Development of a Numerical Platform for the Simulation of Electro-Mechanical Models of the Human Heart

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Science and Engineering

2022

Thomas M A Scrase

Department of Physics and Astronomy

MANCHESTER 1824

The University of Manchester

Contents

Final word count: 40858

List of Figures

List of Tables

Acronyms

ADP Adenosine Diphosphate

ALE Arbitrary Lagrangian Eulerian

ATP Adenosine Triphosphate

BDF Backward Differentiation Formula

CT Computerised Tomography

DEM Discrete Element Method

DOF Degree of Freedom

DOFs Degrees of Freedom

FDM Finite Difference Method

FE forward Euler

FEM Finite Element Method

FERL Forward Euler with Rush-Larsen

FSI Fluid-Solid Interaction

FVM Finite Volume Method

HE	Heun-Euler
MR	Magnetic Resonance
ODE	Ordinary Differential Equation
ODEs	Ordinary Differential Equations
Oomph-lib	Object Oriented Multi-physics Library
PDE	Partial Differential Equation
PDEs	Partial Differential Equations
PVD	Principle of Virtual Displacements
ROA	Rate of Advance
SQT	The Short QT Syndrome
TR	Trapezoid

Abstract

Cardiac diseases are among the most prevalent in the world. The scope and impact of these diseases on society are far-reaching. Therefore, developing tools to aid in identifying and understanding the underlying mechanisms of these diseases is an ongoing and vital field of research in which computational modelling has emerged as an essential tool. Using flexible, open-source software, and modern numerical methods can aid in this endeavour. In this thesis, I have developed a computational platform, for the simulation of bio-physically detailed cardiac tissue models with general cell models. This platform consists of significant additions to the open-source multi-physics finite element library Object Oriented Multi-physics Library (Oomph-lib) and allows for the novel application of Oomph-lib to cardiac modelling. These developments couple single-cell models to tissue and 2D or 3D anatomical models of the heart. The platform can simulate electrical excitation waves and mechanical contraction of cardiac tissue. In addition to this platform, I have also developed methods of applying adaptive operator splitting methods for the stable and efficient numerical solution of cardiac models which I show outperform other more commonly used methods. In conclusion, an efficient, scalable, open-source computational platform applicable to cardiac modelling has been developed and further areas for development have been investigated and noted that form a basis for further model development and validation.

Declaration of originality

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning

Chapter 1

Introduction

1.1 Motivations, aims, and objectives

Cardiac diseases are among the most prevalent in the world, it has been estimated that approximately 18,6 million deaths worldwide in 2019 were due to cardiac diseases[1]. There exist several diseases, such as Torsades de Pointes, which often present no symptoms and are associated with a very high risk of mortality when untreated. Other diseases, such as The Short QT Syndrome (SQT), are rare, affecting only between 0.02% and 0.1% of the adult population[2] [3]. SQT is thought to be highly lethal for individuals of all ages, with a probability of 40% that an affected individual will experience their first cardiac arrest before the age of 40[4]. This elevated estimated lethality however could be due to a detection bias in that, given the small number of reported cases, a large majority of asymptomatic individuals frequently go undetected. Since the disorder is often symptomless the total number of cases may be much higher. However, since most cases first present with cardiac arrest or sudden cardiac death, there are very few estimates for how many individuals have the disease worldwide. Difficulty in estimating the number of individuals suffering from SQTS may be explained in part by a lack of a clear cut-off between a healthy, abbreviated QTc interval and a pathologically short QTc interval[5][6][2][7][8]. Due to their rarity, such rare diseases can present challenges for developing effective therapeutics. Since so few cases are reported, undertaking human trials to determine safe and effective treatments can be very difficult.

Pre-clinical in-silico modelling is an essential and emerging tool in understanding and designing treatments for many diseases[9]. In-silico modelling provides an environment in which well-tested models of the healthy human myocardium can be subject to experiment to attempt to identify the theoretical mechanisms which underlie diseases[9][10].

Potential therapeutics for diseases can be investigated by using models for drug action, or other potential interventions, to which biophysically detailed tissue models are subjected[9]. In-silico simulations enable a better understanding of protocol parameters, thus providing insight into the expected effects. This provides an opportunity to explore different possible study designs, therefore reducing the uncertainty of the outcomes and resulting in the faster and more efficient development of therapeutics. These methodologies can allow for better design of human and animal trials by identifying mechanistic causes and potential issues with drugs through the use of biophysically detailed in-silico tissue models[11].

Incorporation of patient data into simulations allows for modelling that more accurately reflects patient-specific physiologies and geometries[12]. This is achieved through applying clinical data to parameterize cardiac models[13]. Results of these models are then more likely to reflect the conditions in the patient's body.

The primary aim of this work is to develop an environment for the biophysically detailed simulation of cardiac tissue with general cell models. As a result of the many physical processes which occur within the cardiovascular system, mathematical models of the cardiovascular system are by their nature multi-physics[14] and multi-scale[15] models. Simple models of the heart contain fewer physical systems, for example, the simplest models are those of single cells which contain no models for electrical diffusion, solid deformation, or fluid dynamics. As models become more developed, more physical systems are included in the modelling. The current code within the group which is used for simulating cardiovascular physics is difficult to adapt to include these new physics into the system. A novel numerical platform centred around multi-physics is required so that future modelling decisions can be easily incorporated. There are many numerical platforms for the simulation of the myocardium and connected tissues and systems. However, most are either closed-source or highly specific to the simulation of particular regions of the heart and are very difficult to modify.

The Oomph-lib is developed at the University of Manchester and is a versatile, open-source, platform for running multi-physics simulations[16]. It has been applied to the simulation of a number of physical systems but lacks a number of capabilities required for the simulation of the human heart.

The primary aim of this project is to add a number of those functionalities to Oomph-lib so that they may be applied to robust numerical simulations of the human heart. These additions to Oomph-lib capture a range of existing models for cardiovascular physics and the resultant numerical platform both conforms with the Oomph-lib standard, where possible and accommodates the ease of future additions and development.

In developing the numerical platform, aspects of cardiac models that Oomph-lib is currently unable to simulate must be identified and the required additions to add this functionality must be determined. Due to time and resource constraints, not all aspects of cardiac modelling can be incorporated in the time permitted by the project, so the most widely used and most pressing are prioritised. Once the required additions are identified, the systems of equations which represent those physics are implemented and incorporated into the numerical platform. These additions must conform to the Oomph-lib standard wherever possible, including the Oomph-lib ethos of code reuse and coding standards. Any code developed must be optimised to ensure that memory use and computation time are acceptable and scale suitably as simulation size is increased. The validity of the newly developed modules must be carefully validated before they are applied to simulations used in research. Due to time constraints, specific areas of the library were selected for validation and those that were not have been identified for future validation.

A secondary aim of this project is to investigate the novel application of adaptive operator splitting techniques to solving cardiac systems. Numerical accuracy and stability in solving in-silico models are highly important. Any numerical simulations of the physical world must be carefully validated and determined to be both stable and accurate, particularly those intended to inform potential human or animal trials. However, biophysically detailed models of the physiology of the heart result in systems of equations which can be exceedingly expensive to solve. It is therefore also essential that the tools used to solve these equations are efficient since the number of Degrees of Freedom (DOFs) increases rapidly as simulations approach the scale of the entire organ or larger. To this end, operator splitting techniques such as Strang-splitting are often exploited[17]. However, relatively recent developments in adaptive operator splitting techniques present an opportunity to improve upon these current methods[18]. Here, the application of adaptive operator splitting techniques to simulating cardiac physiology is investigated for several second-order operator splitting methods and the application of higher-order methods is discussed. The results given by the adaptive operator splitting methods are compared to the results due to Strang-splitting, furthermore, the numerical complexity of the methods is compared in order to demonstrate the potential advantages of adaptive operator splitting in the context of cardiac simulations. Application of the library to adaptive operator splitting requires several additional functionalities which support reverting the solution to a previously saved solution

These additions, although intended primarily for the simulation of cardiac systems, represent substantial additions to the Oomph-lib environment including anisotropic solid modelling, anisotropic electrical diffusion, partitioned solutions of finite element formulations from non-linear short time-scale point-source terms, and a pipeline for the generation of meshes from arbitrary geometries. Since the additions developed as part of this work are in line with the Oomph-lib standard, it is possible that they can be used in conjunction with proven Newtonian and non-Newtonian fluid dynamics models which already exist as part of Oomph-lib for the simulation of blood flow within the cardiovascular system.

1.2 Heart physiology

1.2.1 Cardiac tissue: micro to macro scale structures

Cardiac myocytes constitute the majority of the heart muscle. Their primary purpose is to produce contraction by generating mechanical tension. The generation of this mechanical tension is initiated by the electrical depolarisation of the cell. It is common practice, in mammalian atrial and ventricular tissue, to represent the cardiac myocytes as a cylinder of diameter between $10 - 20\,\mu$m and length between $50 - 150\,\mu$m [19, 20, 21, 22]. However, since the size and shape of cardiac myocytes is known to vary in complex ways even in small regions of tissue[23, 24], this is considered to be a coarse representation.

Cardiac myocytes are encased within a bi-lipid layer, the sarcolemma, which serves a similar purpose to the bi-lipid layer in other eukaryotes[25], namely it separates the interior of the cell from its exterior and provides a semi-permeable membrane through which soluble material can diffuse[25].

Embedded within the sarcolemma are ion channels, structures which facilitate the ingress and egress of intracellular and extracellular ion species through the sarcolemma via pores or exchanger proteins. These channels serve to regulate the potential difference of the cell membrane and are essential to ensure proper electrical diffusion and mechanical contraction of the myocyte. Most ion channels are more permeable to a specific ion species and are thus referred to in terms of the ion they permit[26]. Pores provide a route by which ions are allowed to flow through the sarcolemma in the direction of the concentration gradient. Ion pumps are capable of forcing an ion species against a concentration gradient, a process which requires energy provided by the hydrolysis of Adenosine Triphosphate (ATP)[26]. The permeability of ion channels and pumps is greatly affected by a multitude of factors including membrane potential, tissue type, disease, and internal and external ionic concentrations.

The cytoplasm of a myocyte is called the sarcoplasm, it is filled with an aqueous solution containing ion species, lipids, glycosomes and protein enzymes[26]. Submerged in this solution are larger structures, and organelles including mitochondria, myofibrils, the sarcoplasmic reticulum, and the cytoskeleton. The cytoskeleton provides an anchoring structure to which the other organelles are attached[26].

The primary chemical process mitochondria undergo is the production of ATP a nucleotide used to provide energy for most cellular processes in the body[26]. Energy is extracted by the mitochondria through the breakdown of glucose in the presence of oxygen into carbon dioxide and water. This energy is released through ATP hydrolysis, where ATP reacts with water to form Adenosine Diphosphate (ADP), inorganic phosphate, and a hydrogen ion[26].

Present in ventricular myocytes and occurring less frequently in atrial myocytes are transverse tubules or T-tubules. T-tubules are invaginations in the sarcolemma, originating over the Z-line at the end of each sarcomere[27], they, therefore, occur at intervals of approximately $2\,\mu$m. T-tubules can be considered an extension of the myocyte's membrane since they are exposed to the same extracellular fluid as the rest of the cell membrane[28, 29]. Due to this and their high concentration of L-type calcium channels near the calcium release sites of the sarcoplasmic reticulum, they are believed to communicate electrical and calcium signals to the interior of the cell in order to facilitate uniform mechanical contraction.

The sarcoplasmic reticulum is a network structure covering the surfaces of the myofibrils. It contains an excess of calcium ions which are released into the sarcoplasm and subsequently diffuse into the myofibrils which catalyse the sliding of actin and myosin filaments, producing mechanical contraction of the fibre[28, 29]. Pumps on the surface of the sarcoplasmic reticulum ensure muscular contraction halts at the correct time. At the end of phase 3 of the action potential, the flow of calcium into the sarcoplasm from the sarcoplasmic reticulum ends and pumps into the surface of the sarcoplasmic reticulum rapidly pumping calcium ions from the sarcoplasm back into the sarcoplasmic reticulum[30, 31].

Cardiac myocytes are electrically coupled via structures called gap junctions. Gap junctions are channels directly connecting the sarcoplasm of two myocytes and allow for ions, larger molecules, and electrical impulses to flow directly from one cell's sarcoplasm to the other[26]. They are formed by two connexons, one in the sarcolemma of each cell. A connexon is a hemichannel, a pore in the sarcoplasm of a cell, consisting of a number of subunit proteins called connexins. Connexins are named according to their atomic weight, the most common of which found in mammalian hearts are connexins 37, 40, 43, 45, 46, and 50[32]. Connexin43 is the most abundant and, with the exception of the sinoatrial and atrioventricular cells, is found in all parts of the organ[32]. Connexin40 appears to be present exclusively in the electrical conduction systems of the atria and ventricles, less common connexins include, 37, 43, 46, and 50 and have been shown to exist in only specific regions such as the endothelium [33, 34, 35, 36] and atrioventricular valves in rat hearts[37].

1.3 Distinct sub-physics of the heart

1.3.1 The iso-potential, leaky cell

The bi-lipid layer surrounding the cell is non-conducting and has a specific capacitance of around $100\,\mu F/m^2$, with some measurements placing it at $1000\,\mu F/m^2$ in Purkinje cells. The bi-lipid layer contains ion channels which allow for ions to enter and leave the cell, the layer is therefore not a perfect insulator and is considered electrically leaky. The system can therefore be considered as analogous to a resistor and capacitor connected in parallel[38]. By considering the total current through the membrane, I_m, in terms of the currents charging the capacitor (membrane), I_c, and the current flowing through the resistor (channels), I_{ion}, we can write

$$I_m = I_c + I_{\text{ion}}. \tag{1.1}$$

The charge stored in the capacitor can be written in terms of the potential difference across it, V_m, and its capacitance, C_m, as

$$Q = C_m V_m. \tag{1.2}$$

The capacitor charges at a rate given by the current applied to it, and it is given by

$$I_c = \frac{\partial Q}{\partial t}. \tag{1.3}$$

Combining 1.1 and 1.3 gives

$$I_m = \frac{\partial Q}{\partial t} + I_{\text{ion}} \tag{1.4}$$

which, when 1.2 is substituted for Q, provides the equation equating membrane current to change in membrane potential, and ion current of a cell modelled as iso-potential

$$I_m = C_m \frac{\partial V_m}{\partial t} + I_{\text{ion}}. \tag{1.5}$$

In reality, the cell membrane is not iso-potential, its membrane takes time to transmit charge from one section to another. However, considering a cell as anisotropically charged results in a dramatic leap in modelling complexity and is an area of research in its own right.

Since this work is predominantly concerned with the solution to the systems of equations governing cardiac tissue dynamics, the specifics and the general theory of ion channel gating and permeability are beyond the scope of this thesis. However, there exist numerous sources which describe and explain the processes by which they operate.

1.3.2 Cell ion kinetics and dynamics

Ion channels can be found in every cell in the body. Ion channels are structures made of proteins and are embedded in the cell membrane. They regulate several essential processes including maintaining homeostasis and regulating electrical signalling with other cells [39][40]. They achieve this by allowing ions to pass through them in a process which alters the transmembrane potential of the cell and is highly dependent on the physical conditions surrounding the channel[41]. Transmembrane potential, ionic concentration, temperature, mechanical strain, and the presence of other chemical compounds can all affect the function of the ion channel, although most are predominantly modulated via the transmembrane potential[41]. Proteins which constitute the ion channel deform and change their configuration in response to these external effects and, through doing so, regulate the flow of ions through the channel by altering its structure[41]. Embedded within the cell membrane are many thousands of ion channels. Each ion channel can be one of several different types which are defined by the proteins it is comprised of. The types of ion channels found in a given cell are known to vary depending on its location in the body and each type of ion channel is observed to have a different response to external effects[41][39]. Each type of ion channel will, in general, have some specificity to a certain ion species, however, there are many which permit the flow of several ion species[41][42].

If a given type of ion channel is present, a large number of such ion channels can generally be found within the cell membrane. Together, the small flux of ions generated by each ion channel combine to form an ion current[41]. For a given cell, these channels will have a maximal conductance, often labelled $\bar{G}_x$, where x is the label given to the ion current.

In cardiac cells, ion currents play a particularly important role in the genesis and maintenance of the action potential[41]. The action potential is the time course of the transmembrane potential of a cell produced by the complex opening and closing of ion channels within the cell membrane[41]. Maintaining the correct action potential is essential for ensuring both proper electrical signalling between cells in all regions of the myocardium and proper mechanical contraction of cardiomyocytes[41][40].

The dynamics of each ion current are governed by the dynamics of the underlying ion channels and their constituent proteins. Hence, as a result of the presence of a given ion channel in the cell membranes, only specific ion currents can be found in each region of the body[40]. For simple ion channels, each channel can be considered to be either open or closed, therefore, for a large number of channels of the same type, the quantity K_x can be defined to be the fraction of those channels which are open. $K_x = 0$ corresponds to all the channels being closed, and $K_x = 1$ corresponds to them being open. Ion channels can be classified according to the mechanisms which affect their gating kinetics, the major categories being voltage-dependent, ligand-dependent, and mechano-sensitive.

Voltage-dependent channels are the most numerous[43][44][41]. Their conductance is modulated by the transmembrane potential, as such their kinetics are directly affected by the stage of the action potential the cell is currently in[43][41].

Ligand-dependent channels have gating kinetics which are dependent on the binding of ligands to receptor sites either on the intracellular or extracellular regions of the ion channel[45][44][41]. A number of ion channels commonly found in the heart are ligand-dependent and act to couple non-electrical processes to the electrical dynamics of the cell. For example, the ATP-sensitive potassium channel $I_{\mathrm{K,ATP}}$ is dependent on the ratio of intracellular concentrations of ADP and ATP and therefore couples the electrical dynamics to the metabolic state of the cell[45][44][41].

Mechano-sensitive channels change configuration and therefore conductance in response to mechanical stimuli[46]. Their presence directly couples the mechanical forces experienced by the cell to its electrical dynamics and action potential[46].

