\[
\nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u_e) = 0, \quad x \in H
\]  

(2.31)

\[
\nabla \cdot (M_T \nabla u_T) = 0, \quad x \in T
\]  

(2.32)

\[
u_e = u_T, \quad x \in \partial H
\]  

(2.33)

\[
n \cdot (M_i \nabla v + (M_i + M_e) \nabla u_e) = n \cdot (M_T \nabla u_T), \quad x \in \partial H
\]  

(2.34)

\[
n \cdot (M_i \nabla v + M_i \nabla u_e) = 0, \quad x \in \partial H
\]  

(2.35)

\[
n \cdot M_T \nabla u_T = 0, \quad x \in \partial T
\]  

(2.36)

Equation (2.29) describes the electro-physiological behavior of heart cells; the equations from (2.30) to (2.31) describe the signal propagation in the heart tissue; equation (2.32) describes potential distribution on the heart surface; equation (2.36) specifies the boundary conditions on the body surface and the connection between the heart and the surrounding body is described by the equations from (2.33) to (2.35).
2.3 THE COMPUTATIONAL MODEL

2.3 The Computational Model

The bidomain model derived in section 2.2.1 consists of a system of ODEs and PDEs. To solve this system two approaches are employed in [63]:

1. Operator splitting

2. Finite element method

Those two approaches are combined together to give the rise of the computational model. Through the operator splitting the whole bidomain model is solved in three steps:

1. Solving the system

\[
\frac{\partial v}{\partial t} = -I_{\text{ion}}(v, s), \quad v(t_n) = v^n
\]

\[
\frac{\partial s}{\partial t} = f(v, s), \quad s(t_n) = s^n
\]

for \(t_n < t \leq t_n + \theta \Delta t\), where \(s^n = s(t_n)\) and \(v^n = v(t_n)\) are known. The solutions of \(v\) at \(t = t_n + \theta \Delta t\) is denoted by \(v^n\), and the solutions of \(s\) at \(t = t_n + \theta \Delta t\) is denoted by \(s^n\).

2. Solving the PDE system

\[
\nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) = \frac{\partial v}{\partial t}, \quad \text{in H} \quad (2.39)
\]

\[
\nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u_e) = 0, \quad \text{in H} \quad (2.40)
\]

\[
\nabla \cdot (M_T \nabla u_T) = 0, \quad \text{in T} \quad (2.41)
\]

where \(v(t_n) = v^n\), for \(t_n < t < t_n + \Delta t\) with the boundary conditions from 2.33 to 2.36. The resulting solution \(v(t_n + \Delta t)\) is denoted by \(v^{n+1}\).

3. Solving system

\[
\frac{\partial v}{\partial t} = -I_{\text{ion}}(v, s), \quad v(t_n + \theta \Delta t) = v^{n+1}_\theta
\]

\[
\frac{\partial s}{\partial t} = f(v, s, t), \quad s(t_n + \theta \Delta t) = s^{n}_\theta
\]

for \(t_n + \theta \Delta t < t < t_n + \Delta t\). The solutions are \(v^{n+1}\) and \(s^{n+1}\) at \(t = t_n + \Delta t\).
Chapter 2. Background

Step 1 and step 2 can be discretized and solved with ODE solvers, the detailed information refers to [63, Chap 5]. The system derived from the operator spiting in the step 2 is a system of PDEs which needs more treatment. For such problem the choice of numerical methods is large but only finite element method is focused. The FEM is based on the H-space theory, it focus on finding solutions for PDEs in a space specified by some prerequests, see [64]. For PDEs derived from step 2 the equivalent problem in terms of mathematics can be formed as following:

Find $v^{n+1} \in V(H)$ and $u^{n+\theta} \in V(H \cup T)$ such that

$$
(v_0^{n+1}, \eta) + \theta \Delta t a_T(v_0^{n+1}, \eta) + \Delta t a_I(u^{n+\theta}, \eta) = (v_0^n, \eta) - (1 - \theta) \Delta t a_I(v^n, \eta)
$$

for all $\eta \in V(H)$

$$
\Delta t a_I(v^{n+1}, \zeta) + \frac{\Delta t}{\theta} a_{I+E}(u^{n+\theta}, \zeta) + \frac{\Delta t}{\theta} a_T(u^{n+\theta}, \zeta) = \frac{1 - \theta}{\theta} a_I(v_0^n, \zeta)
$$

for all $\zeta \in V(H \cup T)$

(2.44)

(2.45)

where

$$
(\psi, \phi) = \int_H \psi \phi dx, \quad \text{for } \psi, \phi \in V(H),
$$

(2.46)

$$
a_I(\psi, \phi) = \int_H M_i \nabla \psi \cdot \nabla \phi dx, \quad \text{for } \psi, \phi \in V(H),
$$

(2.47)

$$
a_{I+E}(\psi, \phi) = \int_H (M_i + M_e) \nabla \psi \cdot \nabla \phi dx, \quad \text{for } \psi, \phi \in V(H),
$$

(2.48)

$$
a_T(\psi, \phi) = \int_T M_T \nabla \psi \cdot \nabla \phi dx, \quad \text{for } \psi, \phi \in V(T).
$$

(2.49)

2.4 The Stationary Bidomain model

One application of the bidomain model is to simulate the changes of the electrical potentials on the body surface caused by various changes in the electrical activity of the heart. Another application is to diagnose the heart
problem through the electrical potentials on the body surface, such application is normally referred as Inverse problem. Because of the complexity of the complete bidomain model, the so called stationary bidomain model (or simply stationary model) is often employed as the alternative to the bidomain model, such as in the articles \[40\] and \[45\]. It takes advantage of known physiology facts to simplify the bidomain model into the stationary model. Following a brief presentation of the stationary model is given. The detailed information refers to \[40\] \[45\]. During resting time a lab measurement of transmembrane potential \(v\) on position \(x\) at time \(t_1\) is \[45\]:

\[
v_1(x) = v(x,t_1) \approx \begin{cases} 
-60\text{mV} & \text{x in ischemic tissue}, \\
-90\text{mV} & \text{x in healthy tissue}, 
\end{cases}
\] (2.50)

Consequently it reformulates equation (2.31) to :

\[
\nabla \cdot ((M_i + M_e)\nabla u_e(x,t_1)) = -\nabla \cdot (M_i \nabla v(x,t_1)) \text{ in } H
\] (2.51)

As at resting stage no ionic current flows between both domains the equation (2.29) is ignored. The stationary model consists of (2.51), (2.32), (2.33), (2.34), (2.35), and (2.36). By applying a series mathematical techniques on the this model the weak formulation is formulated as \[45\]:

\[
\int_B \nabla \gamma \cdot (M \nabla u) dx = - \int_H \nabla \gamma \cdot (M_i \nabla v_1) dx \text{ for all } \gamma \in V(B)
\] (2.52)

where

\[
V(B) = \left\{ \gamma \in H^1(B); \int_B \gamma dx = 0 \right\}
\]

\[
M(x) = \begin{cases} 
M_i(x) + M_e(x) & \text{for } x \in H \\
M_T(x) & \text{for } x \in T
\end{cases}
\]

\[
u(x) = \begin{cases} 
u_e(x,t_1) & \text{for } x \in H \\
u_T(x,t_1) & \text{for } x \in T
\end{cases}
\] (2.53)

### 2.5 Diffpack and The Simulator

Traditionally the scientific computing task is mainly accomplished by utilizing the software written in Fortran or C because of their great execution
speed. Typically, developing software in Fortran or C is a tedious and painful task as lots of low-level programmings are involved in the development process. Although there are some high-level languages such as Matlab which can also be used, the execution speed of the program written in such languages is normally not fast enough. Diffpack \cite{32} is a programming tool kit which allows scientists and engineers working on different fields to efficiently develop the numeric software possessing great execution speed without involving much low level programming. It provides developers with sophisticated libraries as the blocks for building the numerical softwares. Those libraries are focusing on the basic data structures and routines needed by the numerical softwares. The developers can just focus on the main aspects of the problem such as modeling, algorithm and numerical experimentations. It leads to much less programming time than traditional Fortran and C. As this tool kit is developed in object-oriented style the PDE solver can be highly customized. It is very useful when dealing with certain highly specialized problems. Diffpack’s cross platform ability makes the program portable between the Unix-like platforms and the Win32 platforms. Furthermore, with some help from SWIG\cite{6} or other tools, the diffpack routines or programes can be integrated into the code written in script language such that some high level operations can be done in scripting code, and the heavy computational part can be done in C++ code. A previous master student Magne Westlie at University of Oslo had already made some progress towards making the interface between Python and Diffpack in his thesis\cite{71}. The detailed information about Diffpack can be found in \cite{32} and the references therein.

For both the bidomain model and the stationary model, simulators based on Diffpack are available. In this thesis all the experiments are conducted by utilizing the stationary model simulator. The simulator mainly consists of two parts:

1. The part which is responsible for preparing the anatomic and physiological characteristics of the heart and the torso.

2. The part which utilizes the FEM solver to do the actual computing task.

The input for the simulator includes the conductivity of the healthy tissue
and the infarctic tissue, the orientation of the heart fiber, the geometry in-
formation of the heart and the torso, etc. The output includes the potential
in the heart and the torso among others. Those data are stored in several
files. The potentials on the body surface are the only output data interested
in this thesis. The diffpack provides the data filter which can convert the
diffpack output data into other format ready for visualization. The visualiza-
tion tools can be selected in some third party tools such as Matlab, Gnuplot,
and plotmtv, etc.
Chapter 3

Experiment Design

The stability of the stationary bidomain model will be examined through a series of experiments. The main idea is to set up a simulation result as the benchmark, then compare the benchmark with the results from the simulations based on the varied heart physiology and anatomy characteristics. There are mainly five problems which should be solved at the experiment designing stage:

1. Select heart characteristics which will be experimented.

2. Set the variation range of those characteristics and the values which are put in the experiments.

3. Specify the value of the heart characteristics which will be set in the simulation to get the benchmark for simulation results.

4. Define the measure of the difference of the simulation results.

5. Set up the experiments.

6. Set up the experiment procedure.
3.1 Selection of Heart Characteristics for Experiment

A complete description of the heart must include both physiology and anatomy characteristics. For the stationary bidomain model, the description includes but not limited to:

1. The heart conductivity.

2. The heart fiber orientation.

3. The heart geometry, including the heart position and the heart orientation.

The conductivity appears in the model directly. It is straightforward to realize that the conductivity variation would have impact on the solutions of the model. Another important characteristics is the heart fiber orientation. Because the heart conductivity is anisotropic, it is expressed with respect to the heart fiber orientations. The fiber orientation variation at a point would also changes the conductivity at this point. There are several studies that look at the relation between the heart fiber orientations and the heart electrical activities, see [54, 44]. In addition, some studies focus on the relation between the heart geometry and the electrical activities of the heart, such as in paper [14, 24, 41] and references therein. In [41], it claims that the geometric position of the heart appears to be a large source of the variation in body surface potentials. Further research done in [14] confirmed the claim in [41]; in [24], it claims that a significant error in the inverse solution was found when an small error was made in the location of the heart. Particularly in the inverse problem such as determining the position and size of the ischemia region from the observed potentials, the knowledge about the influence of the heart geometry to the body surface potentials is crucial. Because of their importance, the heart conductivity, the heart fiber orientation and the heart geometry are selected to be experimented.
3.2 Variation Ranges

As mentioned in section 3.1 heart conductivity, heart fiber orientation and heart geometry are selected as the subjects for experiment. In reality the values of those are varying from individual to individual. To make the experiments practical, the variation ranges are set according to the values reported in relevant literature.

3.2.1 Range of Variation In Heart Fiber Orientation

A precise knowledge about the fiber architecture can greatly help scientists to understand the electro-physiological phenomena of the heart. Many have tried to construct the model of the fiber architecture, such as in [26, 46, 58, 67]. In a recent paper [70], the intra-individual variation of the fiber orientation is found to be between $-20^\circ$ and $20^\circ$ measured in the angle formed by the epicardial surface and the longitudinal myocyte orientation. Unfortunately to the best of authors knowledge, no paper about the inter-individual variation of the fiber orientations is found. An alternative is to use the relevant knowledge of the canine heart in the experiments. It is known from the classic studies on the cardiac muscle fiber architecture that the fiber architecture in the human and canine heart are very similar, especially the fiber orientation field in the dog heart has been studied in much more detail [65].

In a research report [52], the fiber orientation variations of seven canine hearts were investigated. The observation is that the orientations of the fibers are stable among a population with mean standard deviation of $8.8^\circ$ around the secondary eigenvector and $9.4^\circ$ around the third eigenvector. The principle eigenvector (or primary eigenvector) indicates the fiber direction, the secondary eigenvector indicates the direction which is along the fiber sheet but perpendicular to the fiber direction; the third eigenvector is the normal of the plane defined by the primary and the secondary eigenvector. The detailed explanation refers to [36, 28] and references therein.

With only the standard deviation it’s difficult to derive the variation range. Even with the information of the distribution pattern provided, the variation range derived from it does not give much confidence because the sample size is too small: only 7 canine hearts are investigated.
3.2. VARIATION RANGES

<table>
<thead>
<tr>
<th>i</th>
<th>intra cellular domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>extracellular domain</td>
</tr>
<tr>
<td>L</td>
<td>direction along the fiber</td>
</tr>
<tr>
<td>T</td>
<td>direction perpendicular to L on fiber sheet plane</td>
</tr>
<tr>
<td>n</td>
<td>direction perpendicular to the fiber sheet plane</td>
</tr>
</tbody>
</table>

Table 3.1: The conductivity notations

However, the variation of fiber orientation can be easily approximated according to statistic rules if the variations in the heart fiber orientations are normally distributed. The value variations of some vital signs, such as blood pressure\[9, 50\], heart beat rate \[20\] and body temperature\[11\], can be roughly described with the normal distribution. It may be a hint that the inter-individual variations of the fiber orientations can also be described with the normal distribution.

Denote the standard deviation by $\sigma$ and the mean by $\mu$ and assume the normal distribution, the $68 - 95 - 99.7$ rule states that approximately $47.5\%$ of observations fall within the $[\mu - 2\sigma, \mu]$; another $47.5\%$ of observations fall within the $[\mu, \mu + 2\sigma]$. By taking $9.4^\circ$ as the standard deviation and taking the reference fiber orientations as the mean, the variation range would be $[-18.8^\circ, 18.8^\circ]$.