In general, the fraction of open ion channels for a given current, K_x, changes according to some non-linear time-dependent system of equations. Models of these dynamics can vary in complexity from the relatively simple Hodgkin-Huxley[38] to the more complex Markov chain[47][48]. Most, however, consider a number of channel variables, $\underline{\omega}$, which vary in time according to the general equation

$$\frac{\mathrm{d}\underline{\omega}}{\mathrm{d}t} = f(\underline{\omega}, V_m) \tag{1.6}$$

where V_m is the transmembrane potential of the cell. The open fraction of the current is then written as a function of these variables

$$K_x = K_x(\underline{\omega}). \tag{1.7}$$

Ions can move through ion channels due to the effects of both diffusion and transmembrane potential[41][44]. Diffusive effects will in general cause ions to move through ion channels from the side of high concentration to the side of lower concentration[41][44]. Through electromagnetic forces, the difference in electrical charge between the extracellular and intracellular spaces will act to force the charged ions across the cell membrane in a specific direction dependent on the difference in the charge of the two sides of the channel and the charge of the ion[41][44]. The combination of both the diffusive and electrical effects is often called the electrochemical gradient since it concerns both diffusive (chemical) and electrical effects.

An important quantity which can be determined for each ion species is the Nernst potential[49]. It defines the transmembrane potential at which the diffusive and electrical effects on the ion species cancel out to produce a net-zero flux across the cell membrane and is given by

$$E_x = \frac{RT}{zF} \ln \left(\frac{[x]_o}{[x]_i} \right) \tag{1.8}$$

where, x is the ion species, $[x]_o$ and $[x]_i$ are the extracellular and intracellular concentrations of x respectively, R is the universal gas constant, T is the temperature in Kelvin, F is Faraday's constant, and z is the valence of the ion species.

The total ionic current produced by an ion channel can be written as

$$I_x = \bar{G}_x K_x (V_m - E_x) \tag{1.9}$$

where I_x is the current, $\bar{G}_x$ is the maximal conductance, K_x is the fraction of open channels, V_m is the transmembrane potential, and E_x is the Nernst potential of the ion species carried by the current[41][44][49].

There also exist ion pumps which actively transfer ions across the cell membrane[50]. Since such ion channels can act against the direction of the electrochemical gradient, they require energy, which is derived from ATP hydrolysis, in order to operate[50].

1.3.3 The action potential morphology and function of different regions of the heart

Cellular heterogeneity throughout the heart results in different regions of the heart expressing different proportions and types of ion channels[49][51]. This results in the distinct regions of the heart producing very different electrophysiological behaviours. These heterogeneous behaviours are essential to proper heart function.

Sinoatrial node

The sinoatrial node is located in the right atrium and is the primary pacemaker of the heart. It is capable of spontaneous excitation without requiring an external stimulus to begin depolarisation. Its resting diastolic potential lies between $-60\,mV$ and $-50\,mV$ with a slow upstroke velocity of around $5\,mV/ms$ in humans[51]. As a result of electrical coupling with cells in the atria, sinoatrial node cells located on the periphery have a more negative diastolic potential. Cells within the sinoatrial node have a diameter between $5\,\mu m$ and $10\,\mu m$ and an action potential duration between $100\,ms$ and $200\,ms$. The sinoatrial node has a relatively slow conduction velocity which is typically less than around $0.05\,m/s$.

Atrial myocytes

Atrial myocytes are the primary component of atrial tissue. They are incapable of spontaneous excitation and require an external stimulus to begin depolarisation. They have a resting potential of between $-65\,mV$ and $-80\,mV$ [52] and a fast upstroke velocity of between $150\,mV/ms$ and $300\,mV/ms$ [53, 54, 55, 56, 57][51][58].

Measurements of the action potential of human atrial myocytes have been shown to vary between $150\,ms$ and $500\,ms$ with an external stimulus of $1Hz$[53, 54, 55, 56, 57]. Atrial myocytes are approximately cylindrical, with a diameter ranging from approximately $10\,\mu m$ to $15\,\mu m$ and a length of around $120\,\mu$m. The conduction velocity in atria is significantly higher than that of the sinoatrial node, ranging from around $0.3\,m/s$ to $0.4\,m/s$.

Atrioventricular node

The atrioventricular node serves to delay cardiac conduction from the atria to the ventricles in order to allow the atria to contract and empty their contents before the ventricles are allowed to contract. It is located near the partition between the atria, at the base of the right atrium. Cells in the atrioventricular node, as in the sinoatrial node, are capable of self-excitation with a slow upstroke velocity of around $5\,mV/ms$ to $15\,mV/ms$ [51][58]. The conduction velocity in the atrioventricular node is faster than that in the sinoatrial node, but slower than in the atria, at around $0.1\,m/s$. Such cells have an action potential duration typically between around $100\,ms$ and $300\,ms$ [51][58] and a resting potential of between $-60\,mV$ and $-70\,mV$ [51][58]. The diameter of cells in the atrioventricular node is similar to that of cells in the sinoatrial node, at between around $5\,\mu m$ to $10\,\mu m$ [51][58].

The Bundle of His and Purkinje system

The Bundle of His and Purkinje system is the fast conduction and signalling pathway to the ventricles. They begin at the atrioventricular node, and from there they penetrate the tissue separating the atria and the ventricles and extend down the septum between the ventricles. They then separate into the left and right bundle branches which penetrate the ventricular tissue itself before branching into the network of Purkinje fibres. The His bundle and Purkinje fibres ensure near uniform activation of the ventricles by rapidly conducting electrical signals to the apex of the ventricles and from there throughout the endocardial regions[51][58]. The Bundle of His consists of wide, rapidly conducting muscle fibres with a diameter of around $100\,\mu m$[58] and an upstroke velocity of between approximately $500\,mV/ms$ and $700\,mV/ms$[51][58], which result in a conduction velocity of between $2\,m/s$ and $5\,m/s$[51][58]. They have a typical resting potential between $-90\,mV$ and $-95\,mV$[51][58] and an action potential duration ranging from around $296\,ms$ to $342\,ms$[59].

Ventricular myocytes

Throughout the ventricles, ventricular cardiomyocytes can exhibit different characteristics. APD varies not only between the left and right ventricles but also transmurally from the epicardial to endocardial regions[51]. The endocardial cells (ENDO) are those on the inside of the ventricle and the epicardial cells (EPI) are those on the outside. There is also a region of mid-myocardial cells (MCELL) which lie between the ENDO and EPI regions. Biophysically detailed simulations commonly use a different model for cardiomyocytes in this region. MCELLs have the longest APD with a length comparable to that of Purkinje fibre cells. APD in EPI cells is shorter than that of ENDO cells for most species, however, some experimental observations indicate that the opposite may be true in humans[60].

Ventricular myocytes have a similar diameter to atrial myocytes of $10-$

$20\,\mu m$ [51][58] and a length of approximately $80 - 100\,\mu m$ [61]. Conduction velocity in the ventricles is faster than in the atria, with a speed of between $0.3-1.0\,m/s$ [51][58][61]. Normal depolarisation of the ventricles starts at the endocardium near the apex, initiated by the Purkinje fibres, and propagates to the epicardial wall at the base.

1.3.4 Electrical conduction

Extracellular matrix

The extracellular matrix provides the supporting structure for cardiac myocytes and all other cells in the heart and thus determines the passive mechanical properties of cardiac tissue in the absence of contractile muscle fibres. It consists of strands of collagen and elastin, fibrous proteins, which surround and support cardiac cells. The composition of the extracellular matrix varies throughout the myocardium, varying with age, species, and tissue type[62, 63]. The varying arrangement of the extracellular matrix at different regions of the myocardium results in the diverse structures observed. Within the endomysium, the collagen network surrounds a single myocyte forming fibres[64]. In the perimysium collagen, fibres envelop groups of several adjacent myocytes, forming bundles which produce the laminar structure found in the myocardium[64]. The epimysium is a result of collagen and elastic forming a layer around the surface of the entire muscle[64] as is seen in the interior and exterior walls of the chambers.

1.4 Anisotropic solid mechanics

1.4.1 Structure

Muscle fibres

Cardiac muscle is composed of cardiomyocytes, or fibres, which are themselves composed of smaller subunits. Each fibre has two membranes, an inner membrane and an outer membrane consisting of thin collagen fibres.

Myofibrils

Cardiac muscle fibre contains thousands of rods known as myofibrils[26] arranged in groups. Spaced along the length of each myofibril are thousands of discs called z-discs. The part of the myofibril which lies between adjacent z-discs is called the sarcomere and so the myofibril consists of thousands of sarcomeres. The sarcomere is a specific configuration of thick myosin filaments and thin actin filaments[28, 29, 26] which slide over each other during cellular contraction to produce a contractile force. Actin filaments extend from the z-discs in both directions away from the z-disc parallel to the length of the myofibril. The z-discs are arranged obliquely within the myofibril and extend into adjacent myofibrils, which connects them together and so forms the clustered groups. The traversal of the z-discs through the muscle fibre results in light and dark bands which give cardiac muscle its striped appearance.

Myosin filaments are arranged parallel to one another, connected perpendicularly through their centres by M-line proteins. These proteins act to maintain a regular array of thick myosin filaments within the sarcomere[28, 29, 26].

The thick myosin filaments are made up of many molecules of the protein myosin. Myosin proteins have heads which protrude from the sides of the myosin filaments to form structures known as cross bridges. Actin filaments interact with the myosin filaments via the cross-bridges to produce the sliding motion which results in contraction.

Actin and myosin filaments partially overlap which produces a striped appearance in the myofibril, this is in addition to the striped appearance of the muscle fibre due to the z-discs.

Sarcoplasm, transverse tubules, sarcoplasmic reticulum

Intracellular space contains the thousands of myofibrils which make up the muscle fibre. Between these fibres is the sarcoplasm, the intracellular fluid which contains large amounts of potassium, magnesium, phosphate, and protein enzymes which are essential for ensuring proper mechanical contraction of the fibre[28][65]. Additionally, many mitochondria are aligned parallel to the fibres in order to provide a large amount of energy in the form of ATP necessary for muscular contraction[28][65].

Transverse tubules, or T-tubules, are deep invaginations in the membrane of the fibre. They traverse the interior of the fibre, across the myofibrils, and back to the exterior of the cell[28][65]. T-tubules are therefore an extension of the cell membrane and can therefore communicate changes in the electrical potential to the interior of the cell[28][65]. It is the molecular machinery in the interior of the fibres which detects these changes in the membrane potential and initiates muscular contraction[28][65].

The sarcoplasmic reticulum is a space within the cell kept separated from the sarcoplasm by a semi-permeable membrane. The sarcoplasmic reticulum covers the surfaces of the myofibrils and contains an excess of calcium ions. The terminals of the sarcoplasmic reticulum lie adjacent to the T-tubules and contain calcium channels. Electrical excitation of the T-tubules causes activation of these calcium channels in the terminals of the sarcoplasmic reticulum, and the subsequent electrical excitation through the sarcoplasmic reticulum causes calcium channels throughout the sarcoplasmic reticulum to activate[28][65]. This activation causes calcium ions to flow from the sarcoplasmic reticulum into the sarcoplasm, which catalyses the reactions between actin and myosin filaments in the myofibrils causing contraction[28][65]. When the flow of calcium from the sarcoplasmic reticulum into the sarcoplasm ceases, calcium pumps in the sarcoplasmic reticulum rapidly pump calcium ions back into the sarcoplasmic reticulum which terminates muscular contraction.

1.4.2 Mechanism of contraction

Myosin filament

Each thick myosin filament consists of several myosin proteins. The myosin protein is formed of a rod and two heads. The rod is formed from two heavy chains wrapped into a helix. The heads are globular structures formed from the folding of the heavy chains and are connected to the main filament by an arm. Additionally, each head also contains two light chains. The heads of each myosin molecule, which protrude from the surface of the myosin filament, form the cross-bridges which interact with actin filaments[28][65].

Actin filament

Actin filaments consist of three proteins, actin, tropomyosin, and troponin. The tropomyosin and actin filaments are wrapped around each other. In the resting state, the tropomyosin protein covers the active sites on the actin filament, which are believed to interact with ADP to inhibit contraction. Along the length of the tropomyosin protein are troponin proteins. Troponin consists of three proteins, Troponin I, Troponin T, and Troponin C. The troponin proteins bind tropomyosin to the actin[28][65].

Chapter 2

Mathematical modelling principles

2.1 Ion dynamics

2.1.1 Modelling ion channels

Hodgkin Huxley

The continuous voltage-dependent and time-dependent part of a channel are called the current and is represented, in part, by several 'gating variables'[66]. Hodgkin and Huxley proposed a gating mechanism to describe the behaviour of a channel in which each gate must be in an open state in order to permit the flow of ions through the channel[38]. Consider a gate with the label i, the gating variable n_i represents the proportion of channel pores for which gate i is open. Gate i of the channel is then considered open for every pore if $n_i = 1$, closed for every pore if $n_i = 0$, and open for some if it has a value in between 0 and 1. By representing maximum channel conductance as $\bar{g}_x$, a description of channels conductance can be written as

$$g_x = \bar{g}_x \sum_i n_i. \tag{2.1}$$

By assuming that the state of each gate is independent of the others, the dynamics of n_i were assumed to be described by

$$\frac{\mathrm{d}n_i}{\mathrm{d}t} = \alpha_{n_i}(1 - n_i) - \beta_{n_i} n_i, \tag{2.2}$$

where α_{n_i} is the probability of a closed gate transitioning to an open state, and β_{n_i} is the probability of an open gate transitioning to a closed state. For simplicity and computational convenience, equation 2.2 is often rewritten in the form

$$\frac{\mathrm{d}n_i}{\mathrm{d}t} = \frac{n_{i_\infty} - n_i}{\tau_{n_i}}, \tag{2.3}$$

where

$$\tau_{n_i} = \frac{1}{\alpha_{n_i} + \beta_{n_i}} \tag{2.4}$$

is referred to as the time constant for the gating variable, and

$$n_{i_\infty} = \frac{\alpha_{n_i}}{\alpha_{n_i} + \beta_{n_i}} \tag{2.5}$$

is the steady-state relation for the gating variable. The full Hodgkin-Huxley ion channel formulation can now be written in the following form:

$$I_x = g_x \left(\prod_i n_i \right) (V - E_x) \tag{2.6}$$

Since the gating variables were introduced as a mathematical tool to describe the data and weren't derived from physical processes, seeking physical interpretations for them in order to quantify their dynamics is moot. However, in order to promote understanding, a pseudo-physical interpretation of n_i may be the probability that a protein within the pore structure has undertaken a required conformational change to allow conduction. Another interpretation is that n_i is the probability that a molecule required for the transmission of ions has bound to an active site on the pore. Since foldings of proteins and molecular dynamics is a stochastic process, by assuming a protein or molecule can either be in the required state or not in the required state equation 2.2 can be attributed some physical meaning. The value of gating variables need not be determined via an Ordinary Differential Equation (ODE). Often, an instantaneous voltage-dependent gate is required in order to fully capture a channel's behaviour. In these cases, the form of a gating variable is often determined through experience or insight into the channel being modelled.

Markov chain

Despite inspiring a new field of modelling transmembrane cellular processes and closely reproducing observed current data, the HH model lacks a physical basis and instead represents a purely mathematical approach to describing macro-scale data. For example, the governing equation for gating variables in terms of transition rates is far too simplistic to capture all possible behaviours reliably. Furthermore, transition rates are determined purely through curve fitting to steady states and not by consideration of underlying dynamic molecular processes. In order to facilitate the intuitive modification of cell models in response to the effects of genetic disorders, diseases, and drugs, use is often made of a more sophisticated and general model.

A further drawback of the HH model is the assumption that the possible states a channel can occupy are independent of one another, experiments have since shown that this is not the case. In fact, due to physical limitations, ion channels cannot in general directly transition between any two given states and are instead confined to a small set of transitions. The set of transitions a channel can make reflect physically permissible conformational changes to the channel's components. The use of Markov Chain processes lends itself to modelling and simulating these state-dependent ion channel models. The suitability of Markov chains in modelling ion channel processes has been widely studied and their application has resulted in rapid progress and numerous advancements in the field[67][68]. A Markov chain is a memory-less stochastic process wherein the system can occupy one of a number of states, and the probability of any given transition is only dependent on the current state of the system[69]. In general, the set of states is labelled S_i and the probabilities of the system transitioning between them are given by $P(S_i \rightarrow Sj)$. By taking the assumption that a large number of channel pores are evenly distributed over the cell membrane, the Markov chain process representing the dynamics of a single pore is transformed into a transition rate representation for the entire channel. In doing so, the probabilities in the Markov chain model become the proportion of channel pores in each state, and the transition probabilities become the transition rates of the pores between them. Consider the case of a two-state Markov chain process, figure 2.1 describes the dynamics of a large number of such chains. The proportion of the Markov chains in states A and B are labelled S_A and S_B respectively, and they change according to the transition rates α, and β.

In general the transition rates α and β are time-dependent, the dynamics

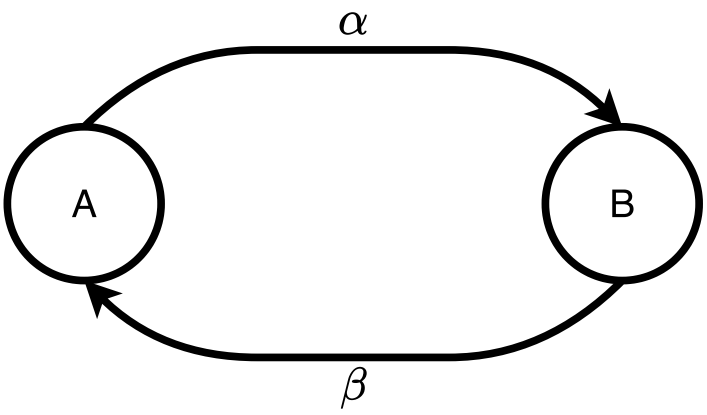


Figure 2.1: A schematic of a two-state Markov chain. The proportion of the population in state A is given by S_A and the proportion of the population in state B is given by S_B. Transitions from state A to state B occur at a rate of α, and transitions from state B to state A occur at a rate of β.

of S_A and S_B are given by the equations

$$\frac{\mathrm{d}S_A}{\mathrm{d}t} = \beta(t)S_B - \alpha(t)S_A, \tag{2.7}$$

$$\frac{\mathrm{d}S_B}{\mathrm{d}t} = \alpha(t)S_A - \beta(t)S_B. \tag{2.8}$$

Since S_A and S_B represent the proportions of pores in the only two possible states, it follows that $S_B = 1 - S_A$. By inserting this into equations 2.7, the two equations reduce to

$$\frac{\mathrm{d}S_A}{\mathrm{d}t} = S_A\alpha_1(t) - (1 - S_A)\beta_1(t) \tag{2.9}$$

which is equivalent to equation 2.2. This demonstrates that the dynamics of a Hodgkin-Huxley gating variable model is a special case of a Markov chain model. Indeed, a Hodgkin-Huxley channel model with N gating variables can be described by a set of N independent 2-state Markov chains. When applied to modelling ion channel dynamics, the Markov chain states can represent the possible configurations under which proteins in each pore of the channel can be folded. Changes to the transmembrane potential, V, of a cell induce dipoles within channel proteins, these dipoles produce free energy barriers which must be overcome for conformational changes to the proteins to occur. Reaction rate theory describes the transition rates between possible protein states as

$$r(V) = r_0 \exp \frac{-\Delta G(V)}{RT}, \tag{2.10}$$

where r_0 is a constant, R and T are the universal gas constant and temperature respectively, and $G(V)$ is the free energy barrier as a function of transmembrane potential. In general, for a protein to transition from its current state to another, it must first pass through an intermediary state in order to overcome the resistance of any induced dipoles. For this reason, the free energy barrier is often written as $G(V) = G^*(V) - G_0(V)$ where $G^*(V)$ is the free energy of the intermediary state, and $G_0(V)$ is that of the current state. A simplifying assumption which is often made is that of a small free energy barrier. By application of Taylor's theorem to $G(V)$, equation 2.10 can then be simplified to

$$r(V) = r_0 \exp \frac{[a + bV]}{RT} \tag{2.11}$$

where r_0, a, and b are parameters which must be determined experimentally for each transition rate in the model. As with the Hodgkin-Huxley formulation, more factors such as ionic or molecular concentrations can be incorporated in order to capture other dependencies exhibited by particular channels[70].

2.2 Cell models

For cardiac electrophysiological modelling, myocytes can be treated as a coupled capacitor-resistance system, where the capacitor represents a semi-permeable membrane that separates the intracellular from the extracellular space, and the variable resistance represents ion channels embedded in the lipid bilayer membrane [71]. The conductivity of each channel is dependent on its particular molecular structure and other environmental factors including transmembrane potential, temperature, and intracellular and extracellular ionic concentrations [71] [72] [73]. Modelling the electrophysiological behaviour of a cardiomyocyte then becomes synonymous with modelling the dynamics of these channels and the resultant kinetics of the ionic species they carry. The entire population of a given channel type can be grouped together and modelled as a single integrated channel. This is because the underlying, discrete, protein dynamics in each channel are governed by the same kinetics and thus the channel can be represented by a continuous stochastic process as the number of such pores becomes very large [69]. In order to capture the complex nature of the channel properties, a number of Ordinary Differential Equations (ODEs) is required to describe their activation and inactivation process. Each channel model can vary in complexity and may result in a large number of state variables, each of which is modelled by a set of ODEs. A cell model consists of many such channels, each carrying one of a number of ionic species. Resultant cell models can therefore contain a very large number of coupled equations, each governed by a potentially stiff differential equation. A general electrophysiological model of cardiomyocyte can be described as:

$$\frac{\mathrm{d}V_m}{\mathrm{d}t} = -\frac{I_{\text{ion}}}{C_m} \tag{2.12}$$

$$\frac{\mathrm{d}\underline{\omega}}{\mathrm{d}t} = \underline{f}(\underline{\omega}, V_m, t) \tag{2.13}$$

$$I_{\text{ion}} = \sum_j I_j(\underline{w}, V_m, t). \tag{2.14}$$

where $C_m(Fm^{-2})$ is the cell membrane capacitance, $\underline{\omega} \in \mathbb{R}^N$ is the vector of N cell state-variables, $V_m(mV)$ is the transmembrane potential, $I_{\text{ion}}(Am^{-2})$ is the total transmembrane ionic current generated by each of the individual ion channels $I_j(Am^{-2})$, t is time, and each element in the vector $\underline{f} : \mathbb{R}^N \mapsto \mathbb{R}^N$ models the time-varying dynamics of the corresponding element in $\underline{\omega}$. Due to the variety of physical processes $\underline{\omega}$ represent, $\underline{f}$ can take a variety of forms of modelling equations, each representing different physiological processes, such as the dynamical course of ionic concentrations, and distribution of conformational states of protein subunits which govern the activation and inactivation of ion channel gating kinetics. Similarly, each I_j can vary in complexity from relatively simple Hodgkin-Huxley models of currents flowing through pores[71] to models of ion exchangers and pumps[74].

This results in the general form of cell models which appear in most publications

$$\frac{\partial\underline{\omega}}{\partial t} = \underline{f}(\underline{\omega}, V_m, t). \tag{2.15}$$

The total ionic flux across the cell membrane is given by the sum of those due to the individual channels,

$$I_{\text{ion}} = \sum_j I_j(\underline{\omega}, V_m, t). \tag{2.16}$$

2.3 Continuum mechanics

Continuum mechanics is a vast subject which covers a wide variety of physical modelling principles and requires a breadth of preliminary mathematical knowledge. Although the derivation of the governing equations of elasticity is dependent on this knowledge, their implementation does not require so much prior information. For this reason, only those areas specific to the implementation of the governing equations of electrical diffusion and anisotropic hyperelastic solids within Oomph-lib are included. The derivations here closely follow that within[75] which should be referred to for a complete derivation.

2.3.1 Describing the world, curvilinear coordinates and covariant and contravariant basis vectors

We apply a set of Lagrangian coordinates ξ^i to parameterise a position vector in the undeformed configuration as $\underline{r} = \underline{r}(\xi^i)$. The choice of the specific Lagrangian coordinates used depends on the application, however, for computational purposes, it is often advantageous for the governing equations to be as compact as possible which is often achieved through the use of the Cartesian coordinate system.

The vector $\underline{r}$ can be written in the Cartesian coordinate system in terms of its components as $\underline{r}(\xi^i) = x^K(\xi^i)\underline{e}_K = x_K(\xi^i)\underline{e}^K$ where $\underline{e}_K$, and $\underline{e}^K$ are the covariant and contravariant Cartesian basis vectors. We note that, since the Cartesian coordinate system is orthonormal, it follows that $\underline{e}_K = \underline{e}^K$.

For general curvilinear coordinates, there are natural basis vectors which can be applied to represent quantities. The covariant vectors for curvilinear coordinates ξ^i are defined by

$$\underline{g}_i = \frac{\partial x^K}{\partial \xi^i} \underline{e}_K. \tag{2.17}$$

Similarly the contravariant vectors are defined by

$$\underline{g}^i = \frac{\partial \xi^i}{\partial x^K} \underline{e}^K. \tag{2.18}$$

The coordinates of a material point within a deformed body can be written in terms of its original, undeformed coordinates as

$$\underline{R}(\underline{r}) = X_K(\underline{r})\underline{e}^K = X^K(\underline{r})\underline{e}_K. \tag{2.19}$$

A mapping between the undeformed and deformed coordinates can be written in the Cartesian basis in the form

$$\mathbf{F} = \frac{\partial X_I}{\partial x_J} \underline{e}_I \otimes \underline{e}^J. \tag{2.20}$$

Physical constraints are imposed on this mapping by the realisation that the body must deform such that a continuous, injective mapping from the undeformed to the deformed configuration exists. These constraints demand that

$$\det \mathbf{F} = J > 0 \tag{2.21}$$

where the additional constraint that the relative orientation of material lines is preserved is applied which disallows negative values.

The matrix F with components F_{IJ} is one possible representation of the deformation gradient tensor. The deformation gradient tensor represents the mapping of undeformed line elements to their corresponding deformed line elements. In this way, a line element, $\underline{dr}$ with components in the Cartesian basis $\mathrm{d}r_J$ are convected with the deformation of the solid and has Cartesian coordinates in the deformed configuration $\mathrm{d}R_I = F_{IJ}\,\mathrm{d}r_J$.

Similarly to undeformed coordinates, deformed coordinates can be parameterised by a set of Lagrangian coordinates, χ^i, as $\underline{R}(\chi^i) = X^K(\chi^i)\underline{e}_K = X_K(\chi^i)\underline{e}_K$. The Lagrangian coordinates used to parameterise the deformed coordinates can, in general, differ from those used for the undeformed coordinates, however, for this application, it will be assumed that they are the same. The covariant and contravariant basis vectors of the deformed configuration are defined to be

$$\underline{G}_i = \frac{\partial X^K}{\partial \xi^i}\underline{e}_K. \tag{2.22}$$

and

$$\underline{G}^i = \frac{\partial \xi^i}{\partial X^K}\underline{e}^K. \tag{2.23}$$

respectively.

The deformation gradient tensor can be re-written in terms of the undeformed contravariant and deformed covariant basis vectors since

$$\mathbf{F} = \frac{\partial X^I}{\partial x^J}\underline{e}_I \otimes \underline{e}^J = \frac{\partial X^I}{\partial \xi^k}\frac{\partial \xi^k}{\partial x^J}\underline{e}_I \otimes \underline{e}^J = \left(\frac{\partial X^I}{\partial \xi^k}\underline{e}_I\right) \otimes \left(\frac{\partial \xi^k}{\partial x^J}\underline{e}^J\right) = \underline{G}_k \otimes \underline{g}^k, \tag{2.24}$$

where $\underline{a} \otimes \underline{b}$ represents the dyadic product of two vectors $\underline{a}$ and $\underline{b}$.

By inverting the expressions 2.17, 2.18, 2.22, and 2.23 the following relations between the basis vectors can be constructed

$$\underline{e}_I = \underline{e}^I = \frac{\partial x^I}{\partial \xi^k}\underline{g}^k = \frac{\partial \xi^k}{\partial x^I}\underline{g}_k = \frac{\partial X^I}{\partial \xi^k}\underline{G}^k = \frac{\partial \xi^k}{\partial X^I}\underline{G}_k \tag{2.25}$$

which are useful when transforming tensor components between different bases.

2.3.2 Tensor product and tensor contraction

The tensor product can be defined through the action of the dyadic product of two vectors $\underline{a}$ and $\underline{b}$, $\underline{a} \otimes \underline{b}$ on a vector $\underline{c}$ to be

$$(\underline{a} \otimes \underline{b})(\underline{c}) = (\underline{b} \cdot \underline{c})\underline{a}. \tag{2.26}$$

The tensor contraction between two tensors given by the dyadic products $\underline{a} \otimes \underline{b}$, and $\underline{c} \otimes \underline{d}$ is defined to be

$$(\underline{a} \otimes \underline{b}) \cdot (\underline{c} \otimes \underline{d}) = (\underline{b} \cdot c)(\underline{a} \otimes \underline{d}). \tag{2.27}$$

2.3.3 Metric tensors

Important quantities when describing deformation are the covariant and contravariant undeformed and deformed metric tensors. These are given by the inner product of the covariant and contravariant undeformed and deformed basis vectors respectively and are listed below

Undeformed metric tensors:

$$\text{Covariant } g_{ij} = \underline{g}_i \cdot \underline{g}_j = \frac{\partial x^K}{\partial \xi^i}\frac{\partial x^K}{\partial \xi^j} = g_{ji} \tag{2.28}$$

$$\text{Contravariant } g^{ij} = \underline{g}^i \cdot \underline{g}^j = \frac{\partial \xi^i}{\partial x^K}\frac{\partial \xi^j}{\partial x^K} = g^{ji} \tag{2.29}$$

$$\text{Mixed } g_i^j = \underline{g}_i \cdot \underline{g}^j = \frac{\partial x^K}{\partial \xi^i}\frac{\partial \xi^j}{\partial x^K} = \delta_i^j \tag{2.30}$$

Deformed metric tensors:

$$\text{Covariant } G_{ij} = \underline{G}_i \cdot \underline{G}_j = \frac{\partial X_K}{\partial \xi^i}\frac{\partial X^K}{\partial \xi^j} = G_{ji} \tag{2.31}$$

$$\text{Contravariant } G^{ij} = \underline{G}^i \cdot \underline{G}^j = \frac{\partial \xi^i}{\partial X^K}\frac{\partial \xi^j}{\partial X^K} = G^{ji} \tag{2.32}$$

$$\text{Mixed } G_i^j = \underline{G}_i \cdot \underline{G}^j = \frac{\partial X^K}{\partial \xi^i}\frac{\partial \xi^j}{\partial X^K} = \delta_i^j \tag{2.33}$$

The right Cauchy-Green deformation tensor

The right Cauchy-Green deformation tensor is given by $C = F^T F$. It represents the square length of deformed line elements relative to the length of their corresponding undeformed line elements.

The pullback of a tensor from the deformed to the undeformed configuration

Given a tensor, $\tilde{A}$, in the undeformed configuration which convects with the solid, there exists a corresponding tensor, A, in the deformed configuration. The two are then related by

$$\tilde{A} = F^{-1} A F^{-T}. \tag{2.34}$$

Volume elements in the deformed configuration, $\mathrm{d}\nu_t$, and the undeformed configuration $\mathrm{d}\nu_0$ are related by $\mathrm{d}\nu_t = J\,\mathrm{d}\nu_0$.

The Arbitrary Lagrangian Eulerian formulation

Within each finite element, the value of a quantity, u is represented by interpolation between the values at the nodes in an equation given by

$$u(\underline{s}, t) = U_k(t)\psi_k(\underline{s}) \tag{2.35}$$

where $\underline{s}$ is the local coordinate in the element, U_k is the value at the k-th node, and $\psi_k(\underline{s})$ is the basis function associated with the kth node evaluated at local coordinate $\underline{s}$.

The quantity $\frac{\partial U_k}{\partial t}$ represents the rate of change of the quantity at the moving node, not at the fixed Eulerian position. Often the rate of change at the fixed Eulerian position is what is required by the governing equation. The rate of change in the Eulerian frame can be determined through the material derivative of the quantity

$$\frac{Du}{Dt} = \frac{\partial u}{\partial t}(\underline{s}, t) + \frac{\partial u}{\partial X_k}(\underline{s}, t)\frac{\partial X_k(\underline{s}, t)}{\partial t} \tag{2.36}$$

X_m is determined by interpolation of the coordinates at the nodes through the equation

$$X_m(\underline{s}) = X_{mk}(t)\psi_k(\underline{s}) \tag{2.37}$$

where X_{mk} is the k^{th} component of the coordinate of the k^{th} node.

Through re-arranging for the Eulerian derivative, 2.36 gives

$$\frac{\partial u}{\partial t}(\underline{s}, t) = \frac{\partial U_k(t)}{\partial t}\psi_k(\underline{s}) - U_k(t)\frac{\partial \psi_k(\underline{s})}{\partial X_m}\frac{\partial X_m(t)}{\partial t}. \tag{2.38}$$

Equation 2.38 is referred to as the Arbitrary Lagrangian Eulerian (ALE) and is used to evaluate the time-derivative in Oomph-lib elements when the nodes of the element move with time, as is the case in moving domains.

2.4 Electrical diffusion

Cardiac tissue is a syncytium of cardiac cells, which are coupled electrically through connexin proteins [76] [77] [78] between adjacent cells, which allows for the diffusion of membrane excitation potentials from cell to cell. Thus, the model of electrophysiology of cardiac tissue, in general, consists of two parts, cell kinetics (i.e., a set of ODEs as described in Equation 2.13), and diffusion of membrane potentials in the tissue via intercellular electrotonic coupling, which is modelled by one or more Partial Differential Equations (PDEs).

2.4.1 Derivation of the conductance tensors in terms of the preferred directions

Electrical conduction in the heart is heavily influenced by conductance. Conductance is represented by the conductance tensors, these describe the anisotropic electrical conduction throughout the tissue. Electrical signals will preferentially conduct along fibres, and conduct more slowly transversely to the fibre orientation. This is often represented[79] by the following equation

$$\mathbf{A} = a_f \underline{f} \otimes \underline{f} + a_s \underline{s} \otimes \underline{s} + a_n \underline{n} \otimes \underline{n} \tag{2.39}$$

where $\mathbf{A}$ is the conductivity tensor, and the vectors $\underline{f}$, $\underline{s}$, and $\underline{n}$ represent the mutually orthonormal fibre, sheet, and sheet-normal directions respectively, and a_f, a_s, and a_n are the conductances in those directions respectively per unit volume. Since

$$\underline{f} \otimes \underline{f} + \underline{s} \otimes \underline{s} + \underline{n} \otimes \underline{n} = \mathbf{I}$$

this allows for 2.39 to be simplified to

$$\mathbf{A} = a_n \mathbf{I} + (a_f - a_n) \underline{f} \otimes \underline{f} + (a_s - a_n) \underline{s} \otimes \underline{s} \tag{2.40}$$

So far, these quantities have been given in the deformed configuration, however, they are often only defined in the undeformed configuration. To do this, the vectors $\underline{f}$, and $\underline{s}$ are represented in terms of their values in the undeformed configuration, $\underline{\tilde{f}}$ and $\underline{\tilde{s}}$. This is achieved by the following equations

$$\underline{f} = \frac{\mathbf{F}(\underline{\tilde{f}})}{||\mathbf{F}(\underline{\tilde{f}})||}, \; \underline{s} = \frac{\mathbf{F}(\underline{\tilde{s}})}{||\mathbf{F}(\underline{\tilde{s}})||}, \tag{2.41}$$

which represent the normalized push-forward of the undeformed vectors $\underline{\tilde{f}}$ and $\underline{\tilde{s}}$, where $||\underline{a}|| = \sqrt{\sum_i {a^i}^2}$.

The equations 2.41 can be applied to 2.40 to determine the representation in the Cartesian basis

$$A_{IJ} = a_n\delta_{IJ} + (a_f - a_n)\frac{F_{IM}\tilde{f}^M F_{JN}\tilde{f}^N}{\tilde{f}^P C_{PQ}\tilde{f}^Q} + (a_s - a_n)\frac{F_{IM}\tilde{s}^M F_{JN}\tilde{s}^N}{\tilde{s}^P C_{PQ}\tilde{s}^Q} \tag{2.42}$$

where $\tilde{f}^M$ and $\tilde{s}^M$, are the cartesian components of the preferred vectors $\underline{\tilde{f}}$ and $\underline{\tilde{s}}$ in the reference frame. Through the pullback formula 2.34 can be converted into the reference frame[79]

$$\tilde{A}_{IJ} = Ja_nC^{-1}_{IJ} + J(a_f - a_n)\frac{\tilde{f}_I\tilde{f}_J}{\tilde{f}^P C_{PQ}\tilde{f}^Q} + J(a_s - a_n)\frac{\tilde{s}_I\tilde{s}_J}{\tilde{s}^P C_{PQ}\tilde{s}^Q} \tag{2.43}$$

where the factor of J accounts for changes in volume due to the deformation, since the conductance is defined per unit volume of tissue.