Obviously such assumption is theoretically lack of hard foundation. But under the condition of lacking necessary information, it is the only way to get the meaningful approximation of the real variation range for the fiber orientations.

3.2.2 Range of Variation in Heart Conductivity

The heart conductivities are expressed with respect to the heart fiber orientations, for convenience the values of those conductivities are denoted here in this thesis by: $\sigma_i^L$, $\sigma_i^T$, $\sigma_i^n$, $\sigma_e^L$, $\sigma_e^n$ and $\sigma_e^T$. See Table 3.1 for the explanation of subscript and superscript.

Different values of the heart conductivities have been proposed in relevant literature, see Table 3.5. Only the values for 2D case are listed in Table 3.5.
CHAPTER 3. EXPERIMENT DESIGN

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma^L_t$</td>
<td>1.7</td>
<td>3.4</td>
<td>3.0</td>
<td>2.8</td>
</tr>
<tr>
<td>$\sigma^T_t$</td>
<td>0.2</td>
<td>0.6</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>$\sigma^L_c$</td>
<td>6.2</td>
<td>1.2</td>
<td>2.0</td>
<td>3.9</td>
</tr>
<tr>
<td>$\sigma^T_c$</td>
<td>2.4</td>
<td>0.8</td>
<td>1.4</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Table 3.2: The proposed conductivity values

<table>
<thead>
<tr>
<th>Conductivity (ms/cm)</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma^L_t$</td>
<td>1.7</td>
<td>3.4</td>
</tr>
<tr>
<td>$\sigma^T_t$</td>
<td>0.2</td>
<td>0.6</td>
</tr>
<tr>
<td>$\sigma^L_c$</td>
<td>1.2</td>
<td>6.2</td>
</tr>
<tr>
<td>$\sigma^T_c$</td>
<td>0.8</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 3.3: The conductivity variation range

as they are part of the main concerns in this thesis.

The value variation range in each conductivity listed in Table 3.5 is selected to span over the range from the minimum to the maximum of all the proposed values for this conductivity, see Table 3.2.

3.2.3 Range of Variation in Heart Geometry

Because only 2D case is considered in this thesis, the heart is represented by the surface formed by cutting transversely through the ventricles, see Fig 3.4. Two aspects of the heart geometry are concerned here: the heart orientation and the heart position.

Range of Variation in Heart Orientation

The heart orientation in [24] is measured with respect to angle between the long axis of the heart and the front plane, see Figure 3.1 from [24, Fig 3] for images showing long axis. The angle varies with standard deviation in ±5°.
Figure 3.1: Long-axis images of six subjects selected from the 25 subjects available. The long axis is indicated by a white line.
As argued in Section 3.2.1, \([-10^\circ, 10^\circ]\) is selected as the variation range.

**Range of Variation in Heart Position**

The position of the heart is also another important issue. In article [16], it claims that left offsets of the heart center with respect to the torso center is approximately \(3 - 6\, \text{cm}\). In experiments done in [41] the variation range of heart position is set as listed in Table 3.4. It can be seen that the shift range in \(x\) axis in Table 3.4 covers the range calculated from [16].

In Table 3.4 the \(x\) axis (Transverse axis) is defined pointing from left to right, \(y\) axis (Medial axis) is pointing from front to back, \(z\) axis (longitudinal axis) is pointing from up to down, see Fig 3.2.

In [41] the source of those values listed in Table 3.4 is referred to [23]. As to the best of author’s knowledge this paper is not available and no other sources of data are found, the values in Table 3.4 from [41] are selected to be experimented. It would also benefit from the selection as the selection makes it possible to compare the experiment result with the one found in [41].
3.3 BENCHMARK

Discretization of Variation Ranges

Mathematically the variation range for each type of heart characteristics is a interval, namely an infinite set of possible values. A finite subset must be selected. In other words, the range must be discretized. There are two ways to discretize the range:

1. Constructing a finite arithmetic sequence covering this range, this method is used in all the experiments done in this thesis.

2. Employing pseudo random number generating algorithm, for example Mersenne twister algorithm\cite{42} to randomly extract a sequence of values from the variation range.

As to the arithmetic sequence, the difference between any two successive members must be specified. It is defined as the minimum value which can be represented by the position of the least significant number of the values specifying the variation range. For instance the least significant number for 5.6 is 0.6, the unit value at this position is 0.1, thus the difference for the arithmetic sequence is 0.1.

The advantage of employing pseudo random number generating algorithm is that the finite arithmetic sequence only covers some values within the whole variation range, still the behaviors of the model with respect to other values remain non-investigated. It is expected that pseudo random number generating algorithm can generate the values not appeared in the arithmetic sequence.

3.3 Benchmark

To compare the change in the behaviors of stationary model due to the variations in the heart characteristics, the benchmark for the behaviours must be established. This benchmark is set to be the result from the simulation with a specific set of heart physiology and anatomy characteristics. The rest of this section is focusing on the specification of such characteristics.
CHAPTER 3. EXPERIMENT DESIGN

<table>
<thead>
<tr>
<th>Conductivity (ms/cm)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_i^L$</td>
<td>1.7</td>
</tr>
<tr>
<td>$\sigma_i^T$</td>
<td>0.2</td>
</tr>
<tr>
<td>$\sigma_e^L$</td>
<td>1.2</td>
</tr>
<tr>
<td>$\sigma_e^T$</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 3.5: The values of the conductivities used in the simulation for the benchmark

3.3.1 Specification of $\sigma_i$ and $\sigma_e$

Both extra-cellular conductivity and intra-cellular conductivity are set to take the values list in Table 3.3.1 They are actually the minimum values listed in Table 3.2.1

3.3.2 Specification of Heart fiber orientations

The heart fiber orientation is specified by a matrix representing the vectors which indicate the cardiac fiber orientation and the origin of the coordinate system used to define those vectors. For each point in the heart the actual cardiac fiber orientation at this point is calculated according to the relation between current point and relevant data in this matrix. The values of the heart fiber orientations are shown below:

11.2329  13.8264  -1.3524  0.4135
12.6414  11.1776   0.8829  0.4695
15.2902  12.586    0.8829  0.4695
15.7037  13.9384  -0.4695  0.8829
13.8818  15.2348  -1.3524  0.4135
11.1769  16.0618  -0.8829  -0.4695
 9.4671  12.8875  -0.4135  -1.3524
 9.0536  11.5351   0.4695  -0.8829
10.406   11.1216   1.3524  -0.4135
12.5854  13.4129  -0.4695   0.8829
3.3. BENCHMARK

Figure 3.3: The heart fiber orientations used in the simulation for the benchmark.

12.1719 12.0605 -0.4695 0.8829

Each row of the matrix represents a vector and the origin of the coordinate system, the last 2 numbers indicate the vector, the others define the origin of the coordinate system. Fig 3.3 shows the actual fiber orientation the matrix represents.

3.3.3 Specification of Heart Geometry

The visualization of the heart and torso geometry is shown in Fig 3.4. For actual representation, see Appendix A.

3.3.4 Specification of Ischemia Region

Subendocardial ischemia is incorporated in the simulation. The ischemia region is created according to two parameters: the region center and the diameter of the region. The region center is located at 9.5, 13.66, the diameter of the area is 1.1cm, see Fig 3.5 for the actual ischemia region.
Figure 3.4: The heart geometry used in the simulation for the benchmark.

Figure 3.5: The ischemia region used in the simulation for the benchmark, the dark red area indicates the ischemia region.
3.4 A Measure of Relative Difference of The Body Surface Potentials

To measure the relative result difference between the benchmark and the simulations with the varied heart characteristics, a measure named Deviation is defined:

**Definition 1** (Deviation). Suppose \( u \in \mathbb{R}^m \), \( \hat{u} \in \mathbb{R}^m \) then the deviation \( d \) of \( \hat{u} \) with respect to \( u \) is defined by:

\[
d = \frac{\sqrt{\sum_{i=1}^{m}(u(x_i) - \hat{u}(x_i))^2}}{\sqrt{\sum_{i=1}^{m}u^2(x_i)}}
\]  

(3.1)

In the following chapters, \( u \) will be the value of electrical potentials on the body surface: the solution of the stationary model from the reference simulation; \( \hat{u} \) will be the solution of the stationary model from the simulation with varied heart physiology or anatomy characteristics.

3.5 Experiment Setups

The experiments are conducted according the property of heart physiology and anatomy characteristics: the conductivities listed in the Table 3.3.1 are experimented separately; for the fiber orientations, all the fibers are rotated isotropically, the experiments are conducted based on dividing the rotation on the fibers into two directions: clockwise and counter-clockwise; for the heart position the experiments are based on the fact that the displacement of the heart can be leftward, rightward, backward and forward. The experiments on the orientation of the heart are also based on two directions: clockwise and counter-clockwise. To isolate the effect of the variation on the potentials, only one of those characteristics will be varied in each experiment. The detailed setup of the experiments are listed in Table 3.6.

To show the distribution pattern of the potentials on the body surface, the torso boundary points are indexed and ordered clockwisely with respect to their positions on the torso boundary, see Fig 3.6.
### Table 3.6: The experiment setups.

<table>
<thead>
<tr>
<th>Heart characteristic</th>
<th>Variation Range</th>
<th>Variation type</th>
<th>NO. simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_e^T )</td>
<td>1.6 ms/cm</td>
<td>Increment</td>
<td>17</td>
</tr>
<tr>
<td>( \sigma_e^L )</td>
<td>5.0 ms/cm</td>
<td>Increment</td>
<td>51</td>
</tr>
<tr>
<td>( \sigma_i^T )</td>
<td>0.4 ms/cm</td>
<td>Increment</td>
<td>5</td>
</tr>
<tr>
<td>( \sigma_i^L )</td>
<td>1.7 ms/cm</td>
<td>Increment</td>
<td>18</td>
</tr>
<tr>
<td>Fiber orientation</td>
<td>18.8 degree</td>
<td>Clockwise rotation</td>
<td>189</td>
</tr>
<tr>
<td>Heart position</td>
<td>3 cm</td>
<td>Rightward displacement</td>
<td>31</td>
</tr>
<tr>
<td>Heart position</td>
<td>3 cm</td>
<td>Leftward displacement</td>
<td>31</td>
</tr>
<tr>
<td>Heart position</td>
<td>2.5 cm</td>
<td>Backward displacement</td>
<td>26</td>
</tr>
<tr>
<td>Heart position</td>
<td>1.0 cm</td>
<td>Forward displacement</td>
<td>11</td>
</tr>
<tr>
<td>Heart orientation</td>
<td>10 degree</td>
<td>Clockwise rotation</td>
<td>11</td>
</tr>
<tr>
<td>Heart orientation</td>
<td>10 degree</td>
<td>Counter-clockwise rotation</td>
<td>11</td>
</tr>
</tbody>
</table>

**Figure 3.6:** Order of torso boundary points. The points are numbered clockwisely.
3.6 Experiment Procedure

All the experiments share the same experimenting procedure described below:

1. Select one type of the heart physiology and antinomy characteristics.

2. Specify the variation range and the variation values.

3. Recalculate the selected heart characteristic according the variation values.

4. Run simulations.

5. Extract out the simulation results.

6. Calculate the deviation for all simulations.
Chapter 4

Software Tool Design

Each of the experiments designed in Chapter 3 consists of a series of simulations. It puts forward the request for a software tool (or program) which allows user to automate the simulations, the calculation of the deviations, and the visualizations of the simulation result, and so on. Further more, there are some heart physiology and anatomy characteristics which are not considered. In the future it might be a need to investigate the stability with respect to those. The investigation might need more sophisticated analysis for the simulation result, or the simulator is different from the simulator used, even to investigate the property of some other models. Thus the program needs to be carefully designed to achieve the maximum flexibility.

4.1 Requirements

For general purpose the program should have the following properties:

1. Conducting the experiments specified by the users. This is the most fundamental requirement.

2. Cross platform ability. The program must be portable between Microsoft windows platform and Linux platform.

3. The program must be well documented.
4.2. PROGRAMMING PARADIGM

4. Plug and Play. Capable to add new type of the heart characteristics to experiment without changing any existing code. When needed, the components of the program can be easily customized or replaced.

5. It has to have ability to export the current work and import the work back later.

6. Visualizing the experiment result and presenting it on the GUI.

7. It needs to have an easy to use graphical interface. The detailed actions it must be able to handle are showed in use case diagram, see Fig 4.1.

8. Organizing the files used and created in the experiments in a systematic way.

As the supporting program for the experiments defined for this thesis, it has to meet the following requirements:

1. Being able to work with diffpack simulator.

2. Currently it has to be able to handle the variations in the cellular conductivity, the heart fiber orientations, and the heart geometry, in 2D case.

3. Visualizing the heart characteristics mentioned above and display them on the GUI.

4. Visualizing the simulation results and display them on the GUI.

5. Analyzing the experiment results according given criterion, visualizing and display them on the GUI.

6. Easy to migrate from 2D case to 3D case

4.2 Programming Paradigm

It mentioned four programming paradigms in [38, 7]: the imperative programming, the object-oriented programming, the logic programming and the
Figure 4.1: Use case of the system
4.3. ARCHITECTURE PATTERN

functional programming. For this program the object-oriented paradigm should be the only choice. Firstly though the requirement of Plug and Play does not necessarily mean the object oriented approach, it is a lot easier to implement with it. The components of the program can be divided into several types. For each type of component, a superclass can be used to declare the necessary interface to the other types of components. When the new component is needed, the subclass can be derived from the superclass with the interface implemented. The implemented interface guarantees the smoothly working of the new component with other components. Secondly the requirement of the graphical user interface suggests the object-oriented paradigm clearly as the property of GUI is best modeled by the object oriented approach. The actions initiated by the user can be modeled as the messages sent to the GUI objects, those objects react to the messages accordingly. As early as in [53] the object oriented approach for GUI is suggested. Now the application of the object oriented approach on the GUI design is fully developed, for example in [37] and [19]. Application of those well developed technique would greatly decrease the complexity of the GUI programming.