Often, the two transverse-to-fibre conductances are modelled as equal, $a_s = a_n = a_T$. This allows for a further simplification to

$$\tilde{A}_{IJ} = Ja_TC^{-1}_{IJ} + J(a_f - a_T)\frac{\tilde{f}_I\tilde{f}_J}{\tilde{f}^P C_{PQ}\tilde{f}^Q}. \tag{2.44}$$

This can be represented in the undeformed covariant basis by

$$\tilde{A}^{ij} = Ja_TG^{ij} + J(a_f - a_T)\frac{\tilde{f}^i\tilde{f}^j}{\tilde{f}^p G_{pq}\tilde{f}^q}. \tag{2.45}$$

where $\tilde{f}^k$ and $\tilde{s}^k$ now represent curvilinear covariant components of the preferred vectors, $\underline{\tilde{f}}$ and $\underline{\tilde{s}}$, in the reference frame.

2.4.2 Bidomain equations

A commonly used diffusion model is that of the bidomain equations. These are derived by assuming that cardiac tissue can be separated into an intracellular and extracellular domain[80]. The continuous and finite intracellular and extracellular domains occupy the same spatial region, $\Omega \subset \mathbb{R}^D$ where $D = 2$ or 3 is the number of spatial dimensions the tissue has, but are distinct from one another, separated by the cell membrane potential. It is assumed that electrical conductivity can differ between the extracellular and intracellular domains and that electrical charge and current are conserved. With these assumptions, it is possible to derive[80] the bidomain equations for the domain Ω to be,

$$C_m \frac{\partial V}{\partial t} + I_{\text{ion}} - I_{\text{i,source}} = \nabla \cdot (D \nabla (V + \phi)) \tag{2.46}$$

$$\nabla \cdot (D \nabla V) + \nabla \cdot ((D + E) \nabla \phi) - I_{\text{i,source}} - I_{\text{e,source}} = 0. \tag{2.47}$$

The boundary conditions are often chosen as insulating by enforcing the following

$$\underline{n} \cdot (D \nabla (V + \phi)) = 0 \tag{2.48}$$

$$\underline{n} \cdot (E \nabla (\phi)) = 0 \tag{2.49}$$

where $\underline{n}$ is the outward unit normal vector to the surface $\partial \Omega$. I_{ion} is the transmembrane ionic flux due to the ion channels and is given by 2.16.

It can be shown that a compatibility condition on the source terms $i_{\text{i,source}}$

and $i_{\text{e,source}}$ exists such that the boundary conditions are satisfied and that this condition is given by[81][82][83]

$$\int_{\Omega} (i_{\text{i,source}} + i_{\text{e,source}})\ \mathrm{d}x = 0. \tag{2.50}$$

This is satisfied by choosing the following constraint on the applied source terms

$$i_{\text{i,source}} = -i_{\text{e,source}} = i_{\text{source}}. \tag{2.51}$$

The above formulations of the Bidomain model are in the deformed configuration. That is the frame following the tissue as it deforms. Often it is advantageous to simulate these equations in the undeformed, or reference, configuration. This is achieved by performing the necessary "pull-backs" of the various quantities, and by representing the time-derivative using the ALE formulation. This results in the following formulation

$$JC_m\left(\frac{\partial \tilde{v}}{\partial t} - \frac{\partial \tilde{v}}{\partial x_I}\frac{\partial X_I}{\partial t}\right) + J(I_{\text{ion}} - I_{\text{source}}) = \frac{\partial}{\partial x_I}\left(\tilde{D}_{IJ}\frac{\partial(\tilde{v}+\tilde{\phi})}{\partial x_J}\right), \tag{2.52}$$

$$\frac{\partial}{\partial x_I}\left(\tilde{D}_{IJ}\frac{\partial \tilde{v}}{\partial x_J}\right) + \frac{\partial}{\partial x_I}\left((\tilde{D}_{IJ} + \tilde{E}_{IJ})\frac{\partial \tilde{\phi}}{\partial x_J}\right) = 0, \tag{2.53}$$

where the boundary conditions in the reference configuration are

$$n_I\left(\tilde{D}_{IJ}\frac{\partial(\tilde{v}+\tilde{\phi})}{\partial x_J}\right) = 0, \tag{2.54}$$

$$n_I\left(\tilde{E}_{IJ}\frac{\partial \tilde{\phi}}{\partial x_J}\right) = 0. \tag{2.55}$$

2.4.3 Monodomain equation

A common simplification applied to the Bidomain model is to assume that the intracellular and extracellular conductivity tensors are scalar multiples[84] [85], that is that

$$\tilde{D}_{IJ} = \lambda \tilde{E}_{IJ} \tag{2.56}$$

where λ is the spatially and temporally independent intracellular-extracellular conductivity ratio.

$$JC_m \left(\frac{\partial \tilde{v}}{\partial t} - \frac{\partial \tilde{v}}{\partial x_I} \frac{\partial X_I}{\partial t} \right) + J(I_{\text{ion}} - I_{\text{source}}) = \frac{\partial}{\partial x_I} \left(\frac{1}{1+\lambda} \tilde{D}_{IJ} \frac{\partial \tilde{v}}{\partial x_J} \right) \tag{2.57}$$

where a relabelling is often applied such that $\frac{1}{1+\lambda}\tilde{D}_{IJ} \to \tilde{D}_{IJ}$. Similarly, the boundary conditions can be simplified to

$$n_I \tilde{D}_{IJ} \frac{\partial \tilde{v}}{\partial x_J} = 0. \tag{2.58}$$

2.5 Solid mechanics

2.5.1 Strain tensor

Strain can be defined in a number of different ways depending on the application and the physical system being modelled. By far the most commonly used measure when discussing cardiac tissue is the Green strain tensor[86][12], which is defined in the Cartesian basis as

$$E_{IJ} = \frac{1}{2} \left(F_{KI} F_{KJ} - \delta_{IJ} \right) \tag{2.59}$$

This definition can be derived by considering the square lengths of an undeformed line, $\mathrm{d}s^2 = \underline{\mathrm{d}r} \cdot \underline{\mathrm{d}r}$, and its corresponding deformed length, $\mathrm{d}S^2 = \underline{\mathrm{d}R} \cdot \underline{\mathrm{d}R} = F\,\underline{\mathrm{d}r} \cdot F\,\underline{\mathrm{d}r}$.

The difference between these two quantities is given by

$$\mathrm{d}S^2 - \mathrm{d}s^2 = F_{IK}\,\mathrm{d}r_K F_{IJ}\,\mathrm{d}r_J - \mathrm{d}r_M\,\mathrm{d}r_M = (F_{IK}F_{IJ} - \delta_{KJ})\,\mathrm{d}r_K\,\mathrm{d}r_J. \quad (2.60)$$

This quantity vanishes when $F_{KI}F_{KJ} - \delta_{IJ} = 0$, since it is expected that there will be no strain within a body if the length of material lines is not changed from the undeformed configuration this motivates the definition 2.59 which provides an objective measure of strain.

The corresponding strain tensor relative to the undeformed contravariant basis vectors can be derived to be

$$\gamma_{ij} = \frac{1}{2}(G_{ij} - g_{ij}), \quad (2.61)$$

where line elements in the undeformed and deformed configurations have instead been defined in terms of their covariant basis vectors as $\underline{\mathrm{d}r} = \underline{g}_i\,\mathrm{d}\xi^i$ and $\underline{\mathrm{d}r} = \underline{G}_i\,\mathrm{d}\xi^i$ respectively.

It can be shown using the relationships described in 2.25 that these quantities are in fact equivalent by considering the following arguments,

$$\begin{aligned}
E_{IJ}\underline{e}^I \otimes \underline{e}^J &= \frac{1}{2}\left(F_{KI}F_{KJ} - \delta_{IJ}\right)\underline{e}^I \otimes \underline{e}^J && (2.62)\\
&= \frac{1}{2}\left(\frac{\partial X^K}{\partial x^I}\frac{\partial X^K}{\partial x^J} - \delta_{IJ}\right)\frac{\partial x^I}{\partial \xi^i}\underline{g}^i \otimes \frac{\partial x^J}{\partial \xi^j}\underline{g}^j && (2.63)\\
&= \frac{1}{2}\left(\frac{\partial X^K}{\partial x^I}\frac{\partial X^K}{\partial x^J}\frac{\partial x^I}{\partial \xi^i}\frac{\partial x^J}{\partial \xi^j} - \delta_{IJ}\frac{\partial x^I}{\partial \xi^i}\frac{\partial x^J}{\partial \xi^j}\right)\underline{g}^i \otimes \underline{g}^j && (2.64)\\
&= \frac{1}{2}\left(\frac{\partial X^K}{\partial \xi^i}\frac{\partial X^K}{\partial \xi^j} - \frac{\partial x^I}{\partial \xi^i}\frac{\partial x^I}{\partial \xi^j}\right)\underline{g}^i \otimes \underline{g}^j && (2.65)\\
&= \frac{1}{2}\left(G_{ij} - g_{ij}\right)\underline{g}^i \otimes \underline{g}^j && (2.66)\\
&= \gamma_{ij}\underline{g}^i \otimes \underline{g}^j && (2.67)
\end{aligned}$$

The following derivatives of the metric tensors with respect to the strain tensor are useful when computing components of the stress tensor,

$$\begin{aligned}
\frac{\partial g_{mk}}{\partial \gamma_{ij}} &= 0 && (2.68)\\
\frac{\partial g^{mk}}{\partial \gamma_{ij}} &= 0 && (2.69)\\
\frac{\partial G_{mk}}{\partial \gamma_{ij}} &= 2\delta^i_m\delta^j_k && (2.70)\\
\frac{\partial G^{mk}}{\partial \gamma_{ij}} &= -2G^{mi}G^{kj}. && (2.71)
\end{aligned}$$

Where 2.68 and 2.69 follow since the undeformed metric tensors do not vary with any quantity once the undeformed configuration has been defined.

2.5.2 Strain invariants

The deformed metric tensor has only real components and is symmetric, this implies that it has real eigenvalues μ and mutually orthogonal eigenvectors $\underline{v}$[75]. Such eigenvectors and eigenvalues are given by the equation $G_{ij}v^j =$

μv_i the strain invariants are therefore the solutions to the equation[75]

$$\det\left(G_{ij} - \mu g_{ij}\right) = 0$$

and are given by[87]

$$I_1 = g^{ik} G_{ki}, \tag{2.72}$$

$$I_2 = \frac{1}{2}\left(I_1^2 - g^{ik} G_{kj} g^{jn} G_{ni}\right), \tag{2.73}$$

$$I_3 = \frac{\det \mathbf{G}}{\det \mathbf{g}}. \tag{2.74}$$

Since these quantities are invariant under coordinate transform, and since the behaviour of a material should not depend on the coordinate system, they constitute an important part of constitutive modelling[75].

These can also be expressed in the cartesian basis as

$$I_1 = C_{KK}, \tag{2.75}$$

$$I_2 = \frac{1}{2}\left(I_1^2 - (C_{KK})^2\right), \tag{2.76}$$

$$I_3 = \det\left(C_{IJ}\right). \tag{2.77}$$

2.5.3 The principle of virtual displacements

Through consideration of conservation laws and variational principles, the governing equations of solid deformation can be derived. However, the details of the derivation of these governing equations are surplus to their implementation within the Oomph-lib framework.

Applied surface traction $\underline{T}$ is a force per unit deformed surface area on some part of the body's deformed surface $A_{\text{tract}} \subset \partial\Omega$[75]. The applied body force, $\underline{\text{f}}$, is the force per unit volume of the undeformed body[75].

It is assumed that displacements are prescribed on the remaining parts of the body surface not subject to surface traction, so that

$$\underline{R}(\xi^i) = \underline{R}^{(BC)}(\xi^i) \quad \text{on} \quad A_{\text{displ}} \subset \partial\Omega,$$

where $A_{\text{tract}} \cap A_{\text{displ}} = 0$ and $A_{\text{tract}} \cup A_{\text{displ}} = \partial\Omega$, and $\underline{R}^{(BC)}(\xi^i)$ are prescribed.

The Principle of Virtual Displacements (PVD) can be derived[75] to be

$$\int\limits_{\Omega} \left\{ s^{ij}\delta\gamma_{ij} - \left(\underline{\mathrm{f}} - \rho\frac{\partial^2 \underline{R}}{\partial t^2} \right) \cdot \delta\underline{R} \right\} \mathrm{d}v - \oint\limits_{A_{\text{tract}}} \underline{T} \cdot \delta\underline{R}\, \mathrm{d}A = 0 \tag{2.78}$$

where s^{ij} is the second Piola Kirchhoff stress tensor. The variation of the strain tensor is

$$\delta\gamma_{ij} = \frac{1}{2}\left(\frac{\partial \underline{R}}{\partial \xi^i} \cdot \delta\frac{\partial \underline{R}}{\partial \xi^j} + \delta\frac{\partial \underline{R}}{\partial \xi^i} \cdot \frac{\partial \underline{R}}{\partial \xi^j} \right). \tag{2.79}$$

Since the second Piola Kirchhoff stress tensor is symmetric, the principle of virtual displacements 2.78 can be re-written in displacement form as

$$\int\limits_{\Omega} \left\{ s^{ij}\frac{\partial \underline{R}}{\partial \xi^i} \cdot \delta\frac{\partial \underline{R}}{\partial \xi^j} - \left(\underline{\mathrm{f}} - \rho\frac{\partial^2 \underline{R}}{\partial t^2} \right) \cdot \partial\underline{R} \right\} \mathrm{d}v - \oint\limits_{A_{\text{tract}}} \underline{T} \cdot \delta\underline{R}\, \mathrm{d}A = 0. \tag{2.80}$$

This equation must be augmented by a constitutive equation which determines s^{ij} in terms of potentially many quantities including the deformation of the body[75]. For elastic solids, however, the stress is a function only of the strain[75].

2.5.4 Constitutive modelling of anisotropic hyperelastic solids

Hyperelastic solids are defined in terms of a strain energy function which, for the sake of objectivity is often written in terms of the strain invariants $W = W(I_k)$.

Most constitutive laws are presented on the Cartesian basis, however, the implementation of hyperelastic solids in Oomph-lib requires them in terms of curvilinear coordinates. It is often simpler to work in terms of the Cartesian basis for the majority of derivations and then convert the second Piola-Kirchhoff stress tensor to a curvilinear basis at the end via the following relationship

$$s^{ij} = \frac{\partial W}{\partial \gamma_{ij}} = \frac{\partial W}{\partial I_k}\frac{\partial I_k}{\partial \gamma_{ij}}. \tag{2.81}$$

Cardiac tissue is inherently anisotropic[87]. Muscle fibres are connected via the extracellular matrix and are arranged into sheets. The stress-strain relationship of the tissue in the sheet plane is observed to be different to the stress-strain relationship normal to the sheets[87][88].

The inclusion of anisotropy to the solid model introduces several new strain invariants which correspond to the fibre, sheet, and normal directions[87].

For unit vectors $\underline{a}$ and $\underline{b}$ in the undeformed configuration several invariants under body deformation can be derived[87] to be

$$I_{4,a} = \underline{a} \cdot C\underline{a}, \tag{2.82}$$

$$I_{4,b} = \underline{b} \cdot C\underline{b}, \tag{2.83}$$

$$I_{5,a} = \underline{a} \cdot (C^2\underline{a}), \tag{2.84}$$

$$I_{5,b} = \underline{b} \cdot (C^2\underline{b}), \tag{2.85}$$

$$I_{8,ab} = \underline{a} \cdot C\underline{b}, \tag{2.86}$$

where a and b are each one of the prefered directions f, s, or n.

Note that since $I_{8,ab}$ can change sign if the direction of either $\underline{a}$ or $\underline{b}$ is reversed, it is not strictly invariant, for this reason it often appears in constitutive laws in either the form $I_{8,ab}^2$ or sometimes $I_{8,ab}\underline{a} \cdot \underline{b}$ provided that $\underline{a} \cdot \underline{b} \neq 0$[87].

The myocardium has three preferred directions, the fibre, $\underline{f}$, sheet, $\underline{s}$, and sheet-normal, $\underline{n}$ directions. However, by definition, these three vectors are considered to form an orthonormal basis. It, therefore, follows that

$$\underline{f} \otimes \underline{f} + \underline{s} \otimes \underline{s} + \underline{n} \otimes \underline{n} = \mathbf{I} \tag{2.87}$$

where $\mathbf{I}$ is the identity. Hence, $I_{4,n}$ can be rewritten in terms of $I_{4,f}$ and $I_{4,s}$. It is also possible to write $I_{5,f}$, $I_{5,s}$, and $I_{5,n}$ in terms of the other invariants[87].

It, therefore, follows that there are 4 additional invariants due to anisotropic preferential vectors[87], the full list of which is given in the

Cartesian basis by

$$I_{4,f}(F) = F_{KM}F_{KM}f_Mf_N, \tag{2.88}$$

$$I_{4,s}(F) = F_{KM}F_{KM}s_Ms_N, \tag{2.89}$$

$$I_{8,fs}(F) = F_{KM}F_{KM}f_Ms_N, \tag{2.90}$$

$$I_{8,sn}(F) = F_{KM}F_{KM}s_Mn_N, \tag{2.91}$$

where $f = f_M(\xi^i)\underline{e}^M$, $s = s_M(\xi^i)\underline{e}^M$, and $n = n_M(\xi^i)\underline{e}^M$ are the fibre, sheet, and sheet-normal directions in the undeformed configuration.

Written in the curvilinear basis the preferential vectors can be written as $\underline{f} = f^k\underline{g}_k$, $\underline{s} = s^k\underline{g}_k$, and $\underline{n} = n^k\underline{g}_k$. Along with the original "isotropic" strain invariants, the complete list of invariants used in this anisotropic constitutive modelling is

$$I_1 = g^{mk}G_{mk}, \tag{2.92}$$

$$I_2 = \frac{1}{2}\left(I_1^2 - g^{mk}G_{kr}g^{rn}G_{mn}\right), \tag{2.93}$$

$$I_3 = \frac{\det G}{\det g}, \tag{2.94}$$

$$I_{4,f} = G_{mn}f^mf^n, \tag{2.95}$$

$$I_{4,s} = G_{mn}s^ms^n, \tag{2.96}$$

$$I_{8,fs} = G_{mn}(f^ms^n + s^mf^n), \tag{2.97}$$

$$I_{8,sn} = G_{mn}(s^mn^n + n^ms^n). \tag{2.98}$$

The derivatives of the complete list of invariants with respect to the Green strain tensor are as follows

$$\frac{\partial I_1}{\partial \gamma_{ij}} = 2g^{ij}, \tag{2.99}$$

$$\frac{\partial I_2}{\partial \gamma_{ij}} = 2\left(I_1 g^{ij} - g^{im}g^{jn}G_{mn}\right), \tag{2.100}$$

$$\frac{\partial I_3}{\partial \gamma_{ij}} = 2I_3 G^{ij}, \tag{2.101}$$

$$\frac{\partial I_{4,f}}{\partial \gamma_{ij}} = 2f^i f^j, \tag{2.102}$$

$$\frac{\partial I_{4,s}}{\partial \gamma_{ij}} = 2s^i s^j, \tag{2.103}$$

$$\frac{\partial I_{8,fs}}{\partial \gamma_{ij}} = 2(f^i s^j + s^i f^j), \tag{2.104}$$

$$\frac{\partial I_{8,sn}}{\partial \gamma_{ij}} = 2(s^i n^j + n^i s^j). \tag{2.105}$$

Where the result 2.101 can be derived through the application of the chain rule, Jacobi's formula[89], and the definition of a matrix adjugate[89]

$$\frac{\partial \frac{\det G}{\det g}}{\partial \gamma_{ij}} = \frac{1}{\det g}\frac{\partial \det G}{\partial G_{kl}}\frac{\partial G_{kl}}{\partial \gamma_{ij}} = \frac{1}{\det g}\mathrm{adj}(G)_{lk} 2\delta^i_k \delta^j_l = \frac{\det G}{\det g}(G^{-1})_{ij}. \tag{2.106}$$

Since for, metric tensors, $G_{ij}G^{jk} = \delta^k_i$ the result 2.101 follows.

Alternatively, as per the derivation in[75], the determinant of G_{ij} can be re-written in terms of the Levi-Civita symbol[75] by first noting that

$$G_{ij} = g_{ij} + 2\gamma_{ij} \tag{2.107}$$

and that

$$\det G = \frac{1}{6}\epsilon^{mnl}\epsilon^{pqr}G_{mp}G_{nq}G_{lr} \tag{2.108}$$

then, by the chain rule, the result 2.101 follows.

Following from 2.81 and using 2.99-2.105 the expanded form of of the second Piola-Kirchhoff stress tensor in terms of derivatives of the invariants is

$$\begin{aligned}\frac{\partial W}{\partial \gamma_{ij}} &= 2g^{ij}\frac{\partial W}{\partial I_1} + 2\left(I_1 g^{ij} - g^{im}g^{jn}G_{mn}\right)\frac{\partial W}{\partial I_2} + 2I_3 G^{ij}\frac{\partial W}{\partial I_3} \\ &+ 2f^i f^j \frac{\partial W}{\partial I_{4,f}} + 2s^i s^j \frac{\partial W}{\partial I_{4,s}} \\ &+ 2(f^i s^j + s^i f^j)\frac{\partial W}{\partial I_{8,fs}} + 2(s^i n^j + n^i s^j)\frac{\partial W}{\partial I_{8,sn}}.\end{aligned} \tag{2.109}$$

2.5.5 Incompressibility constraint

The myocardium constitutes mostly water-filled cells connected by an extra-cellular matrix of macromolecules, such as collagen and elastin filled with intracellular fluid. Since water comprises the majority of cardiac muscle and water is essentially incompressible, cardiac muscle is often considered to be incompressible. The myocardium however also contains many small blood vessels which are ingrained throughout the tissue. These blood vessels are observed to drain during systole which allows the total volume of the myocardium to decrease during contraction. However, these effects are often ignored during modelling due to the mathematical and numerical complexity of the arising poroelastic solid mechanics[90].

Assuming incompressibility, it follows that the volume of infinitesimal elements does not change between the undeformed and deformed configurations[75][87]. It can therefore be written that

$$\det G_{ij} - \det g_{ij} = 0. \tag{2.110}$$

In this case

$$I_3 \equiv 0 \implies \frac{\partial I_3}{\partial \gamma_{ij}} \equiv 0. \tag{2.111}$$

Stress can be separated into a deviatoric, $\bar{\sigma}^{ij}$, and hydrostatic part, $\frac{1}{3}\delta^{ij}\sigma^{kk}$, where the hydrostatic part reflects changes in volumes and written as

$$s^{ij} = \bar{s}^{ij} + \frac{1}{3}\delta^{ij}s^{kk}. \tag{2.112}$$

In order to solve for the deformation with incompressibility constraints, the hydrostatic stress is subtracted from the second Piola-Kirchhoff stress tensor and the incompressibility constraint 2.110 is enforced via a Lagrange multiplier, p, which represents the pressure. The second Piola-Kirchhoff stress tensor used in the PVD 2.78 becomes

$$s^{ij} = \frac{\partial W}{\partial \gamma_{ij}} - \frac{1}{3}\delta^{ij}\frac{\partial W}{\partial \gamma_{kk}} - pG^{ij} \tag{2.113}$$

where p is determined by enforcing the constraint 2.110.

2.5.6 The passive myocardium

Numerous constitutive models of the passive myocardium exist. Several of these models are implemented in the additions made to Oomph-lib as part of this work, along with some simpler models.

The neo-Hookean model is an isotropic passive tissue model and has a strain energy function

$$W = \frac{\mu}{2}\left(I_1 - \delta^i_i\right). \tag{2.114}$$

Exponential strain energy functions can take the form

$$W = \frac{a}{2b}\exp\big(b(I_1 - \delta^i_i)\big). \tag{2.115}$$

Alternative strain energy functions have been proposed in terms of the Green strain tensor decomposed into the preferential vector directions $\underline{f} = \underline{v}^{(0)} = v^{(0)k}\underline{g}_k$, $\underline{s} = v^{(1)k}\underline{g}_k$, and $\underline{n} = v^{(2)k}\underline{g}_k$ which have been labelled numerically to ease the following notation. Such a decomposition is performed by computing the following

$$E_{ij} = \gamma_{pq} v^{(i)p} v^{(j)q}. \tag{2.116}$$

Pole-Zero strain energy function is widely used[91][92][93] and can be written in terms of the above decomposition of the Green strain as follows

$$W = \sum_{i=0}^{i=2} \sum_{j=i}^{j=2} \frac{k_{ij} E_{ij}^2}{|a_{ij} - |E_{ij}||^{b_{ij}}}. \tag{2.117}$$

This model is derived from bi-axial tension tests and is considered unlikely to be suitable for use in simulating other forms of deformation[87].

Holzapfel and Ogden proposed[87] a physically detailed model for strain energy in the form

$$\begin{aligned} W = \frac{a}{2b} \exp\left(b(I_1 - \delta_i^i)\right) + \sum_{i=f,s} \frac{a_i}{2b_i} \left(\exp\left(b_i (I_{4,i} - 1)^2\right) - 1\right) \\ + \frac{a_{fs}}{2b_{fs}} \left(\exp\left(b_{fs} I_{8,fs}\right)^2 - 1\right). \end{aligned} \tag{2.118}$$

2.5.7 The active stress decomposition

A common approach to the inclusion of active stress in the stress-strain relationship is to simply add the active stress generated by the single cells to the passive stress generated by the deformation of the body.

Here, the second Piola-Kirchhoff stress tensor will be given by

$$s^{ij} = s^{P^{ij}} + s^{A^{ij}}, \tag{2.119}$$

where the passive Piola-Kirchhoff stress tensor, $s^{P^{ij}}$, is given by the constitutive law and active Piola-Kirchhoff stress tensor $s^{A^{ij}}$ is described through the equation

$$s^{A^{ij}} = T^A f^i f^j, \tag{2.120}$$

where T^A is the active stress generated by sarcomere shortening, and f^i are the covariant components of the fibre direction in the undeformed configuration. From equation 2.113 it follows that if the tissue is incompressible, the second Piola-Kirchhoff stress tensor is given by

$$s^{ij} = (S^{P^{ij}} - \frac{1}{3}\delta^{ij} S^{P^{kk}}) + (T^A f^i f^j - \frac{1}{3}\delta^{ij} T^A) - pG^{ij} \tag{2.121}$$

The active stress decomposition has been criticised for the phenomenological manner in which forces due to cardiomyocyte contraction are incorporated into the stress tensor[94].

2.5.8 The active strain decomposition

An alternative approach is to consider the active and passive deformations to be two separate virtual components of the final deformation[94].

In such models, the final deformation map $\mathbf{F}$ is decomposed into two virtual deformations, the elastic deformation $\tilde{\mathbf{F}}$ and the active deformation $\hat{\mathbf{F}}$, and is written as follows

$$\mathbf{F} = \tilde{\mathbf{F}}\hat{\mathbf{F}}. \tag{2.122}$$

In general, it is not possible to represent the quantities $\hat{\mathbf{F}}$ and $\tilde{\mathbf{F}}$ in terms of the gradient of a vector mapping since they represent virtual, intermediate deformations, and will therefore not, in general, be conservative. However, the final deformation gradient tensor is given by $F_{IJ} = \frac{\partial X_I}{\partial x_J}$ since it represents the physically deformed conformation of the tissue. In order to determine the invariants of the elastic deformation, it is necessary to calculate $\tilde{\mathbf{F}} = \mathbf{F}\hat{\mathbf{F}}^{-1}$.

Through considering the deformation of a single cardiomyocyte in response to sarcomere shortening/lengthening, $\hat{\mathbf{F}} = \hat{F}_{IJ}(\underline{e}^I \otimes \underline{e}^J)$ may be constructed as

$$\hat{F}_{IJ} = \delta_{IJ} + \gamma_f f_I f_J + \gamma_s s_I s_J + \gamma_n n_I n_J \tag{2.123}$$

where γ_f, γ_s, and γ_n are the strain in the fibre, sheet, and sheet-normal directions respectively generated by changes in sarcomere length.

Through fundamental results of linear algebra, the determinant and inverse of $\hat{\mathbf{F}}$ can be determined to be

$$det\hat{\mathbf{F}} = (1+\gamma_f)(1+\gamma_s)(1+\gamma_n), \tag{2.124}$$

$$\hat{F}^{-1}_{IJ} = \delta_{IJ} - \frac{\gamma_f}{1+\gamma_f} f_I f_J - \frac{\gamma_s}{1+\gamma_s} s_I s_J - \frac{\gamma_n}{1+\gamma_n} n_I n_J. \tag{2.125}$$

The final elastic deformation is then given by

$$\tilde{F}_{IJ} = F_{IK}\left(\delta_{KJ} - \frac{\gamma_f}{1+\gamma_f} f_K f_J - \frac{\gamma_s}{1+\gamma_s} s_K s_J - \frac{\gamma_n}{1+\gamma_n} n_K n_J\right). \tag{2.126}$$

The cytosol of cardiomyocytes is mostly water which is itself incompressible, this leads to the commonly applied assumption that the cardiomyocytes are isochoric (incompressible). Furthermore, for simplicity, cardiomyocytes are often modelled as transversely isotropic cylinders, because there is no distinction at the cellular level between the deformation of the cell in the sheet and in the sheet-normal directions. It follows from this assumption that $\gamma_s = \gamma_n$. Since the cardiomyocyte is assumed to be isochoric, conservation of volume during cellular contraction implies that $det\hat{\mathbf{F}} = \hat{J} = 1$. It can then be determined that $\gamma_s = \gamma_n = \frac{1}{\sqrt{1+\gamma_f}} - 1$. These modelling assumptions allow for the simplification of $\tilde{F}_{IJ}$ to

$$\tilde{F}_{IJ} = F_{IK}\left(\delta_{KJ}\sqrt{1+\gamma_f} + f_K f_J\left(\frac{1}{1+\gamma_f} - \sqrt{1+\gamma_f}\right)\right). \tag{2.127}$$

In order to describe the stresses induced in hyperelastic solids under this strain decomposition, we seek the strain invariants of the elastic deformation $\tilde{\mathbf{F}}$. By defining the quantity

$$g(\gamma_f) = -\gamma_f - \gamma_f \frac{\gamma_f + 2}{(\gamma_f + 1)^2} \tag{2.128}$$

the strain invariants can be succinctly written as

$$I_1^E = I_1(\tilde{\mathbf{F}}) \qquad = (1+\gamma_f)F_{JK}F_{JK} + g(\gamma_f)F_{JK}F_{JM}f_K f_M, \tag{2.129}$$

$$I_3^E = I_3(\tilde{\mathbf{F}}) \qquad = J^2 = I_3, \tag{2.130}$$

$$I_{4,f}^E = I_{4,f}(\tilde{\mathbf{F}}) \qquad = \frac{1}{(1+\gamma_f)^2} F_{PM}F_{PN}f_M f_N, \tag{2.131}$$

$$I_{4,s}^E = I_{4,s}(\tilde{\mathbf{F}}) \qquad = (1+\gamma_f)\, F_{PM}F_{PN}s_M s_N, \tag{2.132}$$

$$I_{8,fs}^E = I_{8,fs}(\tilde{\mathbf{F}}) \qquad = \frac{1}{2\sqrt{1+\gamma_f}} F_{PM}F_{PN}(f_M s_N + f_N s_M), \tag{2.133}$$

$$I_{8,sn}^E = I_{8,sn}(\tilde{\mathbf{F}}) \qquad = (1+\gamma_f)\, F_{PM}F_{PN}(s_M n_N + s_N n_M). \tag{2.134}$$

The invariant I_2^E is omitted since it is not used in any of the example constitutive relations. The third invariant can be determined by considering that

$$\begin{aligned}
I_3^E = \det \tilde{C}_{IJ} &= \det (\tilde{F}_{KI} \tilde{F}_{KJ}) \\
&= \det (\tilde{F}_{KI}) \det (\tilde{F}_{KJ}) \\
&= \det (F_{KM} \tilde{F}^{-1}_{MI}) \det (F_{KN} \tilde{F}^{-1}_{NJ}) \\
&= \det (F_{KM}) \det (\tilde{F}^{-1}_{MI}) \det (F_{KN}) \det (\tilde{F}^{-1}_{NJ}) \\
&= J^2 \hat{J}^{-1} \\
&= J^2 = I_3.
\end{aligned}$$

In order to calculate these in the environment of the Oomph-lib constitutive law class, they must be converted to curvilinear coordinates, they can be re-written as

$$I_1^E = (1+\gamma_f) G_{mk} g^{mk} + g(\gamma_f) G_{mk} f^m f^k \tag{2.135}$$

$$I_3^E = \frac{\det G}{\det g} \tag{2.136}$$

$$I_{4,f}^E = \frac{1}{(1+\gamma_f)^2} G_{mk} f^m f^k \tag{2.137}$$

$$I_{4,s}^E = (1+\gamma_f)\, G_{mk} s^m s^k \tag{2.138}$$

$$I_{8,fs}^E = \frac{1}{2\sqrt{1+\gamma_f}} G_{mk} (f^m s^k + f^k s^m) \tag{2.139}$$

$$I_{8,sn}^E = (1+\gamma_f)\, G_{mk} (s^m n^k + s^k n^m) \tag{2.140}$$

$$\tag{2.141}$$

where, since J^A does not vary with strain.

Derivatives of the strain invariants with respect to the Green strain tensor are given by the following,

$$\frac{\partial I_1^E}{\partial \gamma_{ij}} = 2(1+\gamma_f)g^{ij} + 2g(\gamma_f)f^i f^j, \tag{2.142}$$

$$\frac{\partial I_3^E}{\partial \gamma_{ij}} = 2I_3 G^{ij}, \tag{2.143}$$

$$\frac{\partial I_{4,f}^E}{\partial \gamma_{ij}} = \frac{2}{(1+\gamma_f)^2} f^i f^j, \tag{2.144}$$

$$\frac{\partial I_{4,s}^E}{\partial \gamma_{ij}} = 2\left(1+\gamma_f\right) s^i s^j, \tag{2.145}$$

$$\frac{\partial I_{8,fs}^E}{\partial \gamma_{ij}} = \frac{1}{\sqrt{1+\gamma_f}}(f^i s^j + f^j s^i), \tag{2.146}$$

$$\frac{\partial I_{8,sn}^E}{\partial \gamma_{ij}} = 2\left(1+\gamma_f\right)(s^i n^j + n^i s^j). \tag{2.147}$$

The relationship 2.143 and 2.110-2.113 imply that when applied to stress derived through the active strain decomposition, the incompressibility constraints remain unchanged. This is because the incompressibility condition $\frac{\partial I_3}{\partial \gamma_{ij}} = 0$ implies that $\frac{\partial I_3^E}{\partial \gamma_{ij}} = 0$.

The second Piola-Kirchhoff stress tensor for the active strain decomposition is given by

$$\begin{aligned} s^{ij} = & \frac{\partial W}{\partial I_1^E} 2(1+\gamma_f)g^{ij} && + 2g(\gamma_f)f^i f^j \\ & + \frac{\partial W}{\partial I_2^E}\frac{\partial I_2^E}{\partial \gamma_{ij}} && + \frac{\partial W}{\partial I_3^E} 2I_3 G^{ij} \\ & + \frac{\partial W}{\partial I_{4,f}^E}\frac{2}{(1+\gamma_f)^2} f^i f^j && + \frac{\partial W}{\partial I_{4,s}^E} 2\left(1+\gamma_f\right) s^i s^j \\ & + \frac{\partial W}{\partial I_{8,fs}^E}\frac{1}{\sqrt{1+\gamma_f}}(f^i s^j + f^j s^i) && + \frac{\partial W}{\partial I_{8,sn}^E} 2\left(1+\gamma_f\right)(s^i n^j + n^i s^j). \end{aligned} \tag{2.148}$$

A potential advantage of the active strain decomposition formulation over the active stress formulation is that the resulting simulation of strain-mediated ion channels may be more accurate. Due to a large number of DOFs and the long time-scale associated with solving for solid deformation, their solution is often calculated in a partitioned manner from the electrical model. As such, the deformation of the solid cannot be updated in response to changes in the cell model. For the active stress formulation, this forces the cell model to either assume that tissue strain is given entirely by the cardiomyocyte, ignoring the effects of the continuum solid model, or to assume that tissue strain is constant and given by the deformation of the solid. The active strain decomposition formulation however does not impose such a concession, since the strain in the tissue due to the active strain formulation can be allowed to change in response to varying lengths in the absence of deformation of the solid. This follows from equation 2.127.