4.3 Architecture Pattern

In software engineering field scientists have paid a lot of attention to the architectural pattern, and many great architectural patterns have been proposed, see [5]. Given the object oriented approach and the requirement of the graphical user interface, there exist three known patterns among others: Model-View-Controller (MVC) [53], Presentation-Abstraction-Control (PAC) [15] and Naked Objects [51].

MVC divides program into 3 parts: the Model part is the representation of the business logic; the View part renders the business logic into the proper form for interaction; the Controller part takes care of the interaction between the user and the program. PAC is similar to MVC, the Presentation part is corresponding to the View part in MVC; the Abstraction part is corresponding to the Model part in MVC. PAC has a recursive logic which leads to a hierarchical structure. The main disadvantage of PAC is that the communications between the objects can only happen with the help of
the corresponding control parts, which leads to complicated communication mechanism.

The Naked objects dedicates to make the programmer concentrate on the business logic, the interfaces to the business logics are created automatically. It sounds promising, but this needs support from an extra layer which fulfill the roles of View and Controller, and more have to be considered\textsuperscript{51}. Since it’s too complicated, thus it is not applied in this thesis.

The MVC pattern is simpler than the PAC pattern, it leads to less complex specifications and implementation \textsuperscript{27}. But this pattern is not completely suitable to this program. The heart characteristics are different from each other. The routine written to handle one kind of heart characteristics can not handle other kind. Currently only a part of them are involved in this thesis. When it comes to the issue of extending the program to be able to experiment on the non-involved heart characteristics, the single layer architecture forces the programmer to change the code of the program. To avoid both this problem and the communication complexity the PAC pattern causes, the new pattern is developed with the inspiration from Hierarchical-Model-View-Controller \textsuperscript{8}. The sketch figure of the pattern is shown in Fig 4.2. The difference between the MVC pattern and the new developed pattern is that the latter pattern has a hierarchical structure; the difference between the new developed pattern and PAC is that not all the communications between the components of the mixed pattern have to happen with support of corresponding Control-View part.

\section{4.4 Classes}

The classes are constructed according to the architecture pattern. For each component of the program, the functionality of the Controller, view parts for each component are put in one class, the functionality of the Model part for each component is put into another class.

There are mainly 5 types of Model class defined in the program:

1. \textit{perturber}. The \textit{perturber} class is responsible for dealing with the heart physiology and anatomy characteristics. For each type of the char-
acteristics, there should be a corresponding perturber. The perturber is mainly focusing on the operations in the heart characteristics. Currently there are perturbers for three kinds of heart characteristics: heart conductivity, heart fiber orientation and heart geometry. All the core functionalities are implemented in such a way that they are able to work with both two dimensional case and three dimensional case.

2. analyzer. The class analyzer concentrates on analyzing the experiment result which consists of the data provided by other classes such as perturber and simulator. The analysis is done according to certain criterions.

3. simulator. The class simulator is responsible for the simulation related issues such as extracting out the simulation result and visualizing the simulation result.

4. experiment. The class experiment is the fundamental class in the program, it mainly contains 3 types of object:
(a) The Object(s) of perturber class
(b) An object of analysis class
(c) An object of simulator class

It has the responsibility to initialize the configuration widget of its member object when necessary, drive all its objects to do their jobs and get the relevant result. In addition, the experiment is responsible to make sure that every simulation is done within its own directory such that the simulations won’t mess up with each other.

5. experiment_container. The class experiment_container is simply a container of the objects of the class experiment. It is responsible to initialize an experiment object with the experiment name, or remove a experiment object from it. Its fundamental functionality is to start the selected experiments by the user. Another important functionality is to export the current work and import the work later again. This is necessary as it always needs to save the done experiments for future revision.

The Controller-view classes for the corresponding Model classes are listed following:

1. interface. The main GUI is provided by the class interface and divided into two parts as shown in figure 4.3, the left widget is for the experiments management, the right one is reserved for integrating other components GUI. The menu bar is put on top of the main GUI.

2. experiment_interface. The GUI for the class experiment is provided by class experiment_interface. This GUI is integrated into the main GUI. As shown in figure 4.4, the main widget is divided into three parts located side by side. The left widget is responsible for initializing the experiment setup widget; the middle widget is responsible for displaying the visualization of the heart characteristics; the right widget is responsible for displaying the visualization of the analysis result and the visualization of the simulation result. The configuration widget for experiment is shown in Fig 4.5. It has the the view over the available
4.4. CLASSES

Figure 4.3: The Main GUI

Figure 4.4: The GUI for the class *Experiment*.
perturbers, the added perturbers, and the other components like the analyzer and the simulator.

3. The Controller-view classes for the class analyzer, perturber and simulator. As analyzer, Perturber and simulator are the subject of Plug and Play, the physical appearance of their GUIs are totally depending on themselves. In this thesis the analyzer, Perturber and simulator only offer the necessary configuration widget, see Appendix D.1 for the screenshots. The visualizations from them are presented on the GUI of the experiment

The relations between the Model classes and the Controller-view classes are shown in Fig 4.6.
Figure 4.6: Object Relations
4.5 Inheritance

In order to meet the requirement of Plug and Play, the APIs (interface) between the components must be clearly defined, and every component must follow them. Thus the inheritance is employed to help define those APIs. This inheritance is only applied to the Model classes as the Controller-view classes are highly dependent on the Model classes.

The functionalities which should be inherited include:

1. The main functionalities of each component, and the functionality of providing the proper GUI for each component.

2. Functionality of visualizing the relevant data.

There are several super classes defined for the main classes defined in Chapter 4.4:

1. `visualize`. It defines the prototype method for the functionality of visualizing the data. It is the super class for all the others.

2. `template_experiment`. This class is the super class for the class `experiment`. It defines the prototype methods for the tasks the class `experiment` should have:

   (a) Conduct the experiment.

   (b) Visualize the experiment result.

   (c) Generate the proper user interface.

3. `template_analyzer`. This class is the super class for the class `analyzer`. It defines the prototype methods for the functionalities the class `analyzer` should have:

   (a) Analyze given data according certain rules.

   (b) Visualize the analysis result.

   (c) Generate the proper user interface.
4.6. WORKFLOW

4. template_simulator. This class is the super class for the class simulator. It defines the prototype methods for the functionalities the class simulator should have:

(a) Prepare the simulations.
(b) Perform the simulations and return the simulation results.
(c) Visualize the simulation results.
(d) Creating the proper user interface.

5. template_perturber. This class is the super class for the class perturber. It defines the prototype methods for the functionalities the class perturber should have:

(a) Make the variation in the heart characteristics and return the variation parameter and the varied heart characteristics.
(b) Visualize the variation parameter and the varied heart characteristics.
(c) Generate the proper user interface.

The inheritance tree is shown in Fig 4.7. The communications between the objects of derived classes are done by message passing [4] which is one of the key properties of the object oriented approach. Considering that this program does not have the requirement of distributed computations, the message passing simply takes the form of function calling. As shown in Fig 4.8 the communications are mono-directed.

4.6 Workflow

4.6.1 Main Workflow

Two main workflows are considered in this project:

1. the experiment workflow.

2. the workflow for exporting the current work and importing the work exported previously.
CHAPTER 4. SOFTWARE TOOL DESIGN

Figure 4.7: The inheritance tree

Figure 4.8: The function call direction
4.6. WORKFLOW

It can be seen in Fig 4.9 that the program can start either one of those two workflows. The execution of the program can change from the one workflow to the other. The actions involved in both workflows are shown in this figure. The workflow goes from start point to the direction pointed by arrow in the figure.

4.6.2 Experiment Workflow

As shown in Figure 4.9, the experiment workflow is constituted by four actions:

1. Configuring the experiment. In this action the experiment is added and all the parameters needed are also given.

2. Doing the experiment. The experiments are done by calling the per-
Figure 4.10: The workflow of adding the experiment.

3. Visualizing all the experiment data, including the simulation results, the variation parameters and the varied heart characteristics, the analysis results.

4. Browsing over the visualized experiment result.

Each of those actions can be furtherly divided into sub-actions which are shown in Fig 4.10, 4.11, 4.12 and 4.13. The workflow of configuring the experiment is shown in Fig 4.10. It starts with adding the experiment, goes to adding the perturbers, then configuring the variation in the heart characteristics, at the end the variation is saved.

Fig 4.11 shows the workflow of performing the experiment. It’s a little more complicated than the process of configuring the experiment.

The workflow of visualization process is shown in Fig 4.12. It only contains three steps: determine the data type, find the right visualizer, then
Figure 4.11: The Workflow of doing the experiment
Figure 4.12: The Workflow of visualizing the experiment result.

Figure 4.13: The workflow of browsing the experiment visualization
visualize it.

Fig. 4.13 shows the workflow of browsing over the visualizations of the experiment result.
4.7 Dataflow

There are several different types of data in this program:

1. The heart characteristics and their visualizations.

2. The variation parameters. The variation parameters define the variation in the heart characteristics mentioned above. For each type of heart characteristics, there is one corresponding variation parameter.

3. The simulation related information, including the path of the directories associated with all the simulations, the simulation results and their visualizations.

4. The analysis result and its visualization.

The data flowing direction is related to the role of each class introduced in section 4.4. The class Experiment-container is only a container of the objects of class Experiment, thus there is no data flowing between the object of Experiment-container and the others. The class Experiment plays a central role in the program, it gets all the data necessary from some member objects and feeding it to the other objects when needed. This feature suggests a centralized dataflow model shown in Fig 4.14.
Figure 4.14: The dataflow
Chapter 5

Software Tool Implementation

5.1 Choice of the Programming Language: Python, Ruby and Perl

Since Short Code was born in 1950s [56], a great number of programming languages have been invented including system programming languages and scripting languages. System programming languages are meant to build applications from the beginning; scripting languages are designed to integrate existing programs and “For gluing and system integration, applications can be developed 5-10x faster with a scripting language” [48]. Based on the fact that the simulator of the stationary model and a bunch of visualization tools are available, scripting languages such as Perl [69], Python [39] and Ruby [66] would be good choices. As those languages are designed for similar tasks, there exists very much in common among them, Table 5.1 provides a rough overview.

Though some in common, there are still some differences. Perl addresses object-oriented programming problems, but originally it’s a procedural programming language, the object-oriented programming is an add-on feature. A quota from the output of running the command perldoc perlobj speaks of the fact:

... 

1. An object is simply a reference that happens to know which class it
5.1. CHOICE OF THE PROGRAMMING LANGUAGE: PYTHON, RUBY AND PERL

<table>
<thead>
<tr>
<th></th>
<th>Object oriented programming</th>
<th>High level</th>
<th>Support functional programming</th>
<th>Garbage collected</th>
<th>Expressive</th>
<th>Fast to develop in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Python</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ruby</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5.1: The common aspects of Perl, Python, Ruby

belongs to.

2. A class is simply a package that happens to provide methods to deal with object references.

3. A method is simply a subroutine that expects an object reference (or a package name, for class methods) as the first argument.

... This feature causes many inconveniences, at least for object initiation\(^1\) and dynamical methods loading\(^2\). Perl uses special signs such as @ and $ to mark both the references types and the efault variables. Perl also encourages the programmer to write code differently from others by claiming “There’s more than one way to do it.”. These features of Perl lead to less readable source code, thus less maintainable program. This program is not only meant to provide support for the current experiments, but also stand as the framework which allows the others to extend or even develop the new program for the similar tasks. Hence the readability and maintainability are crucial. The Perl program are typically weak on the both issues.

Ruby is a relatively new scripting language, the pure objected-oriented programming scheme makes Ruby attractive to the programmer. As Ruby

\(^1\)Needs method \texttt{bless} \\
\(^2\)Needs \texttt{Class::Prototyped}
Inheritated many concepts from Perl, the special signs used in Perl also appear in Ruby, which also leads to less readable and maintainable code like Perl. More importantly, as a relatively new language, Ruby provides less programming libraries than the other does and lacks support from third party. The Ruby documentation is not as good as the others. All those make Ruby not the ideal programming language for the program needed in this thesis.

In contrast, Python is designed to be an object-oriented programming language. It is easy to initialize the Python class instance. The dynamic method adding and overloading are straightforward. It does not use any special signs, even the braces commonly used in many other languages such as C/C++ and Java to mark the code block are omitted. This feature forces the programmer to write the source code in a cleaner way. Python supports multi-inheritance and has a good documentation; it also has plenty of third party supporting libraries. Given these advantages, Python is chosen to be the programming language in this thesis.

5.2 Visualization Tools

There are mainly three types of information need to be visualized: the heart characteristics and their variation parameter, the analysis result, the simulation result. Among the informations mentioned above, the heart physiology and anatomy characteristics and the analysis result are visualized by Gnuplot; the others are visualized by plotmtv. Gnuplot is a command-driven interactive function and data plotting program, it is used broadly in scientific research field. Further more it exists an third party Python interface which allows users to manipulate Gnuplot through python’s pipe. The plotmtv is a fast multi-purpose plotting program for visualization of scientific data in an X11-window environment. Diffpack tool simres2mtv can filter the simulation result into the format needed by plotmtv. The detailed information about gnuplot, plotmtv, simres2mtv can be found in [32] and reference therein; the python interface for gnuplot can be found in [34] and reference therein.
5.3 GUI Toolkit

Tkinter [21] and Pmw [21] are selected to construct the GUI. Tkinter module is the standard Python interface to the Tk GUI toolkit [17]. Pmw is a toolkit for building high-level compound widgets in Python using the Tkinter module. Tkinter and Pmw have the following features:

1. Cross platform. Tk are available on most Unix platforms, on Windows and Macintosh systems.

2. Code simplicity. Pmw provides megawidgets constructed by using other Tkinter widgets as the components. Programming in Pmw megawidget gives simpler and more readable code than programming directly in Tkinter widgets does.

3. Documentation. As a standard Python module, Tkinter is well documented. Pmw provides a comprehensive reference menu with example code. These documentations make it easier to understand the behaviors of the program written in Tkinter and Pmw, eventually easier to change the program as desired.