Since it may be advantageous to implement constitutive laws using the active strain decomposition it was therefore elected that the implementation of anisotropic solid mechanics into Oomph-lib allows for the use of active strain decomposition as well as the more commonly used active stress decomposition. This is achieved, without requiring any substantial changes to the implementation of active stress, by treating the active stress and active strain as the same generic quantity. In this manner, active strain or active stress models can be simulated within the same finite elements.

Chapter 3

Numerical methods

Numerical methods are required to solve the equations which were described in the previous chapter. For certain aspects of this solution, the choice of the numerical method used is restricted by the general methodologies imposed by Oomph-lib. For example, Oomph-lib is highly specialised to the finite element method for the solution of PDEs and therefore the use of other methods such as Finite Volume Method (FVM) or Finite Difference Method (FDM) would be better suited to other numerical platforms.

Certain commonly used methods can often be divergent from the standard methodology and application of Oomph-lib. For example, operator splitting methods are commonly used to improve the efficiency of numerical computation, particularly of the electrophysiology problem. These methods, however, are at odds with the monolithic nature of the Oomph-lib solve routines. Such methods were identified and methods of implementing them were explored. The most suitable method was then implemented in the additions to the library.

3.1 The finite element method

The solution of all implemented PDEs is computed with the Finite Element Method (FEM). FEM is chosen through necessity since Oomph-lib is highly specialised for this rather than FDM or FVM.

The Discrete Element Method (DEM) has recently been used to include effects which arise due to the distinct nature of cardiomyocytes. The framework necessary for implementing such a system is very different to that of the finite element method and therefore from that of Oomph-lib. However, the framework set out in later sections for how cardiomyocytes are handled could facilitate future implementations of the DEM. These implementations are discussed in the chapter on future work.

A thorough discussion of the principles and implementation of the finite element method can be found within the Oomph-lib documentation[16] however, for completeness, a brief overview of the most relevant aspects is given here.

The FEM is predicated on efficiently approximating a function using basis functions which have finite support, that is which are zero over the majority of the domain[16].

Computation of integrals can be performed by summing the contribution from each of the finite elements which make up the domain. The contribution from each element can be computed using high-order and efficient numerical approximations, such as Newton-Cotes.

3.2 Basis functions

Basis functions, also called shape functions within the Oomph-lib environment, represent functions by providing a basis for the function space. These basis functions could be any functions which form a complete base over the domain to which the function is being represented, however, in the finite element method it is often useful to consider functions associated with each node of the mesh. Each node in the mesh has associated with it a basis function which has a value of 1 at the location of that node and zero at the locations of all other nodes. Written more rigorously, each basis function satisfies

$$\psi_j(X_i) = \delta_{ij} \tag{3.1}$$

where ψ_j is the j$^{\text{th}}$ basis function, X_i is the coordinate of the i$^{\text{th}}$ node in the mesh, and δ_{ij} is the Dirac delta function. Representation of a quantity is then performed by taking a sum of the contributions due to each node in the mesh such that a quantity u at the global coordinate $\underline{x}$ is given by

$$u(\underline{x}) = \sum_{i=1}^{N} U_i \psi_i(\underline{x}) = U_i \psi_i(\underline{x}) \tag{3.2}$$

where U_i is the local value at the i$^{\text{th}}$ node in the mesh. As such the shape functions provide an interpolation from the nodes of the finite elements to the interior of those elements.

Depending on the implementation, there are certain restrictions on the basis functions which can be used. Specifically, restrictions on how minimally differentiable and integrable a function must arise from the form of the residual and Jacobian functions for the equations being solved.

Within FEM, basis functions are often chosen which have finite support, that is the basis associated with a particular node is only non-zero within elements that contain that node. This ensures that, when considering interpolations and integrals within a particular element, only the basis functions of nodes within that element have to be considered.

The most commonly used set of basis functions in Oomph-lib is Lagrange polynomials. For a set of N points s_i, the Lagrange polynomials are given by

$$\ell_j(s) = \prod_{i=0, i\neq j}^{i=N} \frac{s - s_i}{s_j - s_i} \tag{3.3}$$

which are used to construct the Lagrange interpolating polynomial, $L_y(s)$ through a corresponding set of points $y_i = y(s_i)$:

$$L_y(s) = y_j \ell_j(s). \tag{3.4}$$

In FEM, for a particular element, the set of points s_i are chosen as the local coordinates of the nodes within the element. N is then the number of nodes within the element and the local interpolation of the solution y within an element e is then given by 3.4 for the local nodal values and basis functions within that element only.

3.2.1 Local to global mapping

Given a local coordinate $\underline{s}$ within an element, the corresponding global coordinate $\underline{x}$ can be computed through a local to global mapping. The simplest choice of such a mapping[16] is to simply use the local basis functions to interpolate the global coordinates of the nodes within the element

$$\underline{x}(\underline{s}) = \sum_{j=1}^{n} X_{j,e} \psi_j(\underline{s}). \tag{3.5}$$

3.2.2 Spatial derivatives

Spatial derivatives of this value can be calculated by calculating the derivatives of the shape functions

$$\frac{\partial u(x)}{\partial x_j} = U_i \frac{\partial \psi_i(x)}{\partial x_j}. \tag{3.6}$$

Spatial derivatives of quantities are calculated by differentiating the basis functions with respect to the global coordinate:

$$\frac{\partial u(\underline{s})}{\partial x_i} = U_j \frac{\partial \psi_j(\underline{s})}{\partial x_i} = U_j \frac{\partial \psi_j(\underline{x})}{\partial s_m}\frac{\partial s_m}{\partial x_i}. \tag{3.7}$$

The quantity $\frac{\partial s_m}{\partial x_i}$ is the inverse of the Jacobian of the local to Eulerian mapping. It is calculated by constructing

$$\frac{\partial s_i}{\partial x_j} = \left(\frac{\partial x_j}{\partial s_i}\right)^{-1} = \left(x_j^L \frac{\partial \psi^L}{\partial s_i}\right)^{-1}. \tag{3.8}$$

These calculations are performed within Oomph-lib when spatial derivative functions are called and do not have to be calculated explicitly when implementing new finite element formulations.

3.3 Approximating integrals

Spatial integrals which arise in physical models are often difficult or impossible to compute analytically. They are therefore often evaluated with numerical approximations. In Oomph-lib this is commonly performed using Gaussian quadrature[16][95]. Gaussian quadrature evaluates integrals by taking a weighted sum of the function evaluated at a number of points within the region, D. The general form of Gaussian quadrature is given by

$$\int_D F(\underline{x})\,\mathrm{d}x \approx \sum_{i=1}^{N_{\text{int}}} W_i F(\underline{x}_i) \tag{3.9}$$

where N_{int} is the number of integral points, W_i is the weight associated with integral point i, and $\underline{x}_i$ is the coordinate of the i-th integral point.

3.3.1 General implementation of integration in Oomph-lib

In FEM a domain D is split into N elements each enclosing a region $\mathcal{E}_i$ such that the total region of those elements is the domain D, that is $\mathcal{E}_0 \cup \cdots \cup \mathcal{E}_N = D$. Approximating an integral over the domain D is then computed by taking a sum of the contributions from each element in the mesh. The integral within the i$^{\text{th}}$ element is given by

$$\int\limits_{\mathcal{E}_i} F(\underline{x})\,\mathrm{d}x \approx \sum_{i_{\text{int}}} J_i w_{i_{\text{int}}} F(s_{i_{\text{int}}}) \tag{3.10}$$

where $s_{i_{\text{int}}}$ and $w_{i_{\text{int}}}$ are the local coordinate and integral weight for the ${i_{\text{int}}}^{\text{th}}$ integral point and are provided by the numerical integration scheme of the finite element. J_i represents the scaling of the integral due to the size of the element and is calculated from the Jacobian of the local to global coordinate mapping.

3.4 Time integration

Depending on the specific situation, several approximations of time derivatives are used throughout Oomph-lib.

3.4.1 Forward and backward Euler

Forward Euler is an explicit, single-step method. It is used frequently for the solution of ion kinetics within cell models due to its simplicity and low cost. It requires only one evaluation of the derivative function and is computed by solving

$$y(t_{n+1}) = y(t_n) + \Delta t f(y(t_n), t_n), \tag{3.11}$$

but is generally unstable and has a large associated local error.

Backward Euler is an implicit, single-step method. In Oomph-lib, it is implemented as a special case of a backwards differentiation formula, one with a single step, and is evaluated by solving

$$y(t_{n+1}) = y(t_n) + \Delta t f(y(t_{n+1}), t_{n+1}). \tag{3.12}$$

3.4.2 Rush-Larsen

Special care must be taken when solving 2.3 to ensure that singularities in the formulation are suitably handled as well as ensuring that the chosen time-stepper does not overstep and result in non-physical values of n_i. Success in ensuring stability when solving this has been found through the application of the method of Rush and Larsen. The Rush-Larsen formulation for gating kinetics has been widely used for solving single-cell equations[96] [97]. The success of Rush-Larsen is due to the added stability it provides to gating variables in single-cell models. It is often applied to a single cell gating variable, y, of the form

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{y_\infty - y}{\tau_y} \tag{3.13}$$

where

$$
\begin{aligned}
y_\infty(V_m) &= \frac{\alpha_y(V_m)}{\alpha_y(V_m) + \beta_y(V_m)} \\
\tau_y(V_m) &= \frac{1}{\alpha_y(V_m) + \beta_y(V_m)}
\end{aligned}
$$

and y is the gating variable and V_m is the membrane potential. The Rush-Larsen method assumes that the membrane potential does not vary over the interval of time integration. This assumption simplifies 3.13 into a linear ODE with the exact solution[96]. This assumption simplifies the above into a linear ODE with the exact solution,

$$
y(t_{n+1}) = y_\infty + (y(t_n) - y_\infty) \exp\left(-\frac{\Delta t}{\tau_y}\right) \tag{3.14}
$$

where $y_\infty = y_\infty(V_m(t_n))$ and $\tau_y = \tau_y(V_m(t_n))$.

Application of Rush-Larsen to gating variables in conjunction with the application of forward Euler to other cell model variables results in a method which has been shown to outperform forward Euler alone[98] in terms of accuracy, stability, and efficiency.

3.4.3 Trapezoid rule: Crank-Nicolson

The trapezoid rule, also known as Crank-Nicolson, is a second-order, unconditionally stable, implicit time integration method[99]. It can be derived for a PDE of the form

$$
\frac{\partial y}{\partial t} = f(y, t), \tag{3.15}
$$

by constructing a point-wise approximation to the function $f(y,t)$ over the time interval $[t_n, t_{n+1}]$ in the form of the linear function,

$$f(y,t) \approx f(y(t_n),t_n) + \frac{(t-t_n)}{\Delta t}\left(f(y(t_{n+1}),t_{n+1}) - f(y(t_n),t_n)\right). \tag{3.16}$$

Which, upon integration over the interval, gives the approximation

$$y(t_{n+1}) - y(t_n) \approx \frac{\Delta t}{2}\left(f(y(t_{n+1}),t_{n+1}) + f(y(t_n),t_n)\right). \tag{3.17}$$

When applied to the Monodomain equation this results in the approximation to the PDE to be solved for $V(t_{n+1})$

$$V(t_{n+1}) - V_m(t_n) = \frac{\Delta t}{2}\left(\nabla\cdot(\mathbf{D}(t_{n+1})\nabla V(t_{n+1})) + \nabla\cdot(\mathbf{D}(t_n)\nabla V_m(t_n))\right). \tag{3.18}$$

The components of the conductivity tensor, D_{ij}, will in general vary spatially, but can also vary temporally as the tissue deforms (as was discussed in the sections on solid mechanics). In the heart, the time scale of solid deformation is generally longer than that of electrical activity; as a result, the equations which describe solid deformation are often fully decoupled from those for the membrane potential. Therefore, over the course of solving for the membrane potential, the conductivity, $\mathbf{D}$, is assumed to be constant. As such $\mathbf{D}(t_{n+1}) = \mathbf{D}(t_n)$ and simplifies 3.18 to

$$V(t_{n+1}) - V(t_n) = \frac{\Delta t}{2}\left(\nabla\cdot\left(\mathbf{D}(t_{n+1})\nabla\left(V(t_{n+1}) + V(t_n)\right)\right)\right), \tag{3.19}$$

which is the form of the Monodomain equation implemented in monodomain elements. If it is desired that the electrophysiology equations are solved monolithically with the solid deformation, then a new implementation of the multiphysics element would be required in which the previous node positions are used to calculate the deformation of the conductivity tensor at the last time-step and the current node positions are used to calculate the deformation of the conductivity tensor at the current time-step.

3.4.4 Implicit linear multi-step methods

Implicit linear multi-step methods are commonly used in Oomph-lib for most time-stepping applications. They utilise the solution at previous time steps to approximate time derivatives. They are widely used for the solution of stiff differential equations and have advantages over other high-order, implicit, time integration methods, such as Runge-Kutta.

Backward differentiation formula

Backward Differentiation Formula (BDF) are derived from applying Lagrange interpolation polynomials to the set of points $(t_n, y_n), (t_{n-1}, y_{n-1}), \ldots (t_{n-s}, y_{n-s})$. The derivative of this polynomial then represents an approximation of the derivative of the function $y(t)$ which can be written in the form

$$y'(t_n) = \sum_{i=0}^{i=s} a_i y_{n-i},$$

where the coefficients a_i depend on the time-steps $\Delta t_k = t_{n-k} - t_{n-k-1}$.

The order of the method is determined by the value of s[100]. In Oomph-lib 3 BDF time-steppers are implemented corresponding to $s = 1, 2, 4$. Error estimates can be calculated from the previous time derivatives of the solution. Backward differentiation formulae are often applied to the solution of stiff PDEs [100]

3.4.5 Adaptive time integration

For each Degree of Freedom (DOF) a measure of the error in the solution can be computed. The time-step is adjusted according to some heuristic, that which is used by Oomph-lib is

$$\Delta t_{\text{new}} = 0.9 \Delta t_{\text{old}} \left(\frac{T_E}{\epsilon} \right)^{\frac{1}{2}} .$$

If the error is greater than the specified tolerance, then the time-step length is halved, the solution is rejected, and the solution at the current time-step is re-computed. Otherwise the new suggested time-step length is used for the next time step.

3.5 Monolithic and partitioned solutions

It is often computationally advantageous to solve the individual sub-physics of a problem separately instead of at the same time. For example, problems containing highly non-linear terms alongside linear ones could be solved by solving only either the non-linear or linear terms at any one time.

Partitioned solutions are often used for solving the reaction-diffusion type system of cardiac electrophysiology. There are several reasons for this, firstly the number of DOFs in most physiological cell models is very large. The number of equations and the size of the matrix in the resultant linear system makes a monolithic solution far too expensive. Secondly, the stiffness and non-linearity of the cell equations can result in many Newtonian iterations before convergence is achieved. Additionally due to the common existence of poles within the cell equations very small time steps are often required in order to resolve the solution close to these singularities. If the solution of the system is not partitioned, then stiffness in the cell equations requires that the entire system is solved many times. Furthermore, since the cell equations have no explicit spatial dependence, each cell can be solved independently of the others which makes inverting the very large, dense, matrix due to the monolithic solution redundant in the absence of diffusion. Since the cell equations tend to have much smaller time scales than the diffusion equations, solving the two together results in many more time steps over which the linear system must be assembled and solved potentially many times.

Partitioned solution methods are at odds with the *de facto* method within Oomph-lib of solving all sub-physics of a problem at the same time in a monolithic fashion using high-order implicit time-steppers. Their use, therefore, necessitated additions to the library and changes to the time-stepping procedure. However, we will see that changes to driver codes are minimal and, with future work, could be reduced further.

3.5.1 Monolithic solutions

Monolithic solutions are widely considered to be the most stable way to solve coupled systems of equations, however, it is commonly assumed that the monolithic solution of large systems is more computationally expensive than alternative partitioned methods. Heil and Hazel demonstrated that for Fluid-Solid Interaction (FSI) problems, with proper use of preconditioners, a monolithic solution is competitive when compared with a partitioned solution using Picard iteration.

It has been argued[101] that monolithic solvers with suitable preconditioners can outperform partitioned solvers. Commonly, FSI problems are solved using a partitioned method, but it is understood that a monolithic solution with a suitable pre-conditioner for the resultant block matrix can converge faster in certain circumstances.

This example differs from simulations of cardiac biomechanics since the cell models in the multi-physics problems contain many variables and often contain no explicit spatial dependence in their formulation. Additionally, cell variables often change on a much shorter time scale to the membrane potential and so solving the two together will result in having to construct and invert a very large Jacobian many times. Although block pre-conditioners will likely assist in reducing the time taken to perform this computation, such pre-conditioners will have to be determined for each cell model and could be exceedingly complex. These are therefore not investigated here, nor are monolithic implementations attempted. Instead, partitioned solvers were considered.

3.5.2 Weak coupling

Weak coupling is the term often used to describe when the solutions of two systems with different intrinsic time scales are coupled. In such situations, the solution of one system often requires significantly smaller time steps than the other. It may be more efficient to separate the solution of the subsystems so that the system with the longer intrinsic time-scale is not solved to unnecessarily high accuracy.

In the context of cardiac biomechanics, two examples where weak coupling may be effectively applied are in the separation of the sub-cellular dynamics from the electrical diffusion, and the separation of the cellular and electrical dynamics from the FSI.

During an operator-splitting solve, the cell model is solved for the time-step prescribed by the operator-splitting method. Often, cell models are known to have a maximum time step for which the model is stable. If the operator splitting method requires a time step which exceeds this value, as can be the case for adaptive operator splitting methods, then the cell model must take multiple smaller time steps in order to fulfil the time step requested by the operator splitting method. In this way, although operator splitting techniques are generally strongly coupled solvers, since there may be multiple solutions of the cell model for each solution of the diffusion equations, the diffusion equations and cell model equations can be described as being dynamically weakly coupled when adaptive operator splitting methods are implemented.

Solid and fluid models can be solved alongside the electrophysiological model by applying weak coupling. Here, the electrophysiological system is solved using operator splitting, then the solid/fluid system is solved over the whole time step with the active strain prescribed by the cell model. This method is cheaper than strongly coupling the solid/fluid system to the electrophysiological system since the linear and non-linear systems can be separated, and thus the systems solved at each stage of the method have fewer DOFs. Additionally, the solid/fluid system only requires to be solved once for each time step of the simulation instead of the several times which would be required if the solid/fluid system were solved monolithically with the electrophysiological system or if an ABC operator-splitting method.

3.5.3 Operator splitting

Partitioned solutions can sometimes be used to effectively reduce the cost of numerical computation for certain physical systems. This is achieved by solving for the individual sub-physics separately. Many techniques exist for calculating the partitioned solution, several of which will be introduced and discussed.

Cell models usually consist of many state variables, ranging from a minimal 3 in caricature cell models [102] to many dozen in biophysically detailed cell models [103]. These state variables are explicitly dependent on the membrane potential (V_m) and are hence implicitly dependent on spatial derivatives of V_m when solving for their values at some future time. The full electro-physiological tissue model consists of the point-wise cell model equations (e.g., a set of ODEs for ion channel kinetics and membrane potential) and the spatially dependent diffusion of membrane potential. Finding a solution to this combined system involves solving both of these sub-problems simultaneously. However, solving for the cell state variables within a finite element method turns out to be prohibitively expensive, since this necessitates assembling and inverting a very large and dense matrix at each iteration of the Newtonian solver.

Instead, operator splitting techniques [18] are employed such that the cell equations are solved point-wise in time and a relatively small, linear matrix which arises from the diffusion problem is inverted at each iteration of the Newtonian solver.

Perhaps the most intuitive operator splitting technique is that of strong coupling. Strong coupling is the name often given to the Lie-Trotter operator splitting method which corresponds to solving one differential operator over the entire time-step and then solving the other using the solution given by the first operator as the initial conditions. Lie-Trotter splitting is the most simple and the least accurate of operator splitting methods.

For systems with two distinct sets of DOFs, such as FSI where DOFs can be separated into fluid and solid, Picard fixed point iterations can be applied. In these cases, the solution is given by two or more sets of equations with coupling terms. Each set of equations corresponds to its own set of DOFs and therefore solving one of these sets of equations corresponds to updating its corresponding DOFs. Each set of equations is solved alternately, with the DOFs corresponding to the other equation held as fixed. This is a common method of tackling large FSI simulations where the monolithic problem is often considered to be too large to solve efficiently.

However, the use of partitioned solutions is not always suitable or guaranteed to improve the cost of simulation. For example, it has been demonstrated by Heil and Hazel that the common assumption that partitioned solutions of FSI systems converge faster than monolithic solutions may not always be true. They demonstrated that provided a suitable pre-conditioner for the resultant linear system, a monolithic solution can indeed converge significantly faster than a partitioned solution using Picard fixed-point iterations and Iron and Tucks convergence acceleration. Additionally, they showed that the monolithic solution was considerably more stable than the Picard iteration counterpart which often failed to converge or diverged quickly.

Furthermore, since this method only works for implicit equations it may not be suitable for applications including cell models since these are often solved explicitly. Indeed methods for solving the bidomain and monodomain equations exist which make use of a single Picard iteration[104][105]. However, these methods necessitate the implicit solution of cell models. Since cell models are often highly non-linear, these implicit solutions can take many Newton iterations to converge and hence negatively impact computation time.

The cell equations and diffusion equations are often solved using the operator splitting method known as Strang splitting. Here, the diffusion equation is often solved using an implicit, single-step, time-stepper such as Crank-Nicolson, and the cell equations are often solved using explicit, single-step, time-steppers, such as forward Euler or Rush-Larsen, depending on the specific cell equation being solved. This paradigm has generally been found to be the most computationally efficient and widely used for most choices of cell and diffusion models.

To illustrate this method of solving the cell and diffusion equations, the steps taken to calculate the solution using Strang-splitting as applied to a cell model, and the Monodomain model are described explicitly.

1. Initially the solution is given at time $t = t_n$ to be V_n and w_n for the membrane potential and the cell variables respectively.

2. The solution is advanced in time by $\Delta t/2$ according to the cell equations to give the first intermediate solution $V_1^{1/2}$ and $w_1^{1/2}$.

3. The membrane potential is then advanced in time by Δt according to the Crank-Nicolson formulation of the Monodomain equation starting at $V_1^{1/2}$

$$V_2^{1/2} - V_1^{1/2} = \frac{\Delta t}{2}\left(\nabla \cdot \left(D_n \nabla \left(V_2^{1/2} + V_1^{1/2}\right)\right)\right) \tag{3.20}$$

   which gives the second intermediate solution $V_2^{1/2}$, and $w_2^{1/2} = w_1^{1/2}$.

4. Finally, the solution is advanced in time by $\Delta t/2$ according to the cell equations, starting at $V_2^{1/2}$ and $w_2^{1/2}$ to give V_{n+1} and w_{n+1}

Operator splitting introduces new uncertainty into the solution in the form of splitting error, the size and nature of which are dependent on the operator splitting scheme chosen and the particulars of the problem being solved [18].

A general class of operator splitting schemes is based on sequentially solving the two sub-operators over fractional time steps. These techniques are derived from the Campbell-Baker-Hausdorff theorem as applied to matrices which do not commute under the exponential mapping [18] [106]. Such techniques have been shown to be capable of achieving high degrees of accuracy and may significantly reduce the cost of solving some multi-physics and multi-scale problems [107]. The operator splitting scheme of this kind most commonly applied to cardiac electro-physiological systems is Strang splitting [108] [109]. Also known as Störmer-Verlet or leapfrog, Strang splitting is second-order accurate and is applied extensively throughout the field to great success [108] [110] [111]. However, despite its widespread use, Strang splitting does not provide a dynamic measure of error and as a result, cannot be used to adaptively change the time-step length. Depending on the stiffness of the differential equations being solved, this may result in an under-resolved or over-resolved solution, giving either incorrect or inefficient results.

For a general differential equation which can be split into two differential operators, it may be represented as:

$$\frac{\partial y}{\partial t} = A(y,t) + B(y,t). \tag{3.21}$$

To solve the equation, we can define the mapping φ_τ as in [106] which corresponds to advancing y in time by τ according to 3.21, such that

$$y(t_n + \tau) = \varphi_\tau y(t_n). \tag{3.22}$$

We can further define the mappings $\Psi_\tau^{[A]}$ and $\Psi_\tau^{[B]}$ which correspond to advancing the system in time by τ according to the PDEs

$$\frac{\partial \hat{y}}{\partial t} = A(\hat{y},t) \quad \text{and} \quad \frac{\partial \tilde{y}}{\partial t} = B(\tilde{y},t)$$

respectively, such that

$$\hat{y}(t_n + \tau) = \Psi_{\tau}^{[A]} \hat{y}(t_n) \tag{3.23}$$

$$\tilde{y}(t_n + \tau) = \Psi_{\tau}^{[B]} \tilde{y}(t_n) \tag{3.24}$$

In the context of the Monodomain equation, this corresponds to

$$\frac{\partial \hat{V}_m}{\partial t} = \nabla \cdot D \nabla \hat{V}_m = A(\hat{V}_m, t) \tag{3.25}$$

$$\frac{\partial}{\partial t} \begin{pmatrix} \tilde{V}_m \\ \underline{\tilde{\omega}} \end{pmatrix} = \begin{pmatrix} -I_{\text{ion}}(\underline{\tilde{\omega}}, \tilde{V}_m)/C_m \\ \underline{f}(\underline{\tilde{\omega}}, V_m, t) \end{pmatrix} = B(\underline{\tilde{\omega}}, \tilde{V}_m, t) \tag{3.26}$$

Strang-splitting can be written using this notation:

$$\Psi_{\Delta t}^{\text{Strang}} = \Psi_{\frac{\Delta t}{2}}^{[A]} \Psi_{\Delta t}^{[B]} \Psi_{\frac{\Delta t}{2}}^{[A]}. \tag{3.27}$$

Strang-splitting is a specific case of a larger family of operator splitting schemes which are evaluated by alternately performing the two mappings [18]. In general, a member of this family can be written using the ordered product:

$$\Psi_{\Delta t} = \prod_{i=1}^{i=m} \Psi_{\beta_i \Delta t}^{[B]} \Psi_{\alpha_i \Delta t}^{[A]} \approx \varphi_{\Delta t}, \tag{3.28}$$

where $\prod$ successively applies terms to the left-hand side, m is any finite positive integer, and $\alpha_i, \beta_i \in \mathbb{C}$.

For a proper choice of m, α_i, and β_i a method of order p such that $\Psi_{\Delta t} = \varphi_{\Delta t} + \mathcal{O}\left(\Delta t^{p+1}\right)$ can be found [106]. It has been shown [106] that necessary and sufficient conditions on the values of m, α_i, and β_i exist for a method to be of order 1, 2, or 3 and are as follows:

1. order 1:
$$\sum_{i=1}^{m} \alpha_i = 1, \sum_{i=1}^{m} \beta_i = 1$$

2. order 2:
$$\sum_{i=1}^{m} \beta_i \left(\sum_{j=1}^{i} \alpha_j \right) = \frac{1}{2}$$

3. order 3:
$$\sum_{i=1}^{m-1} \beta_i \left(\sum_{j=i+1}^{m} \alpha_j \right)^2 = \frac{1}{3}, \quad \sum_{i=1}^{m} \alpha_i \left(\sum_{j=1}^{m} \beta_j \right)^2 = \frac{1}{3}$$

where methods of order p satisfy all conditions on orders $\leq p$. It has further been shown [106] that the conditions for order 3 necessitate the existence of at least one negative coefficient in both α_i and β_i and therefore any method of order ≥ 3 must contain backwards time integration of the differential operators A and B. Ideally, a high-order splitting method will be used in order to ensure that the splitting error is of the same order as the errors in both the diffusion and cell solvers. However, this presents an issue since the equations in the Monodomain and Bidomain models are parabolic PDEs and widely understood to be ill-posed when solved backwards in time. It was found that attempts to use methods with order greater than two resulted in unreliable solutions which often diverged unpredictably from run to run, indicating that numerical round-off error determined whether or not the solution converged and thus that the resultant system is ill-posed. For this reason, the focus was restricted to finding efficient second-order, adaptive operator splitting schemes. Algorithms for generating methods of arbitrary order have been developed [18], and a large repository exists [18] which contains several methods of orders 1 and 2 as well as higher orders.

Predictor-corrector methods have long been used in numerical integration to achieve a solution with the desired accuracy [112] [113]. In these, two different methods, $\Psi^{(1)}_{\Delta t}$ and $\Psi^{(2)}_{\Delta t}$, are applied and their solutions are compared in order to give an approximation of the solution error [18]. There exist several broad types of predictor-corrector methods in the context of operator splitting [18], in this study only methods which are either Milne pairs or palindromic types were considered.

Milne pairs are constructed by applying the Milne device [114] to an operator splitting scheme. In the repository used [18], Milne pairs were generated by seeking methods such that the local errors are related according to

$$\Psi^{(1)}_{\Delta t} y - \varphi_{\Delta t} y = C(\Delta t, y)\Delta t^{p+1} + \mathcal{O}\big(\Delta t^{p+2}\big) \tag{3.29}$$

$$\Psi^{(2)}_{\Delta t} y - \varphi_{\Delta t} y = \gamma C(\Delta t, y)\Delta t^{p+1} + \mathcal{O}\big(\Delta t^{p+2}\big) \tag{3.30}$$

for some $\gamma \neq 1$.

The solution is then given by the additive scheme

$$\varphi_{\Delta t} y \approx \frac{1}{1-\gamma}\left(-\gamma \Psi^{(1)}_{\Delta t} y + \Psi^{(2)}_{\Delta t} y\right), \tag{3.31}$$

with the local error estimate

$$\Psi_{\Delta t} y - \varphi_{\Delta t} y \approx \frac{1}{1-\gamma}\left(\Psi^{(1)}_{\Delta t} y - \Psi^{(2)}_{\Delta t} y\right). \tag{3.32}$$

Palindromic methods are achieved through swapping the order of the two mappings as applied in $\Psi^{(1)}_{\Delta t}$ when applying $\Psi^{(2)}_{\Delta t}$, and so if

$$\Psi^{(1)}_{\Delta t} = \prod_{i=1}^{i=m} \Psi^{[B]}_{\beta_i \Delta t} \Psi^{[A]}_{\alpha_i \Delta t} \tag{3.33}$$

$$\implies \Psi^{(2)}_{\Delta t} = \prod_{i=1}^{i=m} \Psi^{[A]}_{\beta_i \Delta t} \Psi^{[B]}_{\alpha_i \Delta t}. \tag{3.34}$$

The local error estimate of $\Psi^{(1)}_{\Delta t} y - \varphi_{\Delta t} y$ provided by a palindromic method is dependent on $\Psi^{(1)}_{\Delta t} y$ and $\Psi^{(2)}_{\Delta t} y$.

Another type of predictor-corrector for operator splitting is embedded methods (similar in concept to Runge-Kutta pairs [115]) in which the predictor and corrector share several initial steps and so $\Psi^{(1)}_{\Delta t}$ can be partially (or fully) embedded within $\Psi^{(2)}_{\Delta t}$ [116]. These methods may result in reduced computational complexity, however, relatively few embedded operators splitting schemes exist in the collection [18] and there were no second-order methods which out-performed either the Palindromic or Milne methods tested.

Auzinger et al[18] provide a collection of many operators splitting schemes of which all first and second-order predictor-corrector methods which contain non-negative coefficients were selected for use in this study. This results in three methods: Palindromic Lie-Trotter, Strang-Milne, and Symmetric-Milne-32.

The Strang-Milne [18] method requires 5 applications of $\Psi^{[A]}_{\tau}$ and 3 applications of $\Psi^{[B]}_{\tau}$ and is given by equations 3.27 and 3.35, where 3.35 provides an error measure when compared to 3.27

$$\Psi^{\text{Strang-Milne Device}}_{\Delta t} = \Psi^{[A]}_{\frac{\Delta t}{4}} \Psi^{[B]}_{\frac{\Delta t}{2}} \Psi^{[A]}_{\frac{\Delta t}{2}} \Psi^{[B]}_{\frac{\Delta t}{2}} \Psi^{[A]}_{\frac{\Delta t}{4}}. \tag{3.35}$$

An alternative to Strang-Milne is Symmetric-Milne-32 which is given by equations 3.36 and 3.37, where 3.36 provides an error measure when compared to 3.36. Although Symmetric-Milne-32 requires one additional application of $\Psi_{\tau}^{[B]}$ when compared to Symmetric-Milne it may provide a better estimate of the solution and local error.

$$\Psi_{\Delta t}^{\text{Symmetric-Milne}} = \Psi_{\frac{433494437}{2269806340}\Delta t}^{[A]} \Psi_{\frac{\Delta t}{2}}^{[B]} \Psi_{\frac{701408733}{1134903170}\Delta t}^{[A]} \Psi_{\frac{\Delta t}{2}}^{[B]} \Psi_{\frac{433494437}{2269806340}\Delta t}^{[A]} \tag{3.36}$$

$$\Psi_{\Delta t}^{\text{Symmetric-Milne Device}} = \Psi_{(1-\sqrt{2}/2)\Delta t}^{[B]} \Psi_{\frac{\sqrt{2}\Delta t}{2}}^{[A]} \Psi_{\frac{\sqrt{2}\Delta t}{2}}^{[B]} \Psi_{(1-\sqrt{2}/2)\Delta t}^{[A]} \tag{3.37}$$

The final method considered is the palindromic Lie-Trotter method described in equations 3.38 and 3.39. Although each of $\Psi_{\Delta t}^{\text{Palindromic-Lie-Trotter-Forward}} y$ and $\Psi_{\Delta t}^{\text{Palindromic-Lie-Trotter-Backward}} y$ are of first order, their combination $\frac{1}{2}\left(\Psi_{\Delta t}^{(1)} y + \Psi_{\Delta t}^{(2)} y\right)$ is a second order approximation to $\varphi_{\Delta t}$, and their difference $\frac{1}{2}\left(\Psi_{\Delta t}^{(1)} y - \Psi_{\Delta t}^{(2)} y\right)$ provides a local error estimate of $\Psi_{\Delta t}^{(1)} y$.

$$\Psi_{\Delta t}^{\text{Palindromic-Lie-Trotter-Forward}} = \Psi_{\Delta t}^{[B]} \Psi_{\Delta t}^{[A]} \tag{3.38}$$

$$\Psi_{\Delta t}^{\text{Palindromic-Lie-Trotter-Backward}} = \Psi_{\Delta t}^{[A]} \Psi_{\Delta t}^{[B]} \tag{3.39}$$

For all methods applied, for a given variable x_i, the error estimate of that variable, ϵ_i, is normalised to its magnitude according to

$$\epsilon_i = \frac{\epsilon_i}{1 + \max\left(|\hat{x}_i|, |\tilde{x}_i|\right)} \tag{3.40}$$

where $\hat{x}_i$ and $\tilde{x}_i$ are the approximations to x_i given by the two methods. The total error over all N variables is then given by

$$\epsilon = \sqrt{\frac{1}{N}\sum_{i}^{N} \epsilon_i^2}. \tag{3.41}$$

The time step taken by the operator splitting scheme is modified in response to the error estimate. Applying the widely used method [117] given by

$$\Delta t_{\text{new}} = 0.9 \Delta t_{\text{old}} \left(\frac{T_E}{\epsilon} \right)^{\frac{1}{2}}$$

where $T_E > 0$ is the error tolerance, ϵ is the error approximation, and Δt_{old} is the previously taken time-step, results in error estimates very close to and slightly under the specified tolerance and therefore only rarely resulted in re-computation of the solution.