4. Functionality. The Tkinter and Pmw GUI provides full functionality for interactions between the user and the GUI.

5. Extending the program. Tkinter is wildly used in the Python community. For example the pyOpenGL [22] is interoperable with Tkinter; Mayavi employs Tkinter to build up its GUI interface and can render data in the VTK [57] format. This makes it possible to implement program’s own visualization subroutine or even to integrate the Mayavi’s rendering widget into this program. As Diffpack offers a tool to convert the simulation data into VTK format, this is a feature desirable. Mayavi is available in public domain[2].

5.4 Documentation Tool

There are many different documentation tools available, including pydoc, Happydoc andepydoc. For this program, documentations in both PDF and
HTML format are needed. As the Python’s standard documentation tool, pydoc only supports the plain text and HTML as output format. And it does not support any mark language. Happydoc is another documentation tool which supports several different doc string mark languages including StructuredText. Those mark languages are powerful but complex. Furthermore, Happydoc does not support the PDF format output. Epydoc is another third party tool for generating API documentation for Python modules. It also extracts the API documentation from Python Doc string. A simple but practical mark language Epytext is employed. It can generate both HTML format and PDF format documentation among others. epydoc is selected as the documentation tool because of its simply doc string mark language and the support for PDF format documentation. The details about epydoc and happydoc can be found in [33] and references therein.

5.5 Implementation Details

5.5.1 Affine Transformation

The affine transformation [3] are employed to form the simple variations in the heart geometric properties such as positions, orientations, and the part of shapes. For more sophisticated variations, for example, the shape of the heart recovered from myocardial infarction, some other special approaches must be engaged. Affine transformation can be decomposed into four basic transformations: translation, rotation, shear, and scaling. By employing the homogeneous system and denoting 3 dimensional case with $3D$, 2 dimensional case with $2D$, the transformation matrixes for those four basic transformations are [3 43]:

1. Translation

$$T_{3D}(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_{2D}(x, y) = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix},$$ (5.1)

where $x, y, z$ denote the translation in x-axis, y-axis, z-axis, respectively.
5.5. IMPLEMENTATION DETAILS

2. Rotation

\[ R_{3D}(x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos x & -\sin x & 0 \\ 0 & \sin x & \cos x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_{3D}(y) = \begin{bmatrix} \cos y & 0 & \sin x & 0 \\ 0 & 1 & 0 & 0 \\ -\sin y & 0 & \cos x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ R_{3D}(z) = \begin{bmatrix} \cos y & -\sin x & 0 & 0 \\ -\sin y & \cos x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_{2D}(z) = \begin{bmatrix} \cos y & -\sin x & 0 \\ -\sin y & \cos x & 0 \\ 0 & 0 & 1 \end{bmatrix} \]  

(5.2)

where \( x, y, z \) denote the rotation angle around x-axis, y-axis, z-axis, respectively.

3. Scaling

\[ S_{c3D}(x, y, z) = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad S_{c2D}(x, y) = \begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{bmatrix} \]  

(5.3)

where \( x, y, z \) denote the scaling factor in x-axis, y-axis, z-axis, respectively.
4. Shear

\[
S_{3D,xy}(a, b) = \begin{bmatrix}
1 & 0 & a & 0 \\
0 & 1 & b & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
S_{3D,xz}(a, c) = \begin{bmatrix}
1 & a & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & c & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
S_{3D,yz}(b, c) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
b & 0 & 1 & 0 \\
c & 0 & 0 & 1
\end{bmatrix}
\]

\[
S_{2D,x}(a) = \begin{bmatrix}
1 & a & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix},
S_{2D,y}(b) = \begin{bmatrix}
1 & 0 & 0 \\
b & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

(5.4)

where \( x, y, z \) denote the shearing axis and \( a, b, c \) denote the shearing factor in those axis, respectively.

Then the affine transformation \( L \) on a vector \( v \) can be expressed by the combination of those:

\[
L(v) = M_0 \ast M_1 \ast M_2 \ast M_3 \ast v, M_0, M_1, M_2, M_3 \in \{T, Sc, S, R\}
\]

(5.5)

5.5.2 Header File

Normally there exist some variables which are constant and utilized by different parts of the program, while they are seemingly not belonging to any part. In C/C++ language the header file can be used to declare those variables. Whenever needed, statement “#include” is issued to make those variables accessible. For constant variables, a special specifier named \textit{const} is utilized in this variables declaration/definition. In python the statement ”import” does the same as “#include” does in C/C++, but Python does not support constant variables. However, if the variable is put into a class, the special method “\texttt{setattr}” can be used to implement similar restriction on the variable as \textit{const} does. Though it can not prevent the user to intentionally change the variable value\(^3\), it does avoid the accidental change of the variable value.

\(^3\)By directly assigning the value to the self._dict_\[variable name\]
Variable name | Meaning | Value
--- | --- | ---
S_WIDTH | small width size of GIF image | 300
S_HEIGHT | small height size of GIF image | 200
M_WIDTH | middle width size of GIF image | 400
M_HEIGHT | middle height size of GIF image | 300
L_WIDTH | large height size of GIF image | 640
L_HEIGHT | large height size of GIF image | 480

Table 5.2: Constants used in the program

The constant variables and their values defined in this header file are shown in Table 5.2.

5.5.3 Visualizations on The GUI

There are two ways to display the visualization on the graphical user interface:

1. Implemente a visualization routine with support from Tkinter and PIL (Python Image Library), even for the 3D visualization, see [21].

2. Utilize the existing visualization tools to do the visualizations, then display them on the interface.

To implement the visualization routine, the data format which can be recognized by the visualization routine must be defined. Every time the data need to be visualized, it must be converted to this format. It essentially requires a lot of work. It is better to utilize the existing tools to do the work. The simres2\* programs shipped together with the Diffpack package can filter simulation data into the proper format; Gnuplot, matlab, and plotmtv are all tools can be used for the visualization.

If the rendering window of either one of the visualization tools can be embedded into to the GUI developed in Tkinter and Pmw, the interaction between the visualizations and the users would be kept as it is in the original program. Unfortunately there does not exist any simple way to implement
it. There is no API which can be used for this purpose. An alternative is to make the visualization tools run in the background and save the visualization results in a file with certain format supported by Tkinter and PIL. The user interface reads in such file and renders it on its rendering widget. This file format is set to be GIF. Tkinter and PIL support almost 30 graphic file formats including GIF [39]. Gnuplot and Matlab also support GIF format, for tools which don’t support GIF file format lots of third party tools can be used to convert from a bunch of file formats into GIF, for example Imagemagick [60]. Selecting GIF format gives three benefits:

1. GIF is among the most widely used image file formats.

2. GIF supports animation. This adds to possibility of displaying visualized time-dependent simulation results.

3. GIF supports transparent background. This increases the visual effect quality on the user interface.

There are three types of information which need to be visualized. Some of them can be visualized and directly saved in GIF format, some of them can not:

1. The heart characteristics. The visualization of the conductivity is rendered as text, saved in the GIF format with the help of PIL functions. Then Imagemagick is employed to make transparent background. The heart fiber field data and heart geometry data are visualized with the help of Gnuplot, directly saved in the GIF format with transparent background. The size of GIF figures are defined in file header.py.

2. The analysis result. This kind of data can also be visualized by Gnuplot, directly saved in the GIF format with transparent background.

3. The simulation result. This kind of data is converted into the plotmtv data file format, then visualized with the help of plotmtv. The output file format is post script. The Imagemagick is used to convert it into transparent GIF, and adjust it into the proper size.
5.5.4 Simulation Result Extraction

The stationary model generates a grid file with name started with ".", and ended with ".grid" to give the information of the grid used in the simulation. The simulation result file with name started with ".", and ended with ".field" to give the informations of the value on each node which is defined within the grid file. With the help of those 2 files the simulation result extraction can be done in 3 steps:

1. Extract out the coordinate of nodes from the ".grid" file.
2. Extract out the values on the nodes according the coordinates of nodes from the ".field" file.
3. Return both the values on the nodes and the corresponding coordinates of nodes.

The reason of extracting both the values on the node and the coordinates of nodes is that the node numbering scheme in the ".field" file is different from simulation to simulation though the node numbering scheme in the input grid files are the same. Only by the nodal number, the extracted simulation result might be wrong. Further more when there is a need to compare the nodal values from different simulations, it’s better to sort the grid nodes in every simulation by their natural locations, then the values on those nodes can be compared. Thus the coordinates of those nodes must also be extracted out. For 2D case the sorting can be done as following:

1. Define an arbitrary point, say $p$, located inside the torso contour.
2. Translate the coordinate system of the points into another coordinate system with the coordinates of the point $p$ as the origin.
3. Express the position of transformed points in terms of polar coordinate system $(r, \theta)$.
4. Sort points with respect to $\theta$. 
5.5.5 Preparation for Simulation

There are some preparations needed to be done before simulation. First a directory has to be created for the simulation. All results of each single simulation are saved in its own directory, which helps to trace back the informations of simulations. After directory has been created, all the files needed to conduct a simulation are copied into this directory except the executable simulator itself. Some of those files may be changed by the perturbers afterwards.

5.5.6 Automatic Load of Perturber

In Section 4.1, it is mentioned in the requirement that the perturbers have to be able to Plug and Play. There could be many different perturbers added to the program in the future. With the help of python regular expression and dynamic code execution, those perturbers can be loaded automatically. The loading process is divided into five steps:

1. Find the directory which contains the perturbers.
2. Read in the name of all the files located in this directory.
3. Identify the perturbers with the help of Python regular expression.
4. Construct the code to load the identified perturbers.
5. Execute the code.

Some restrictions have to be put on the name of perturber and the location where the perturber locates:

1. The perturber must locate in the same directory as the module Experiment does.
2. The name of the perturber must be identical to the basename of the file which contains the perturber.
3. The name of the file containing the perturber must meet the predefined requirement. A python regular expression match pattern is used to
identify the file. This match pattern can be given as the argument of the constructor of the module Experiment.

The default regular expression pattern is

```
'(^make_[A-Za-z]+_perturbation)\.py$')
```

All the files with the name started with “make_” , followed by the combination of 26 English letters, and ended with “.perturbation” are identified and all modules with the same names defined in those files are imported.

5.5.7 Abstract Super Classes

In Chapter 4.5, several super classes are defined to provide the necessary APIs. To force all the derived classes to reimplement all the methods, C++ provides the concept abstract class [61] and Java provides the concept interface [17]. Python does not support either one of these. However, by taking advantage of Python’s exception-handling mechanism, the super class can be implemented to have the similar effect on the sub-class as the C++ abstract class does:

```python
class AbstractSuperClass(object):
    def foobar ():
        raise Exception, """
        This is a abstract function which is inherited from base class %s. It must be re-implemented. """"%(self.__class__.__name__)
```

When the instance method of the class inherited from AbstractSuperClass is called, and if it is not implemented in sub-class, the implementation of this method in super class is activated. It raises an exception instance to inform the user.

For the detailed informations about the super-class defined in 4.5 see Appendix D.2.
5.5.8 Experiment Exportation and Importation

To be able to export the current work and import it back again, the program needs to have ability to save all the necessary runtime objects on the disk and restore them from disk without losing any information. As this program does not have database involved three modules are available in Python for doing this:

1. pickle
2. marshal
3. shelve

Module marshal is intended to the internal use of supporting reading and writing the “pseudo-compiled” code for Python modules of .pyc files. Module shelve is used to save the Python objects in a Python dictionary-like way. It associates a dumped object with a given key which can be used to restore the dumped object afterwards. The pickle module simply converts the objects into the byte stream and saves it on the disk. When needed module pickle converts them back to object hierarchy. Python also has a C version pickle module named Cpickle, it’s 1000 time faster than pickle module. Because of its speed and simplicity the Cpickle module is used in the program. But none of the modules mentioned above can work with Tkinter objects since those objects are not python objects. Tkinter is only a object oriented python interface to Tk GUI toolkit, the operations on Tkinter objects are actually handled by Tk GUI toolkit. Thus the state of the user interface is not saved. Those Tk objects are rebuilt according the data stored in other non-tk objects.

5.5.9 Graphical Exceptions

It is not practical that the informations of the exceptions occured in the program execution are sent to the text-based system console. With the help of some python modules, such informations can be displayed on the graphical interface. In python the standard input, the standard output and the error streams are abstracted as file objects assiged to the variables with name
sys.stdin, sys.stdout, sys.stderr; those objects interact with operating system’s standard input, standard output and standard error.

In Python module sys, there exists a method excepthook which prints out the given traceback and exception to sys.stderr. Define a method to replace this method:

```python
def excepthook( type, value, tr):
    gui_except.write(traceback.format_exc())

sys.excepthook = excepthook
```

Whenever a exception is generated, the method excepthook is called, then the method gui_except.write displays the relevant information on the graphical interface.

For Tkinter objects the exceptions are handled in the mainloop method. It does not utilize the sys.excepthook to output the exceptions and tracebacks, but directly write the information to sys.stderr. In Tkinter module Tk, it defines a method report_callback_exception which reports exception to sys.stderr; this method can be overloaded to display the informations to the graphical interface:

```python
class myTk(Tk):
    def report_callback_exception(self, exc, val, tb):
        sys.excepthook(exc, val, tb)

Tkinter.Tk = myTk
```

5.5.10 From 2D to 3D

In this thesis all the simulations are done in two-dimensional geometry. But the simulator itself is coded for both two-dimensional and three-dimensional cases. Considering future’s application all the perturbers are implemented to be able to cope with both dimensions. The implementation is fairly simple, an variable of integer type is employed to indicate the current dimension, the program tests this variable and processes accordingly. Still the program is not totally ready for three-dimensional simulation. The variations in the
heart geometry require to re-generate the grid of the heart geometry. The current grid generator only works with the two dimensional grid generation, thus the Python module which prepares data for the grid generator is also implemented to only cope with the two-dimensional cases. To migrate from 2D to 3D, a new grid generator has to be implemented or selected from existing ones, the new Python module to prepare data for selected grid generator might also need to be reimplemented. For informations about existing grid generators, see [49].

5.5.11 File Organization

This program consists of many components: the C++ simulator, the Python modules, the data files needed by the modules, the font files, the data files containing the original data needed by the simulator, a bash script which runs the program and so on. There are also a bunch of files generated in each experiment, including the files generated by the simulator and analyzer, and the files containing the visualized data from the experiment. Those files must be organized in a systematical way such that for the user it’s easy to find the relevant information.

Directory Hierarchy

The program directory hierarchy is listed below. The root directory of this program is named project.

project
|-- bin
|-- create_doc.sh
|-- data
 | |-- axis_trans.gif
 | |-- basis.axis
 | |-- basis2d.axis
 | |-- contour.contour
 | |-- contour.gif
 | |-- contour.corser.contour
5.5. IMPLEMENTATION DETAILS

|-- contour_finer.contour
|-- contour_finest.contour
|-- field.gif
|-- heart.gif
|-- ischfile
|-- fonts
|  |-- arial.ttf
|-- install.py
|-- lib
|-- modules
|  |-- AbstractClassException.py
|  |-- GuiException.py
|  |-- README
|  |-- analysis.py
|  |-- diffpack_simulator.py
|  |-- experiment.py
|  |-- grid_maker.py
|  |-- header.py
|  |-- interface.py
|  |-- make__vec__perturbation.py
|  |-- make_documentation.py
|  |-- make_extraconds_perturbation.py
|  |-- make_field_perturbation.py
|  |-- make_heart_perturbation.py
|  |-- make_intraconds_perturbation.py
|  |-- make_perturbation.py
|  |-- model.py
|  |-- simulator.py
| |-- triangle
|   |-- visualize.py
|-- run.sh
|-- simu_data
|  |-- app
|  |-- case.con
The file *README* in root directory is the note of the program development. Subdirectory *data* contains files needed by python modules. Some fonts files needed to render text into image locate in subdirectory *fonts*. The directory *modules* holds all the Python modules of this program, the grid generator is also found there. The script file *run.sh* is used to run the program. The subdirectory *simu_data* contains the C++ simulator program and all the files it needs for the simulation.

**Naming Scheme**

A subdirectory named “experiments” is not in the directory hierarchy listed above. This directory will be generated when doing the experiments. By default, subdirectory “experiments” is located in the directory where the program runs, it holds all the files created in the experiments. For each experiment a corresponding directory is created under directory “experiments”, the name is the same as that of the experiment. Each experiment has it’s own name either given by default or given by the user. Each simulation is associated with a sub-directory of the experiment directory. The name of this sub-directory is formed by appending the ordinal number of the simulation to the experiment name. For example, if the experiment named “example” consists of 2 simulations, the name of the directory for the first simulation would be “example0”; the name of the directory for the second simulation would be “example1”. The directory tree can best show it:

```
  experiments
  |-- example
  |    |-- Bias
  |    |-- Cond_extra.gif
  |    |-- Potential(unit: mv)
  |    |-- e2
```
In the directory tree shown above, “example” and “test” are two directories for the corresponding experiments. Experiment “example” consists of 3 simulations “example0”, “example1” and “example2”. Other files in the experiment directory are the files containing the visualizations of the experiment result.
Chapter 6

Experiment Results

As described in Chap 3, the sensitivity experiments have been conducted for four different heart characteristics: the extra-cellular conductivity, the intra-cellular conductivity, the heart fiber orientation, and the heart geometry. For convenience the setup of experiments are listed again here in Table 6.1. The experiment results are presented in following sections and categorized according to the experimented heart characteristics. The measure “Deviation” defined in Section 3.4 is used to summarize the relative differences between simulations and give an indication for the sensitivity of the stationary bidomain model.

6.1 Effects of Variations in $\sigma_e$

Fig 6.1 shows the deviations of the simulations with the increased $\sigma_e^L$. The deviation appears to be monotonically increasing along with the increment in $\sigma_e^L$. The relation between the deviation value and the increment in $\sigma_e^L$ resembles approximately a linear function, but with an small upward arc; Fig 6.2 shows that the relation between the deviation and the increment of $\sigma_e^T$ has the same pattern as the one shown in Fig 6.1. The largest deviation value is approximately 0.58 in the experiment for $\sigma_e^L$ and approximately 0.3 in the experiment for $\sigma_e^T$.

The distribution of actual potential values on the body surface are shown in Fig 6.3 and 6.4.
### 6.1. EFFECTS OF VARIATIONS IN $\sigma_E$

<table>
<thead>
<tr>
<th>Heart characteristic</th>
<th>Variation Range</th>
<th>Variation type</th>
<th>NO. simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{TE}$</td>
<td>1.6 ms/cm</td>
<td>Increment</td>
<td>17</td>
</tr>
<tr>
<td>$\sigma_{LE}$</td>
<td>5.0 ms/cm</td>
<td>Increment</td>
<td>51</td>
</tr>
<tr>
<td>$\sigma_{TI}$</td>
<td>0.4 ms/cm</td>
<td>Increment</td>
<td>5</td>
</tr>
<tr>
<td>$\sigma_{LI}$</td>
<td>1.7 ms/cm</td>
<td>Increment</td>
<td>18</td>
</tr>
<tr>
<td>Fiber orientation</td>
<td>18.8 degree</td>
<td>Clockwise rotation</td>
<td>189</td>
</tr>
<tr>
<td>Fiber orientation</td>
<td>18.8 degree</td>
<td>Counter-clockwise rotation</td>
<td>189</td>
</tr>
<tr>
<td>Heart position</td>
<td>3 cm</td>
<td>Rightward displacement</td>
<td>31</td>
</tr>
<tr>
<td>Heart position</td>
<td>3 cm</td>
<td>Leftward displacement</td>
<td>31</td>
</tr>
<tr>
<td>Heart position</td>
<td>2.5 cm</td>
<td>Backward displacement</td>
<td>26</td>
</tr>
<tr>
<td>Heart position</td>
<td>1.0 cm</td>
<td>Forward displacement</td>
<td>11</td>
</tr>
<tr>
<td>Heart orientation</td>
<td>10 degree</td>
<td>Clockwise rotation</td>
<td>11</td>
</tr>
<tr>
<td>Heart orientation</td>
<td>10 degree</td>
<td>Counter-clockwise rotation</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 6.1: The experiment setups

![Deviation VS Variation on Extra Cellular Conductivity](image)

Figure 6.1: The relation between the deviations and the increments in $\sigma_E^L$. 
It seems that the increment in $\sigma_e^L$ affects the potential values on the most area of the body surface; the variation in $\sigma_e^T$ gives much less impact on the the potentials than the increment in $\sigma_e^L$ does. This explains the fact that the largest deviation value is a little larger than 0.3 in the experiment for $\sigma_e^T$. In both experiments, the greatest change in potential value among simulations happens in the area around the point 80. The points which have the smaller potential values are located in this area. Note that the potentia distribution patterns are kept unchanged, this confirms with the primary finding declared in [25] in a sense that the conductivity of torso tissue is set to be isometric.
6.1. EFFECTS OF VARIATIONS IN $\sigma_E$

Figure 6.3: The comparison of the body surface potentials from the simulations with different increments in $\sigma_E^L$.

Figure 6.4: The comparison of the body surface potentials from the simulations with different increments in $\sigma_E^T$. 
6.2 Effects of Variations in $\sigma_i$

For experiments on intra cellular conductivity the relations between the deviation and the increase in $\sigma_i$ appear to be of the similar pattern to the one between the deviation and the increment in $\sigma_e$, but with less obvious arcs, see Fig 6.5 and 6.6.

Figure 6.5: The relation between the deviations and the increments in $\sigma_i$.

The results suggest that the increment in $\sigma_i^T$ has much more impact on the potential values on the surface compare with the one in $\sigma_i^L$. With less than 0.4ms/cm increment in $\sigma_i^T$, it gives the deviation value of about 0.78.

The deviation values conform to the distributions of potentials on body surface, see Fig 6.8 and 6.7. The distribution pattern of the potentials does not change when $\sigma_i$ is changed; the greatest difference of the potential value between simulations occurs in the area containing the point 80, its value is about 0.6mv in experiment for $\sigma_i^L$ and 0.5mv in experiment for $\sigma_i^T$. 
6.2. EFFECTS OF VARIATIONS IN $\sigma_i$  

Figure 6.6: The relation between the deviations and the increments in $\sigma_i^T$.

Figure 6.7: The comparison of the body surface potentials from the simulations with different increments in $\sigma_i^L$. 

Figure 6.8: The comparison of the body surface potentials from the simulations with different increments in $\sigma_i^T$. 
6.3 Effects of Variations in Heart Fiber Orientation

The relation between the deviation value and the increment in the rotation angle also resembles a linear function in both the experiments for the clockwise and counter-clockwise rotation in the fiber orientation, the observed greatest deviation value is around 0.9 which is of the simulation with 18.8 degree counter clockwise rotated heart fiber orientation and 1.2 which is of the simulation with 18.8 degree clockwise rotated cardiac fiber orientation, see Fig 6.9 and 6.10.

![Deviation VS Variation on Fiber orientation](image)

Figure 6.9: The relation between the deviations and the clockwise rotation angles in the fiber orientation.

The potential value distributions for several simulations are shown in Fig 6.11 and 6.12. For the clockwise rotation, the potential values are essentially increasing on the points numbered from 0 to about 90, decreasing on the points numbered from about 100 to 150 and then increasing again on the rest of the points. There also exist some points numbered from about 90 to 100 on which the actual potential values are almost neither increasing nor decreasing; the opposite behavior occurs for counter-clockwise rotation, the potential values are decreasing on the points numbered from 0 to about 60, increasing on the points numbered from about 80 to about 120, and
Figure 6.10: The relation between the deviations and the counter-clockwise rotation angles in the fiber orientation. Decreasing on the rest of the points.
6.3. EFFECTS OF VARIATIONS IN HEART FIBER ORIENTATION

Figure 6.11: The comparison of body surface potentials from simulations with the clockwise rotations in fiber orientation.

Figure 6.12: The comparison of body surface potentials from simulations with the counter-clockwise rotations in fiber orientation.
6.4 Effects of Variations in Heart Geometry

The relationship between the deviation values and the variations in the heart geometry can be seen in Fig. 6.13, 6.14, 6.16, 6.15, 6.17 and 6.18. Among those Figures, 6.14 shows that the 3cm leftward heart displacement gives the biggest deviation value about 0.5. The deviation value is also monotonically increasing while the variation value is increasing, but the increment of the deviation value show less smoothness compare to the one observed in the experiments on the other types of heart characteristics, see Fig. 6.17. Note that in Fig. 6.13 there is a small peak, this is because of the refinement of the grid of the heart geometry. With the less refined grid, it disappears.

The potential value distributions can be seen in Fig. 6.19, 6.20, 6.21, 6.22, 6.23 and 6.23. Those figures show that the distribution pattern does not change when the heart geometry is changed. Compare the potential value from different simulations with the one from the reference simulation for each point on the body surface, the biggest potential variation happens approximately in the area around point 70 with the value about 0.6mv while the heart is 3.0 cm leftward displaced. In each simulation with the varied heart geometry, the variations in the potential values are also inhomogeneous as it is observed in each simulation with the varied heart fiber orientation.
6.4. EFFECTS OF VARIATIONS IN HEART GEOMETRY

Figure 6.14: The relation between the deviations and the leftward displacements of the heart.

Figure 6.15: The relation between the deviations and the forward displacements of the heart.
Figure 6.16: The relation between the deviations and the backward displacements of the heart.

Figure 6.17: The relation between the deviations and the clockwise rotations of the heart.
6.4. EFFECTS OF VARIATIONS IN HEART GEOMETRY

Figure 6.18: The relation between the deviation and the counter-clockwise rotations of the heart.

Figure 6.19: The comparison of the body surface potentials from simulations with the heart rightward displaced.
CHAPTER 6. EXPERIMENT RESULTS

Figure 6.20: The comparison of the body surface potentials from simulations with the heart leftward displaced.

Figure 6.21: The comparison of the body surface potentials from simulations with the heart forward displaced.
6.4. EFFECTS OF VARIATIONS IN HEART GEOMETRY

Figure 6.22: The comparison of the body surface potentials from simulations with the heart backward displaced.

Figure 6.23: The comparison of the body surface potentials from simulations with the heart clockwisely rotated.
Figure 6.24: The comparison of the body surface potentials from simulations with the heart counter-clockwisely rotated.
Chapter 7

Discussion

7.1 The Stability

The behaviors of the stationary model have been examined through a series experiments. The model appears to be very stable to the inter-individual variations in some physiology and anatomy characteristics in a sense that:

1. the distribution patterns of the body surface potential values from simulations are not changed compare to the one from the reference simulation.

2. The deviation value increases monotonically while the size of the variation increases. The relationships between them resembles approximately linear function with different smoothness.

To check the robustness of the above observations, the experiments with transmural ischemia incorporated at two different positions are also conducted, see Appendix B and C. The results also produce the similar pattern.

Note that for the observation above to be meaningful, the numerical algorithm used to solve the model has to be numerically stable and consistent. The stability and consistency of the numerical algorithm insure that the numerical solutions converge to the real solutions[18].

In practice, to the problems such as bidomain model, it’s almost impossible to derive an analytical scheme to check the stability and consistency. The alternative is to conduct numerical tests on the algorithm [63]. As the
main concern of this thesis is not the stability and the consistency of the
numerical algorithm, the numerical algorithm is assumed to be numerically
stable and consistent with the original problem.

7.2 The Most influential Factor

The position of the ischemia region is found to affect the body surface po-
tential most. The variation in the ischemia region affects both the potential
values and the potential distribution patterns. In contrast, the size of the
ischemia region and the inter-individual variations in the heart characteris-
tics such as the heart conductivities, the heart fiber orientations, the heart
position, the heart orientation and the size of the ischemia region only affect
the body surface potential values, but not the potential distribution pattern.
See Figures in chapter 6 and Figures in Appendix B and C.

Another observation is that neither the variations in the heart position
nor in the heart orientation changes distribution of the potential values on
the heart surface, see Fig 7.1 and 7.2. This observation confirms to the result
about effect of heart position on epicardial potentials reported in [14].
7.2. THE MOST INFLUENTIAL FACTOR

(a) The result of the simulation without variations in the heart characteristics.

(b) Simulation with 3cm leftward displacement of the heart.

(c) The result of the simulation without variations in the heart characteristics.

(d) Simulation with 3cm rightward displacement of the heart.

(e) The result of the simulation without variations in the heart characteristics.

(f) Simulation with 1cm forward displacement of the heart.

Figure 7.1: Epicardial Potentials
CHAPTER 7. DISCUSSION

(a) The result of the simulation without variations in the heart characteristics.

(b) Simulation with 2.5cm backward displacement of the heart.

(c) The result of the simulation without variations in the heart characteristics.

(d) Simulation with 10 degree counterclockwise rotation of the heart.

(e) The result of the simulation without variations in the heart characteristics.

(f) Simulation with 10 degree clockwise rotation of the heart.

Figure 7.2: Epicardial Potentials 2
Chapter 8

Conclusion and Future Work

Earlier in [40] MacLachlan, M.C. et al concluded that subendocardial ischemia is indeed locatable by ST segment shift above the ischemic region on both the epicardial and torso surface. Later in [45], it is shown that the position and size of the ischemia can roughly be recovered from both pure observation data (resting potential on the body surface) and observation data with Gaussian noise added. To investigate the stability of the stationary bidomain model with respect to the inter-individual variations in the heart characteristics which are not considered in either [40] or [45], a series experiments are designed and conducted in this thesis. The experiment results show that the stationary model is stable to the inter-individual variations in the heart characteristics experimented so far in terms of the “Deviation” defined in Chapter 3.4 and the body surface potential distribution pattern in 2 dimensional case. The results also show that if the ischemia region is fixed, the variations in the body surface potential caused by the inter-individual variation in some heart data are predictable, it may indicate that the variations in the body surface potential caused by the inter-individuals may be separated from the one caused by the ischemia mathematically or empirically such that the ischemia region can be discovered more accurately. However, because it still remains unclear that how the model will behave itself to the non-explored inter-individual variation and the experiment results are not very deeply analyzed, the meaning of such stability observation is very limited. Thus the investigation of the stability of the model has to be done in

91
more extent and depth.

For both investigation and software tool there are some improvements could be done in future:

1. 3D simulation. Currently all the simulations are done based on the 2D geometry. To be more realistic 3D simulations have to be considered. Most parts of the software tool developed in this thesis are designed for both 2D and 3D experiment except that the component responsible for new grid creation can only handle 2D case. The 3D simulations could be supported by replacing this component or making some changes on the source code of the program.

2. VTK visualization. Currently the visualization is done with support from other third party programs as Gnuplot and plotmtv. They visualize the data and store them in GIF format. The GUI component loads the GIF file and display it. This is acceptable for 2D cases. The fixed viewpoint makes this type of solution very inconvenient for 3 dimensional cases. It’s better to generate the visualization directly on the GUI so that the user could have full interaction with visualization. As described in [33] Tk interface for vtk allows sophisticated visualization on Tk GUIs. There exists Tk based visualization program named “vtkviz” to visualized data in VTK format in a single Tk widget. Because it is written in tcl and tkinter has a special function called $tk()$ which can execute given tcl command, with some efforts “vtkviz” could be integrated into the program written for this thesis. Of course the orignial GIF scheme can be kept as an alternative.

3. Incorporation of the lung. The lung is not included in simulations done in this thesis. The conductivity of tissue between heart and torso is set to be isometric. In future the characteristic of lung can be incorporated and experiments on the inter-individual variation in those can be conducted.

4. Experimentation on the inter-individual variation in the conductivities of the ischemia region. In the experiments done so far, the only
difference between the ischemia region and the healthy region is the conductivity. The conductivity of the ischemic tissue may also vary in a certain range. The variation certainly gives the impact on the body surface potentials.

5. Conduction of the experiment with all the variations combined together. In this thesis each single experiment is conducted with the variation only in one kind of heart characteristics. This is not realistic. In future it could be done with all kinds of possible variations combined in an experiment. The challenge is to find a way to qualify and quantify such combination of the variations, and to interpret the experiment results properly.
Appendix A

Heart and Torso Geometry Data

The heart geometry is given in form of 4 matrices representing the contour of torso, heart, left ventricle, right ventricle on the transverse plane, respectively.

A.1 Torso Contour

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.08890571e+01</td>
<td>1.89983609e+01</td>
</tr>
<tr>
<td>9.25461218e+00</td>
<td>1.92279343e+01</td>
</tr>
<tr>
<td>7.61726875e+00</td>
<td>1.90576832e+01</td>
</tr>
<tr>
<td>6.02431178e+00</td>
<td>1.86322779e+01</td>
</tr>
<tr>
<td>4.51269471e+00</td>
<td>1.79485225e+01</td>
</tr>
<tr>
<td>3.25007377e+00</td>
<td>1.69143636e+01</td>
</tr>
<tr>
<td>2.32581169e+00</td>
<td>1.55443902e+01</td>
</tr>
<tr>
<td>1.71360631e+00</td>
<td>1.40013030e+01</td>
</tr>
<tr>
<td>1.46489153e+00</td>
<td>1.23809216e+01</td>
</tr>
<tr>
<td>1.51452460e+00</td>
<td>1.07380772e+01</td>
</tr>
<tr>
<td>1.41158411e+00</td>
<td>9.10438148e+00</td>
</tr>
<tr>
<td>1.23242008e+00</td>
<td>7.46740384e+00</td>
</tr>
<tr>
<td>1.12278394e+00</td>
<td>5.82302913e+00</td>
</tr>
<tr>
<td>6.05322900e-01</td>
<td>4.24175236e+00</td>
</tr>
</tbody>
</table>
5.42996900e-02, 2.73693722e+00,
8.97592410e-01, 1.33183069e+00,
2.41499278e+00, 4.55177140e-01,
4.03326747e+00, 1.66930010e-01,
5.68578135e+00, 1.53627840e-01,
7.33277902e+00, 1.58984400e-01,
8.97264931e+00, 1.41578540e-01,
1.06112876e+01, 1.02273280e-01,
1.22537934e+01, 4.51090900e-02,
1.38984533e+01, 1.32902000e-03,
1.55400936e+01, 1.59927400e-02,
1.71740374e+01, 1.29800710e-01,
1.88070289e+01, 2.86136870e-01,
2.04567775e+01, 3.34701640e-01,
2.21201087e+01, 2.55833250e-01,
2.37472927e+01, 3.23340780e-01,
2.52904636e+01, 8.16699250e-01,
2.67244447e+01, 1.78710176e+00,
2.76786939e+01, 3.12252292e+00,
2.76709613e+01, 4.66722024e+00,
2.70387983e+01, 6.28567874e+00,
2.63935990e+01, 7.87238055e+00,
2.60367008e+01, 9.44990371e+00,
2.60955829e+01, 1.10775533e+01,
2.61384898e+01, 1.27200412e+01,
2.56451042e+01, 1.42885677e+01,
2.48116248e+01, 1.57054188e+01,
2.37752953e+01, 1.69895985e+01,
2.24901126e+01, 1.79931291e+01,
2.09789038e+01, 1.86670231e+01,
1.93829471e+01, 1.91327758e+01,
1.77661508e+01, 1.92595382e+01,
1.61578401e+01, 1.89616104e+01,
1.45526431e+01, 1.83672370e+01,
## A.2 Heart Contour

<table>
<thead>
<tr>
<th>x-coordinates</th>
<th>y-coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.66221466</td>
<td>14.94340146</td>
</tr>
<tr>
<td>15.2155008</td>
<td>15.28879736</td>
</tr>
<tr>
<td>14.71078103</td>
<td>15.58139515</td>
</tr>
<tr>
<td>14.18555149</td>
<td>15.85376075</td>
</tr>
<tr>
<td>13.64519397</td>
<td>16.09017893</td>
</tr>
<tr>
<td>13.0943778</td>
<td>16.27386328</td>
</tr>
<tr>
<td>12.5377226</td>
<td>16.38802738</td>
</tr>
<tr>
<td>11.9800466</td>
<td>16.41588485</td>
</tr>
<tr>
<td>11.42599947</td>
<td>16.34086103</td>
</tr>
<tr>
<td>10.88875438</td>
<td>16.16003073</td>
</tr>
<tr>
<td>10.3893849</td>
<td>15.89497565</td>
</tr>
<tr>
<td>9.93037521</td>
<td>15.57484882</td>
</tr>
<tr>
<td>9.50539764</td>
<td>15.2106377</td>
</tr>
<tr>
<td>9.10684779</td>
<td>14.80732751</td>
</tr>
<tr>
<td>8.72710751</td>
<td>14.36990265</td>
</tr>
<tr>
<td>8.36361958</td>
<td>13.90293007</td>
</tr>
<tr>
<td>8.05561756</td>
<td>13.40752971</td>
</tr>
<tr>
<td>7.86293177</td>
<td>12.88312262</td>
</tr>
<tr>
<td>7.84016555</td>
<td>12.32999158</td>
</tr>
<tr>
<td>7.95389043</td>
<td>11.76331729</td>
</tr>
<tr>
<td>8.1558297</td>
<td>11.21548906</td>
</tr>
<tr>
<td>8.45944614</td>
<td>10.72455175</td>
</tr>
<tr>
<td>8.88094745</td>
<td>10.32880408</td>
</tr>
<tr>
<td>9.42262545</td>
<td>10.0785318</td>
</tr>
<tr>
<td>9.7253842</td>
<td>10.12047801</td>
</tr>
<tr>
<td>10.46897597</td>
<td>10.39594973</td>
</tr>
<tr>
<td>11.04509276</td>
<td>10.42848868</td>
</tr>
<tr>
<td>11.5748408</td>
<td>10.56480806</td>
</tr>
</tbody>
</table>
A.3. RIGHT VENTRICLE CONTOUR

11.9866542 , 10.99436665,
12.51058355, 11.06467821,
12.962055, 11.3981397,
13.01002293, 11.06467821,
13.3528645, 12.2381417,
13.91003467, 12.45822298,
14.56509476, 12.65242657,
15.20160605, 12.9047449,
15.7051639, 13.2871651,
15.9962685, 13.78632472,
16.00471106, 14.34050387

A.3 Right Ventricle Contour

15.00211192, 14.590916 ,
14.36923245, 14.75074638,
13.6525871, 14.79960249,
13.10851811, 15.17652463,
12.63823115, 15.63993625,
11.94597444, 15.61182166,
11.36473673, 15.86598316,
10.73104199, 15.8088903 ,
10.41304428, 15.36893281,
11.17079592, 15.21611057,
11.79223638, 15.26732399,
12.29197323, 14.80201076,
12.55828486, 14.23850894,
12.76588881, 13.6173237 ,
13.10986077, 12.94628685,
13.62323782, 12.60666439,
14.10582066, 13.09823764,
14.77263434, 13.2331167 ,
15.23479809, 13.70927034
A.4 Left Ventricle Contour

11.49838337, 13.60406697,
11.20668917, 13.92466723,
10.77810761, 14.13066750,
10.32689827, 14.16412235,
9.97083174, 13.96759439,
9.75963486, 13.61001444,
9.57470302, 13.24125267,
9.25836410, 12.76437253,
8.98067237, 12.30907212,
9.03993874, 12.10539906,
9.45950315, 12.11565129,
9.58218870, 11.70067552,
9.87845073, 11.54028110,
10.33052255, 11.61231890,
10.73484326, 11.63379534,
11.18033320, 11.72803911,
11.54718122, 11.95316048,
11.66099475, 12.33134126,
11.66122657, 12.77503702,
11.62969202, 13.20776378,
Appendix B

Result of Experiments with Transmural Ischemia

B.1 Ischemia Region

The center point of ischemia region is at 9, 13.99, the semi-diameter is 1.3cm. The actual position of transmural ischemia is showed in Fig B.1(a).

Figure B.1: Tranmural Ischemia and Points Order

B.2 Experiment Results
APPENDIX B. RESULT OF EXPERIMENTS WITH TRANSMURAL ISCHEMIA

(a) The relation between the deviations and the increments in $\sigma_T$

(b) The comparison of the body surface potentials from the simulations with different increments in $\sigma_T$

(c) The relation between the deviations and the increments in $\sigma_L$

(d) The comparison of the body surface potentials from the simulations with different increments in $\sigma_L$

(e) The relation between the deviations and the increments in $\sigma_T$

(f) The comparison of the body surface potentials from the simulations with different increments in $\sigma_T$

Figure B.2: Result of experiment with Transmural Ischemia
B.2. EXPERIMENT RESULTS

(a) The relation between the deviations and the increments in $\sigma_i^L$

(b) The comparison of the body surface potentials from the simulations with different increments in $\sigma_i^L$

(c) The relation between the deviations and the clockwise rotation angles in the fiber orientation.

(d) The comparison of body surface potentials from simulations with the clockwise rotations in fiber orientation.

(e) The relation between the deviations and the counter-clockwise rotation angles in the fiber orientation.

(f) The comparison of body surface potentials from simulations with the counter-clockwise rotations in fiber orientation.

Figure B.3: Continue: result of experiment with transmural ischemia
APPENDIX B. RESULT OF EXPERIMENTS WITH TRANSMURAL ISCHEMIA

(a) The relation between the deviations and the rightward displacements of the heart.

(b) The comparison of the body surface potentials from simulations with the heart rightward displaced.

(c) The relation between the deviations and the leftward displacements of the heart.

(d) The comparison of the body surface potentials from simulations with the heart leftward displaced.

(e) The relation between the deviations and the forward displacements of the heart.

(f) The comparison of the body surface potentials from simulations with the heart forward displaced.

Figure B.4: Continue: result of experiment with transmural ischemia
B.2. EXPERIMENT RESULTS

(a) The relation between the deviations and the backward displacements of the heart.

(b) The comparison of the body surface potentials from simulations with the heart backward displaced.

(c) The relation between the deviations and the clockwise rotations of the heart.

(d) The comparison of the body surface potentials from simulations with the heart clockwise rotated.

(e) The relation between the deviation and the counter-clockwise rotations of the heart.

(f) The comparison of the body surface potentials from simulations with the heart counter-clockwise rotated.

Figure B.5: Continue: result of experiment with transmural ischemia
Appendix C

Result of Experiments with Transmural Ischemia-2

C.1 Ischemia Region

The center point of ischemia region is at 10.5, 11, the semi-diameter is 1.3 cm. The actual position of transmural ischemia is showed in Fig C.1(a).

Figure C.1: Tranmural Ischemia2 and Points Order2

(a) Tranmural ischemia. The dark area indicates the ischemia region.
(b) Order of torso boundary points. The points are numbered clockwise.

C.2 Experiment Results
C.2. EXPERIMENT RESULTS

(a) The relation between the deviations and the increments in $\sigma_e^T$

(b) The comparison of the body surface potentials from the simulations with different increments in $\sigma_e^T$

(c) The relation between the deviations and the increments in $\sigma_e^L$

(d) The comparison of the body surface potentials from the simulations with different increments in $\sigma_e^L$

(e) The relation between the deviations and the increments in $\sigma_i^T$

(f) The comparison of the body surface potentials from the simulations with different increments in $\sigma_i^T$

Figure C.2: Result of experiment with Transmural Ischemia
APPENDIX C. RESULT OF EXPERIMENTS WITH TRANSMURAL ISCHEMIA-2

(a) The relation between the deviations and the increments in $\sigma_i^L$.

(b) The comparison of the body surface potentials from the simulations with different increments in $\sigma_i^L$.

(c) The relation between the deviations and the clockwise rotation angles in the fiber orientation.

(d) The comparison of body surface potentials from simulations with the clockwise rotations in fiber orientation.

(e) The relation between the deviations and the counter-clockwise rotation angles in the fiber orientation.

(f) The comparison of body surface potentials from simulations with the counter-clockwise rotations in fiber orientation.

Figure C.3: Continue: result of experiment with transmural ischemia
C.2. EXPERIMENT RESULTS

(a) The relation between the deviations and the rightward displacements of the heart.

(b) The comparison of the body surface potentials from simulations with the heart rightward displaced.

(c) The relation between the deviations and the leftward displacements of the heart.

(d) The comparison of the body surface potentials from simulations with the heart leftward displaced.

(e) The relation between the deviations and the forward displacements of the heart.

(f) The comparison of the body surface potentials from simulations with the heart forward displaced.

Figure C.4: Continue: result of experiment with transmural ischemia
APPENDIX C. RESULT OF EXPERIMENTS WITH TRANSMURAL ISCHEMIA-2

(a) The relation between the deviations and the backward displacements of the heart.

(b) The comparison of the body surface potentials from simulations with the heart backward displaced.

(c) The relation between the deviations and the clockwise rotations of the heart.

(d) The comparison of the body surface potentials from simulations with the heart clockwisely rotated.

(e) The relation between the deviation and the counter-clockwise rotations of the heart.

(f) The comparison of the body surface potentials from simulations with the heart counter-clockwisely rotated.

Figure C.5: Continue: result of experiment with transmural ischemia
Appendix D

Screenshots and API Documentation

D.1 Screenshots

Figure D.1, D.2, D.3, D.4 and D.5 show the screenshots taken from the running program which is developed in this thesis to automate the conduction of the experiments. For the program itself, please see appended CD or download it from http://folk.uio.no/zhiguanj/experimenter.tar.gz
Figure D.1: One experiment has added

Figure D.2: Configuring added experiment.
Figure D.3: Configuring added simulator.
Figure D.4: Configuring the variation in heart geometry.
Figure D.5: After all the experiments are done.
D.2 API Documentation
This class is meant to display the exception information on the graphical user interface. It is used to replace the original object associated with systematics.

### Methods

1. **__init__(self, message)***
   
   This method initializes the exception information on the graphical user interface.

2. **__call__(self)***
   
   The method calls the method that is associated with systematics.

### Properties

- **message*** (str)
  
  The message that is displayed on the graphical user interface.
Module analyzer

2 Module analyzer

This file is written for master thesis, by jia zhiguang.

2.1 Class analyzer

Object

Module analyzer

2.1.1 Methods

def print(self, benchmar, nonodes):    # initializes list of benchmarks
    benchmark: other values

    sim: the dictionary will be used in returned function.

    Returns
    value of relative error of two operators

    self, benchmar

    def helper (self, value, benchmark)
        This function computes the VALUE of relative error "e2". The value is
        computed by taking the ratio of sum over absolut value of (value-benchmar)
        and frobenius norm of benchmark.
        Parameters
        value: type=list or nested list float
        benchmark: reference value
        Returns
        value of relative error of two operators

    def cal get(self, value, benchmark)
        This function computes the VALUE of relative error 'deviation'. The value is
        computed by taking the ratio of frobenius norm of benchmark
        and frobenius norm of (values2-benchmar) and Frobenius norm of benchmark
        Parameters
        value: type=list or nested list float
        benchmark: inference value
        Returns
        value of deviation

    def size_get(self) 
        self, vector, return the p-vector norm

    def norm(self, a)  
        If a is a vector, then this function return the p-vector norm
        Otherwise, return frobenius norm
        Parameters
        a: array
        Returns
        frobenius norm of benchmark

    def get norm(self, a)
        If a is a matrix , then this function return the frobenius norm if a is
        a list or a matrix
        Parameters
        a: array
        Returns
        frobenius norm of benchmark

    def getSize(self)
        This function return the size of benchmark
        Parameters
        Returns
        size

    def getMaxMin(self, data1, data2)
        Find out max value of data1, and min value of it
        Parameters
        data1: finding out max value of it
        data2: finding out min value of it
        Returns
        max of data1, min of data2

    def visualize (self, data, gnuplot command, and name of file which will store the result of visualization)
        Parameters
        data: data needed to perform visualization
        gnuplot command, and name of file which will store the result of visualization
        Returns
        a list of [deviation, perturbation]

    def sorting(points, benchmark)
        Sorting the points on the contour according there natural positions.
        Parameters
        points: list of points
        benchmark: the point which is counter clockwisely closest to this
        Returns
        sorted points

This function return the size of benchmark
Parameters
Returns
size

APPENDIX D. SCREENSHOTS AND API DOCUMENTATION
3.1.1 Methods

Methods are functions bound to the configurations of the dialog.

- `self.execute(event)`: executes the configuration panel.
- `self.init(name, event)`: initializes the configuration panel.

Inherited from: `object`

Parameters:
- `name`: the name of the experiment.
- `event`: the tkinter event.

Return Value:
- This function returns the configuration panel.

3.1.2 Inheritance

Inherited from: `object`

3.2.1 Methods

Methods are functions bound to the configuration of the dialog.

- `self.call(name, event)`: calls the configuration panel.
- `self.init(name, event)`: initializes the configuration panel.

Inherited from: `object`

Parameters:
- `name`: the name of the experiment.
- `event`: the tkinter event.

Return Value:
- This function returns the configuration panel.

3.2.2 Inheritance

Inherited from: `object`

4.1.1 Methods

Methods are functions bound to the configuration of the dialog.

- `self.call(name, event)`: calls the configuration panel.
- `self.init(name, event)`: initializes the configuration panel.

Inherited from: `object`

Parameters:
- `name`: the name of the experiment.
- `event`: the tkinter event.

Return Value:
- This function returns the configuration panel.

4.1.2 Inheritance

Inherited from: `object`

5.1.1 Methods

Methods are functions bound to the configuration of the dialog.

- `self.call(name, event)`: calls the configuration panel.
- `self.init(name, event)`: initializes the configuration panel.

Inherited from: `object`

Parameters:
- `name`: the name of the experiment.
- `event`: the tkinter event.

Return Value:
- This function returns the configuration panel.

5.1.2 Inheritance

Inherited from: `object`
3.1.2 Properties

13

3.2.1 Methods

14

13

14

15

16
Inherited from Tkinter.Mix

Parameters
- `command`: this will be printed out to stdout

Return Value
- `None`

API DOCUMENTATION

119

Class experiment interface Module experiment

4.1 Variables

Parameter
- `cmd`: command to execute

Return Value
- `None`

4.2 Class experiment

This class controls the processing flow of the whole program. It creates a GUI interface with
help from interface py, and then pass the program control to the interface. The program
because control-driven. The GUI will use the experiment name and do corresponding
job according GUI actions.

It has three functionals:
1. create and delete experiments
2. build up a GUI interface
3. export the current work or import one

This file is written for master thesis, by jia zhiguang
May, 2008, structure changed
2008, Jan, 16 revised 2008, Feb 6, several new functions added, 17, 18, 19
2007, 9, nov changed to class style
2007, 11, set. The first version of this module is too long, going to compose it by utilizing
structured programming technique.

D.2. API DOCUMENTATION

4.3 Class experiment

This file is written for master thesis, by jia zhiguang
May, 2008, structure changed
2008, 9, revised 2008, 11, 31 revised 2008, Feb 6, several new function added, 17, 18, 19
May, 2008, structure changed

3.3.2 Class Variables

Name
- `cmd` Description
- `notebook`
## Class grid maker Module grid maker

**polygons2GridFE** (self, filename, para, index =None, max area=None)

Making finite element grid from polygon file

- **Parameters**
  - filename: name of polygon file (type=string)
  - para: parameters, index=None, max_area=None

### Return Value

None

**writeTrianglePoly** (self, filename, Para, index=None)

Prepare data for calling triangling program

- **Parameters**
  - filename: name of polygon file (type=string)
  - Para: parameters, index=None

### Return Value

None

**writeT3n2D** (self, name, N, E)

Parameters

- name: name of grid file (type=string)
- N: come later
- E: come later

### Return Value

None

**make** (self, write filename=None, data=None)

Make grid from given data

- **Parameters**
  - write filename: name of grid file (type=string)
  - data: data from which grid will be created.

### Return Value

The name of created grid file

---

### 5.1.2 Properties

**Name**

**Description**

Inherited from object
class

---

**GIF INFO CLASS Module header**

This class is a data class, acting like C/C++ header file. It holds some general specifications needed by other classes, for example size of the GIF file.

NB: When importing this class, it can be either imported by statement 'from header import header' or 'from header import GIF INFO'. By former case, the 'GIF INFO' class is imported as header.

### 6.2 Class GIF INFO CLASS

**init** (self)

- **init**(...) initializes x; see x.
class for signature

**Overrides:** object

**setattr** (self, name, value)

- x.setattr('name', value) <== x.name = value

**Overrides:** object

---

### Inherited from object
delattr(), getattribute(), hash(), new(), reduce(), reduce ex(), repr(), str()
Inherited from Tkinter.Misc

Inherited from Tkinter.Tk

Class interface Module interface

Class myTk Module interface

7.2 Class interface

Interface interface

This is the gui interface for program.

7.2.1 Methods

This function specify the action taken when the single click on experiment

Parameters

experiment : the experiment click.

Return Value

None

7.2.2 Class Variables

Inherited from Tkinter.Misc

noarg

Inherited from Tkinter.Tk

7.3 Properties

Name Description

None

Inherited from object

This is the gui interface for program.

7.3.1 Methods

The function build up the experiment managing part of GUI, for example the

Parameters

container : instance of class experiment container.

Return Value

None

Inherited from Tkinter.Misc

Noarg

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

This function will raise a file dialog to let user to specify a file which will be

Parameters

file : the name of file.

Return Value

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None

Inherited from Tkinter.Tk

None

Inherited from Tkinter.Misc

None
8 Module make_vec_perturbation

This file is written for master thesis, by jia zhiguang 2007, Oct, 19-2007, Oct, 30, done with
first version of this class
2008, Jan 21, edit: epydoc type of doc string

8.1 Class make_vec_perturbation

Para

This class is meant to be as the parent class for others, and its inherited from both tem-
plate_perturbation and vec. It has some functions defined for matrix affine transformation
operations - can work smoothly for both 2D and 3D files

8.1.1 Methods

create affine matrix(self, para)
create the affine ... from template perturber.template perturber (Section 17.1)
name code, name flag, name string, perturb type, which one

Parameters

Para:

Return Value

8.1.2 Properties

Inherited from template_perturbation_template_gauze_perturbation (Section 17.1)

Inherited from object

8.1.3 Class Variables

Inherited from object

9 Module make_documentation

Making documentation for the program

9.1 Class make_documentation

Object

Para

Inherited from object

9.1.2 Properties

Inherited from object
This module includes one class

10.1 Class make_extraconds_perturbation

object

template = make_extraconds_perturbation.make_extraconds_perturbation

template = make_extraconds_perturbation.make_extraconds_perturbation

template = make_extraconds_perturbation.make_extraconds_perturbation

This class is made for doing perturbation on extracellular conductivity inherited from two classes: template = make_extraconds_perturbation and visualize.

10.1.1 Methods

- **pseudo_random_perturb(index, data)**
  - **Parameters**
    - index: The method index. 0 for method "uniform division", 1 for method "pseudo random perturb".
    - data: The original data.
  - **Return Value**
    - unperturbed data.

- **uniform_perturb(paras, data)**
  - **Parameters**
    - paras: A list containing the parameters for perturbation.
    - data: The original data.
  - **Return Value**
    - The perturbed data.

- **visualize**
  - **Parameters**
    - data: The original data.
  - **Return Value**
    - Visualized data.

- **write_to_file(self, data, path=None)**
  - **Parameters**
    - data: The data to be written to file.
    - path: The path to the file.
  - **Return Value**
    - The file path.

- **write_back(self, data, path)***
  - **Parameters**
    - data: The data to be overwritten.
    - path: The path to the file.
  - **Return Value**
    - The file path.

This method sets up a random perturbator, configures the perturbation parameters, and visualizes the extracellular conductivity.
11 Module make_field_perturbation

class make_field_perturbation

the file is written for master thesis, by jia zhiguang 2007, Oct, 16.

11.1 Class make_field_perturbation

object java

visualization template 'make_field_perturbation'

make_field_perturbation make_field_perturbation

This class is responsible to make perturbation on heart fiber direction field. It's also responsible for visualizing the fiber orientation.

11.1.3 Methods

get_visualization(self, data, name)

Get visualization of unperturbed data.

Parameters:

- data: data needs to be visualized.
- name: the name of created 'gif' file

Return Value:

- the name of created 'gif' file

11.1.4 Instance Variables

perturbed name of conductivity

41
## Class make_field_perturbation

### Method create_canvas(self, parent, call back)

This function creates a configuration window for a perturbation experiment. It contains the option to set trigger, then make, then translate, then rotate. The configuration window is created by a class. Parameters:
- `parent`: the parent window of the configuration window
- `call back`: a function that will be called when the configuration window is closed

### Method create_affine_matrix(self, para)

Create an affine transformation matrix. The matrix is created by applying the function indicated by method `call back` to the given `para`.

#### Parameters
- `para`: the perturbation parameters used to create the affine transformation

#### Return Value
- `None`

### Method create_visu_data(self, data, what type, which one, para)

Create visualization data for a perturbation experiment. The function `call back` is used to compute the visualization data.

#### Parameters
- `data`: data to be visualized
- `what type`: type of visualization
- `which one`: which type of visualization
- `para`: the perturbation parameters used to create the visualization

#### Return Value
- `None`

### Method vector_re_calculate(self, which one, perturb para, apply to)

Recalculate vectors for perturbation. The function `call back` is used to compute the vectors.

#### Parameters
- `which one`: which type of perturbation
- `perturb para`: perturbation parameters used to create the perturbation
- `apply to`: the perturbation parameters used to apply the perturbation

#### Return Value
- `None`

### Method visualize_data(self, data)

Visualize the data for a perturbation experiment. The function `call back` is used to compute the visualization data.

#### Parameters
- `data`: data to be visualized

#### Return Value
- `None`
### 11.2 Module make field perturbation

This file is written for master thesis, by Jia Zhiguang 2007, Oct. 16.

<table>
<thead>
<tr>
<th>Class dialog Module make field perturbation</th>
</tr>
</thead>
<tbody>
<tr>
<td>delattr (), getattribute (), hash (), new (), reduce (), reduce ex (), repr</td>
</tr>
<tr>
<td>This function is bound to configuration dialog, activate different method when configure widget is pressed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>str (for doing an experiment)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>

### 12.1.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>para:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=list of float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>list is a dictionary type instance. It contains all information needed</td>
</tr>
</tbody>
</table>

### 12.1.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>angles:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=list of float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
</tr>
</tbody>
</table>

### 12.1.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>create scaling matrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>self, para</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
</tr>
</tbody>
</table>

### 11.3.4 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=list of float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
</tr>
</tbody>
</table>

### 12.3 Module make field perturbation

12.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=list of float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
</tr>
</tbody>
</table>

### 11.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=list of float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
</tr>
</tbody>
</table>

### 12.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>type=list of float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
</tr>
</tbody>
</table>
12.2 Class: make heart_perturbation

```python
class make heart perturbation:
    def __init__(self, filename: str, paras: dict) -> None:
        self.filename = filename
        self.paras = paras

    def perturb(self, vertexes: list) -> None:
        # Perturbation process
        pass

    def read(self) -> None:
        # Reading the ischemia region information
        pass
```

### Methods

- `perturb(vertexes)`: Perturbation function.
- `read()`: Reading the ischemia region information.

### Parameters

- `filename`: The name of the file containing the ischemia region information.
- `paras`: Perturbing parameters for an experiment.

### Returns

- `None`

---

12.2.1 Methods

#### `perturb(vertexes)`

```python
def perturb(self, vertexes: list) -> None:
    # ... perturbation process...
    pass
```

#### `read()`

```python
def read(self) -> None:
    # Reading the ischemia region information
    pass
```

### Parameters

- `vertexes`: A list of vertexes.

### Returns

- `None`
### D.2. API DOCUMENTATION

#### 12.3 Class dialog

**Module** make heart perturbation
dialog

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>init (self, parent, pro model=None)</td>
<td>parent: Up ... 'Apply'</td>
<td></td>
</tr>
<tr>
<td>dummy func(self, data)</td>
<td>event: The tkinter event</td>
<td>None</td>
</tr>
</tbody>
</table>

**Properties**

- **parent:** Up to the configuration widget

**Methods**

- **array to list(self, data):** This function is bound to configuration dialog, activate different method when outcome OK/ Discard / Apply
- **get unperturbed visu(self, name):**
- **perturb**
- **write to file:**
- **visualize data , and save it in gif format with transparency**

### 13 Module make intraconds perturbation

**Class** make intraconds perturbation

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>init (self, parent, pro model=None)</td>
<td>parent: Up ... 'Apply'</td>
<td></td>
</tr>
<tr>
<td>dummy func(self, data)</td>
<td>event: The tkinter event</td>
<td>None</td>
</tr>
</tbody>
</table>

**Properties**

- **parent:** Up to the configuration widget

**Methods**

- **array to list(self, data):**
- **get unperturbed visu(self, name):**
- **perturb**
- **write to file:**
- **visualize data , and save it in gif format with transparency**

---

**Inherited from:**

- model make perturbation
- model make perturbation(Section 8.1)
- model make perturbation(Section 8.1)

**Inherited from object:**

- object
- object

---

** Attributes**

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>init (self, parent, pro model=None)</td>
<td>parent: Up ... 'Apply'</td>
<td></td>
</tr>
<tr>
<td>dummy func(self, data)</td>
<td>event: The tkinter event</td>
<td>None</td>
</tr>
</tbody>
</table>

**Properties**

- **parent:** Up to the configuration widget

**Methods**

- **array to list(self, data):**
- **get unperturbed visu(self, name):**
- **perturb**
- **write to file:**
- **visualize data , and save it in gif format with transparency**

---

**Inherited from:**

- model make perturbation
- model make perturbation(Section 8.1)
- model make perturbation(Section 8.1)

**Inherited from object:**

- object
- object

---

** Attributes**

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>init (self, parent, pro model=None)</td>
<td>parent: Up ... 'Apply'</td>
<td></td>
</tr>
<tr>
<td>dummy func(self, data)</td>
<td>event: The tkinter event</td>
<td>None</td>
</tr>
</tbody>
</table>

**Properties**

- **parent:** Up to the configuration widget

**Methods**

- **array to list(self, data):**
- **get unperturbed visu(self, name):**
- **perturb**
- **write to file:**
- **visualize data , and save it in gif format with transparency**

---

**Inherited from:**

- model make perturbation
- model make perturbation(Section 8.1)
- model make perturbation(Section 8.1)

**Inherited from object:**

- object
- object
### Class `make_intraconds_perturbation` Module `make_intraconds_perturbation`

#### Inherited from object

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__abstract__</code></td>
<td>Abstract class</td>
</tr>
</tbody>
</table>

#### 13.2.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__abstract__</code></td>
<td>Abstract class</td>
</tr>
</tbody>
</table>

#### 13.2.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>value</code></td>
<td>Parameter which the module could understand</td>
</tr>
</tbody>
</table>

#### 13.2 Class dialog

**configuration panels container**

**perturbation**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>value</code></td>
<td>Parameter which the module could understand</td>
</tr>
</tbody>
</table>

#### 13.2.4 Methods

- `__call__(self, para):`
  - **Parameters**
    - `para`: Parameters for one experiment
  - **Return Value**
    - Return value

- `__init__(self, parent, para=None):`
  - **Parameters**
    - `parent`: Up level container
  - **Overrides**
    - `AbstractClass.__init__(self, parent, para=None)`
  - **Return Value**
    - The index of direction of conductivity

- `build_interior(self):`
  - **Parameters**
    - `which`: The index of direction of conductivity
  - **Overrides**
    - `AbstractClass.build_interior(self, which):`
  - **Return Value**
    - Return value

- `get_config_data(self):`
  - **Parameters**
    - `event`: Tkinter event
  - **Overrides**
    - `AbstractClass.get_config_data(self, event):`
  - **Return Value**
    - Return value

- `getperturbation_data(self, para):`
  - **Parameters**
    - `para`: Perturbation parameter which the module could understand
  - **Return Value**
    - Return value

- `initialize_perturbation(self):`
  - **Parameters**
    - `para`: Perturbation parameter which the module could understand
  - **Return Value**
    - Return value

- `pseudo_random_perturb(self, which, one, para=None):`
  - **Parameters**
    - `which`: The index of direction of conductivity
    - `one`: The index of direction of conductivity
    - `para`: Parameters for one experiment
  - **Return Value**
    - Return value

- `setperturbed_data(self, data):`
  - **Parameters**
    - `data`: Data needs to be visualized
  - **Overrides**
    - `AbstractClass.setperturbed_data(self, data):`
  - **Return Value**
    - Return value

- `visualize_perturbed_data(self, para=None):`
  - **Parameters**
    - `para`: Perturbation parameter which the module could understand
  - **Overrides**
    - `AbstractClass.visualize_perturbed_data(self, para=None):`
  - **Return Value**
    - Return value
14.2.1 Methods

\texttt{init}(self, parent, \texttt{pro\_model}=
\texttt{None})

**Parameters**

- **parent**: Up level container

\texttt{build\_interior}(self)

Build up the interior

\texttt{execute}(self, \texttt{event})

This function is bound to configuration dialog, indicating the actions activated when following buttons are pressed: 'Ok' 'Discard' 'Apply'

**Parameters**

- **event**: The tkinter event

15.1.1 Methods

\texttt{call}(self, \texttt{casename}, \texttt{perturbations}, \texttt{sim\_res})

The method of abstract class, if called, AbstractClassException is raised. This function is meant to be called by others when needed. Function itself should know the name of file which the data will be written into, the outside caller will decide where to put this output file.

**Parameters**

- **casename**: experiment name
- **perturbations**: perturbation parameters and perturbed data
- **sim\_res**: simulation results

\texttt{build\_interface}(self, \texttt{parent})

The method of abstract class, if called, AbstractClassException is raised. This function takes care of creating a experiment configuration panel, which is the child of the Tkinter instance specified by 'parent'.

**Parameters**

- **parent**: root widget for gui instance

**Return Value**

created GUI instance

Inherited from \texttt{visualize\_visualize(Section 19.1)}

\texttt{visualize()}

15.1.2 Properties

**Name** | **Description**
--- | ---

Inherited from \texttt{object}

16.1.1 Methods

\texttt{call}(self, \texttt{casename}, \texttt{perturbations}, \texttt{sim\_res})

The method of abstract class, if called, AbstractClassException is raised. This function is meant to be called by others when needed. Function itself should know the name of file which the data will be written into, the outside caller will decide where to put this output file.

**Parameters**

- **casename**: experiment name
- **perturbations**: perturbation parameters and perturbed data
- **sim\_res**: simulation results

\texttt{build\_interface}(self, \texttt{parent})

The method of abstract class, if called, AbstractClassException is raised. This function takes care of creating a experiment configuration panel, which is the child of the Tkinter instance specified by 'parent'.

**Parameters**

- **parent**: root widget for gui instance

**Return Value**

created GUI instance

Inherited from \texttt{visualize\_visualize(Section 19.1)}

\texttt{visualize()}

16.1.2 Properties

**Name** | **Description**
--- | ---

Inherited from \texttt{object}

Inherited from \texttt{object}

\texttt{delattr()}, \texttt{getattribute()}, \texttt{hash()}, \texttt{init()}, \texttt{new()}, \texttt{reduce()}, \texttt{reduce\_ex()}, \texttt{repr()}, \texttt{setattr()}, \texttt{str}()
17.1 Class template:perturber

Abstract super class for all other perturbing class. It means to be as a kind of interface which every perturbation making class has to implement in order to make module can Plug and Play.

Besides the functions defined here which must be re-implemented in different classes, there are 3 class variables which must be re-implemented as well:

- name
- code
- flag

17.1.1 Methods

- __init__(self)
  - x.__init__(...) initializes x; see x.__class__.__doc__ for signature
  - overrides: object.__init__(inherited documentation)

- write_to_file(self, data, path=None)
  - The method of abstract class, if called, AbstractClassException is raised.
  - This function is meant to be called by others when needed.
  - Function itself should know the name of file which the data will be written into, the outside caller will decide where to put this output file.

  Parameters
  - data: object which needs to be written into file
  - path: path of output file. Default value is None

- call(self, perturb_pars)
  - The method of abstract class, if called, AbstractClassException is raised.
  - This function does perturbation according to the argument it got. Usually, the argument is what the child got from configure canvas, or it can be manually created parameter which the function can understand.

  Parameters
  - perturb_pars: perturbation parameter which the module could understand

- Return Value
  - perturbed data

- config_canvas(self, parent, callback)
  - The method of abstract class, if called, AbstractClassException is raised.
  - This function takes care of creating an experiment configuration panel, which is a child of the Tkinter instance specified by "parent".

  Parameters
  - parent: the uplevel Tkinter instance (type=any kind of Tkinter container)
  - callback: the function need to be called inside this function

- Return Value
  - Nothing

Inherited from visualize.visualize(Section 19.1)

- visualize()

Inherited from object

- getattr(), setattr(), delattr(), hash(), init(), new(), reduce(), reduce_ex(), repr(), str()
### 18.1 Class template simulator

Object visualize.template.simulator
class for simulator. It has been generalized to run all different programs in batch style. It has
functions to compile, visualize simulation results.

#### 18.1.1 Methods

- **prepare**
  - **Function**
    - `prepare(self)`
    - Copies the file needed by simulation to the current directory
    - The method of abstract class, if called, AbstractClassException is raised.
    - This function is meant to be called by others when needed.
    - Function itself should know the name of file which the data will be written into, the outside caller will decide where to put this output file.

  **Parameters**
  - `casename`: experiment name
  - `perturbations`: perturbation parameters and perturbed data
  - `simres`: simulation results

- **call**
  - **Function**
    - `call(self, cmd)`
    - Do the actual simulations.
    - The method of abstract class, if called, AbstractClassException is raised.
    - This function is meant to be called by others when needed.
    - Function itself should know the name of file which the data will be written into, the outside caller will decide where to put this output file.

  **Parameters**
  - `casename`: experiment name
  - `perturbations`: perturbation parameters and perturbed data
  - `simres`: simulation results

#### 18.1.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inherited from object</td>
</tr>
</tbody>
</table>
Bibliography


[41] Robert S. MacLeod, Quan Ni, Bonnie Punske, Philip R. Ershler, Bu-
lent Yilmaz, and Bruno Taccardi. Effects of heart position on the

[42] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-
dimensionally equidistributed uniform pseudo-random number genera-


[44] M Neunlist and L Tung. Spatial distribution of cardiac transmembrane
potentials around an extracellular electrode: dependence on fiber ori-

[45] B. F. Nielsen, Marius Lysaker, and Aslak Tveito. On the use of the
resting potential and level set methods for identifying ischemic heart

model of geometry and fibrous structure of the heart. *Am J Physiol


[50] Cornel Pater. The blood pressure "uncertainty range" - a pragmatic ap-
proach to overcome current diagnostic uncertainties (ii). *Current Con-