Since source functions for applied electrical stimulus are often discontinuous, frequently taking the form of Heaviside functions, uncontrolled adaptive operator splitting methods can often overstep the application of external stimuli. Therefore, in order to ensure that important features of a simulation, such as stimuli, are not excluded, the maximum size of Δt for each time step must be controlled.

As an example, a simulation in which the substrate is subjected to pacing stimuli with period Δ_{BCL}, stimulus duration Δ_{duration}, and total simulation length $t_{\max}$ is defined. The time since the last stimulus is Δ_{last}, and the current time is t. A diagram describing this situation is given in figure 3.1

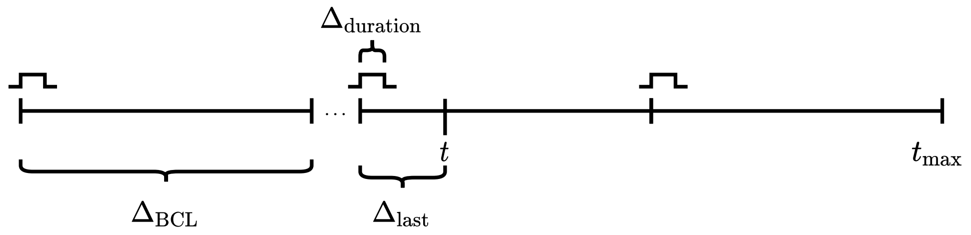


Figure 3.1: Schematic of time step control for simulation with paced applied electrical stimulus. Pacing stimuli with period Δ_{BCL} are applied with durations Δ_{duration}. The time since the last applied stimulus is Δ_{last}. A maximum time step can then be calculated to ensure that the entire duration of each stimulus is applied and that the next applied stimulus is not excluded.

The time step can be controlled such that: neither the stimuli are ignored nor the desired end of the simulation is overstepped by the following procedure

1. $\Delta t = \Delta t_{\text{new}}$
2. if $\Delta_{\text{duration}} > \Delta_{\text{last}}$ then $\Delta t = \min\left(\Delta_{\text{duration}} - \Delta_{\text{last}}, \Delta t\right)$
3. $\Delta t = \min\left(\Delta_{\text{BCL}} - \Delta_{\text{last}}, \Delta t\right)$
4. $\Delta t = \min\left(t_{\max} - t, \delta_t\right)$

An upper limit on Δt was imposed by setting $\Delta t = \min\left(\Delta t_{\text{new}}, T_{\text{Next}} - T\right)$ in order to ensure that external stimuli applied to the simulated tissue are not overstepped by the solver. Here T_{Next} is the time of the next applied stimulus or the time at which the simulation is to end, whichever comes first, and T is the current simulation time.

Due to the nature of adaptive operator splitting schemes, arbitrarily coarse time steps could be imposed on the solver of the cell model. For such coarse time steps, the commonly used methods of forward Euler and Rush-Larsen[96] are unlikely to give a sufficiently accurate or stable solution. The maximum value of the *super time steps* taken by the operator splitting method could be controlled by imposing a hard upper limit known to be under the maximum value for which FERL converges, however, this will potentially result in a less efficient overall method since then the diffusion equation will also have to be solved over this smaller time-step.

Instead, in order to account for this, cell model methods are solved several times over each time interval. This ensures that the time step used for each solution of the cell equations is no larger than the maximum time step that is known to produce accurate and stable results. This time step is iteratively applied until the equations have been solved over the desired interval. If a maximum time-step of $\Delta t_{\max} < \Delta t$ is imposed while solving the cell equations over an interval of length Δt. Then $N + 1$ steps can be taken such that $\Delta t = N\Delta t_{\max} + \epsilon$, where $N \geq 0$ is any positive whole number and $0 < \epsilon < \Delta t_{\max}$. The cell equations are solved N times with the time-step Δt and once with the time-step ϵ. It was found, during validation, that this method was sufficiently robust and accurate for solving the cell equations during the required stages of the adaptive operator splitting schemes.

Solid and fluid systems are non-linear and have much DOFs. In comparison, the diffusion system is linear and has comparatively few DOFs. Solving for the solid and fluid dynamics at every iteration of the electrophysiology solve is therefore much more expensive than solving for electrophysiology alone.

Operator splitting methods alternately solve for the individual differential operators over fractional time steps from the solution of which the next differential operator is solved. Solving with inertial effects (i.e unsteady, solid and fluid mechanics) requires BDF/Newmark time-steppers, which use previous solution values to approximate time derivatives. Since the fractional time-steps found in operator splitting methods are not physical solutions, it may not make sense to apply generally, high order, BDF/Newmark time-steppers to them.

Single-step time-steppers such as backward Euler, which is synonymous with the single-step BDF time stepper, are used in operator splitting to solve the cell-diffusion coupled system. This could also be applied to the FSI coupled system, however, due to the nonlinearity and stiffness of hyperelastic solid equations and fluid models, this may result in a high error and therefore require very small time steps to guarantee convergence. The solution of solid and fluid mechanics with inertial effects may therefore not be feasible over the course of an operator splitting solve.

Commonly, inertia-free assumptions are made for cardiac tissue[97]. If the inertia-free assumption is made, then the solid and fluid mechanics can be solved during the operator splitting method in a third stage using an ABC operator splitting method. However, changes in the pressure of blood entering the heart, and deformation of the fluid domain during muscular contraction and relaxation result in a physical system which is unlikely to be suitably approximated by a steady fluid flow.

It is observed that the time scale of sarcomere shortening and therefore that of solid deformation and changes to blood flow are longer than that of cellular electrical dynamics. This motivates the use of weak coupling.

3.5.4 Operator splitting applied to the bidomain and monodomain equations

The bidomain equations solved by the newly implemented Oomph-lib elements are given by 2.52 and 2.53 along with 2.15 which is solved for the cell variables and where I_{ion} is given by 2.16.

When operator splitting is employed to solve this system, the non-linear source terms are separated from the linear diffusion terms. This results in the equations

$$JC_m\left(\frac{\partial\tilde{v}}{\partial t}-\frac{\partial\tilde{v}}{\partial x_I}\frac{\partial X_I}{\partial t}\right)=\frac{\partial}{\partial x_I}\left(\tilde{D}_{IJ}\frac{\partial(\tilde{v}+\tilde{\phi})}{\partial x_J}\right), \tag{3.42}$$

$$\frac{\partial}{\partial x_I}\left(\tilde{D}_{IJ}\frac{\partial\tilde{v}}{\partial x_J}\right)+\frac{\partial}{\partial x_I}\left((\tilde{D}_{IJ}+\tilde{E}_{IJ})\frac{\partial\tilde{\phi}}{\partial x_J}\right)=0, \tag{3.43}$$

$$C_m\frac{\partial\tilde{v}}{\partial t}+I_{\text{ion}}(\underline{\omega},\tilde{v},t)-I_{\text{source}}(\underline{x},t)=0, \tag{3.44}$$

$$\frac{\partial\omega}{\partial t}=\underline{f}(\underline{\omega},\tilde{v},t). \tag{3.45}$$

With the same boundary conditions defined in 2.54 and 2.55.

The equations 3.42 and 3.43 now represent one of the splitting operators, and equations 3.44 and 3.45 represent the other. When Strang-splitting is applied, the non-linear source terms are the first operator and the diffusion terms are generally the second operator.

In order to derive the system of equations for the monodomain model with operator splitting the same simplifying assumption 2.56 is made. This results in

$$JC_m\left(\frac{\partial\tilde{v}}{\partial t}-\frac{\partial\tilde{v}}{\partial x_I}\frac{\partial X_I}{\partial t}\right)=\frac{\partial}{\partial x_I}\left(\tilde{D}_{IJ}\frac{\partial(\tilde{v})}{\partial x_J}\right), \tag{3.46}$$

$$C_m\frac{\partial\tilde{v}}{\partial t}+I_{\text{ion}}(\underline{\omega},\tilde{v},t)-I_{\text{source}}(\underline{x},t)=0, \tag{3.47}$$

$$\frac{\partial\omega}{\partial t}=\underline{f}(\underline{\omega},\tilde{v},t), \tag{3.48}$$

with the same boundary condition 2.58 and where a relabelling of the intracellular conductivity tensor has incorporated the $\frac{1}{1+\lambda}$ term into the conductivity tensor components $\tilde{D}_I J$

3.5.5 Mechano-electrical feedback

The conductivity tensor changes in response to tissue deformation. Strong coupling allows for the conductivity tensor to be updated in response to solid deformation, whereas for weak coupling the solid deformation is fixed for the duration of the electrophysiological solution.

Strong coupling, for the stress decomposition formulation, also allows for SAC to be mediated by a continuously changing measure of strain, whereas for weak coupling the strain either has to be taken only from the cell model or prescribed by the solid deformation and hence held constant. However, since the elastic strain formulation varies over the course of solving the electrophysiology system, the active strain decomposition method allows for the strain to be updated even when the deformation of the body is held constant. The active strain decomposition may therefore mitigate the drawbacks of only weakly coupling the electro-physiology solve to the FSI physics.

3.6 Solving linear systems

3.6.1 Newton's method

Oomph-lib assumes all problems to be non-linear and that the solution is dependent on M discrete values. Some variables may be prescribed, e.g. boundary conditions, and therefore only some subset N of these values are unknowns. It is assumed that these values are determined by solving a system of N non-linear equations which may be written in a residual form as

$$\mathcal{R}_i(U_j) = 0, \text{ for } j = 1, \ldots, N.$$

This system of equations is then solved using Newton's method.

Consider a choice of parameters $\bar{U}_j$ to be some initial guess for the true solution U_j. Assuming that the initial guess is suitably close to the true values we can write

$$\bar{U}_j + \delta U_j = U_j.$$

Substituting this form of U_j into the residuals provides

$$\mathcal{R}_i(\bar{U}_j + \delta U_j) = 0$$

and the task is now to find the values δU_j. We perform Taylors expansion on the residuals which yields

$$\mathcal{R}_i(\bar{U}_j) + \left.\frac{\partial \mathcal{R}_i}{\partial U_k}\right|_{\bar{U}_j} \delta U_k \approx 0$$

which when rearranged provides

$$\mathcal{J}_{ik}\big|_{\bar{U}_j} \delta U_k = -\mathcal{R}_i(\bar{U}_j). \tag{3.49}$$

The default procedure to solve equations 3.49 is through iteration. An initial guess is provided, the variance δU_K is calculated and the guess is updated and fed back into the equations. The full procedure is outlined as

1. Initialise the iteration counter, $n = 0$.
2. An initial approximation of the unknowns is given by $U_j^{(n)}$.
3. The residuals are evaluated as $\mathcal{R}_i^{(n)} = \mathcal{R}_i(U_j^{(n)})$.
4. A suitable norm of the residual vector, $\mathcal{R}_i^{(n)}$, is evaluated. If the value of this norm is less than some predefined tolerance then stop and accept $U_j^{(n)}$ as the solution.

5. The Jacobian matrix is evaluated with $\mathcal{J}_{ik}^{(n)} = \frac{\partial \mathcal{R}_i}{\partial U_k}\Big|_{U_k^{(n)}}$.

6. The linear system $\mathcal{J}_{ik}^{(n)} \delta U_k = -\mathcal{R}_i^{(n)}$ is solved.

7. The guess is updated with $U_j^{(n+1)} = U_j^{(n)} + \delta U_j$.

8. Increment $n = n + 1$ and go to step 3.

3.6.2 Direct and iterative solvers

Solving the linear system,

$$\mathcal{J}_{ij} \delta U_j = -\mathcal{R}_j \tag{3.50}$$

can vary in numerical complexity depending on the form of the matrix $\mathcal{J}_{ij}$. Through personal experience, systems arising in cardiac simulations are adequately and most efficiently solved using direct linear solvers, however, for comparison, a brief overview of iterative solvers will be given.

Issues with convergence emerged when SuperLU was used in solving the monodomain equation. These issues were remedied when the Mumps direct parallel solver was used. Mumps is a parallel sparse direct solver and its use within Oomph-lib is well documented[118].

Chapter 4

Developing the numerical package

Oomph-lib is a proven numerical library for the simulation of multi-physics problems. It has been applied to research in a number of different fields and presents a possible alternative approach to numerical simulation of cardiac multi-physics.

Cardiac simulations involve the solution of many, often stiff, systems of equations. The robust method in which the Oomph-lib monolithic solver handles stiff numerical systems, such as those which arise in FSI with large-scale deformations[16], potentially makes it particularly well suited for whole organ cardiac or whole cardiovascular system simulations.

FEM, although generally more complicated to implement than FDM and DEM, has several advantages such as robust spatial error measures, well-documented mesh refinement procedures, and taking advantage of the weak formulation to reduce smoothness requirements on the solution. However, cardiac biomechanics can be, and often are, simulated with FVM[119], FDM[95], DEM[120], or FEM[109]. Oomph-lib is implemented to use FEM, although it is conceivable that the finite elements within Oomph-lib could be repurposed to use other numerical methods such as FDM. Indeed, since

Oomph-lib is open source, any changes or additions to Oomph-lib necessary for any purpose can be made to the library.

One of the key advantages of Oomph-lib over other numerical libraries, other than its open-source licensing, is the flexibility in which boundary conditions and source terms can be implemented as well as how multiphysics interactions between separate elements are set up.

Implementing the equations required for simulating cardiac biomechanics within Oomph-lib will add several key features to the library. Namely, the addition of operator splitting methods for highly non-linear point source terms. Oomph-lib is a monolithic solver, which is well-suited for most physical problems. However, the simulation of biophysically detailed cardiac biomechanics requires the computation of cell models with many variables, often of the order 10^2. Although the Oomph-lib's monolithic solver could indeed simulate these point source terms alongside the electrical diffusion equations that couple them, this approach is often prohibitively slow since the resultant Jacobian in the Newton solver would be very large. For this reason, operator splitting schemes are often implemented to separate the non-linear point source terms from the diffusion terms. In developing cardiac electrophysiology elements, additions were made to the library which allows for the separation of point source terms from the rest of the Oomph-lib finite elements and meshes which may be applicable to other, structurally similar, physical problems. Anisotropic solid mechanics was also added. The equations which govern non-linear solid deformation within Oomph-lib are implemented in an isotropic fashion. Currently, changes can be made to the constitutive laws which determine stress-strain relationships to include anisotropic effects such as fibre reinforcement, however, if anisotropic effects vary spatially in a way which is difficult to capture analytically, as is found in the heart, then a reimplementation of these elements is required. Anisotropic versions of the isotropic non-linear solid elements were developed and implemented in such a way that anisotropic terms can be derived from external finite elements, including the non-linear point source terms. This approach allows for tissue models based on detailed Magnetic Resonance (MR)/Computerised Tomography (CT) imaging to be simulated. A pipeline has been developed for the generation of Oomph-lib meshes from such MR/CT images which provides an alternative approach to the current method by which biological solid meshes are generated within Oomph-lib.

By developing the equations for cardiac biomechanics within Oomph-lib, the choice of numerical methods, such as time-stepping and spatial interpolation and integration, can be restricted to those which are already implemented within Oomph-lib. Although the library does not prohibit the development of alternative methods should they be more suitable for a specific purpose.

Finally, Oomph-lib has been predominantly developed at the University of Manchester which has allowed for insightful and expert guidance during the development of several of these additional features.

4.1 Oomph-lib

Oomph-lib is an open-source finite-element library developed and maintained within the Department of Mathematics at The University of Manchester. In order to write a simulation with Oomph-lib, a user will write their own C++ driver code using high-level objects contained within the library. Extensive documentation is provided for existing functionality contained within the library along with example codes and tutorials.

Oomph-lib provides a large number of existing objects including elements and time-steppers which can be combined to solve many problems. For any problems which cannot be solved with the provided objects, new functionality can be added.

Self-tests are provided for use during code development, which can be enabled by setting the compiler flags *DPARANOID* and *DRANGE_ CHECKING*. If an issue is encountered when these flags are set then the code will terminate gracefully and provide the information required to perform diagnostic checks[121]. Considerable computational overhead is incurred through the use of such checks, so they can be turned off in production runs.

The main aim in the design and implementation of the Oomph-lib library is to provide an environment which facilitates robust, monolithic solutions to multi-physics problems while maximising the potential for code reuse. Application of monolithic solvers allows for the complete system of algebraic equations, derived after discretisation of systems of PDEs, to be solved using Newton's method[122]. Newton's method is known to converge quadratically for a suitable initial guess which leads to a robust solution to coupled multi-physics problems. In contrast to Picard, or partitioned, solvers which often converge slowly or fail to converge at all, this is a favourable outcome.

Oomph-lib has been used in a number of studies in fluid mechanics, solid modelling, FSI, and acoustics. Its open-source nature has allowed for the implementation of new functionality specific to cardiac modelling which is capable of interfacing with pre-existing Oomph-lib objects and code.

Oomph-lib is designed for ease of use at the highest level of interaction. Comparatively, little training is required for a user to write their own driver codes and run simulations using the existing high-level objects.

Support is provided for parallel processing using mumps and for use of external linear solvers and preconditioners including scalapack, trillinos, and hypre.

Support and tutorials exist for conversion between medical scans and Oomph-lib meshes. Examples include analysis of fluid flow through bifurcations of blood vessels[123], the geometry of which is derived from MR/CT scans using the Vascular Modelling Toolkit, VMTK[124].

Two-dimensional and three-dimensional idealised geometries can be generated using structured or unstructured techniques. Hard-coded structured meshes can be written in the form of new geometries and meshes, whereas unstructured meshes can be generated using third-party open-source software such as Triangle and Tetgen. Tutorials for both are provided in the online documentation.

4.2 Structures within Oomph-lib

The main components of Oomph-lib are Data, Node, GeneralisedElement, Mesh, and Problem

4.2.1 Data

The most basic elementary data in Oomph-lib is Data. Data stores a double precision number, the value of which is typically determined by a system of equations. The solution of this system will usually require a numbering scheme which describes the unknowns, for this reason, Data also stores an integer which represents the number of the unknown in the global numbering scheme.

Often when solving such equations, DOFs can be pinned. Oomph-lib adopts the convention which enforces that the equation number is set to a negative value if the Data is pinned.

In time-dependent problems, approximations of time derivatives are required. Therefore Data also stores auxiliary variables which describe the values of the unknowns at previous time steps and a pointer to the time-stepper which relates these history values to the time derivatives of the values.

In many problems, the solution is given by vector-valued unknowns. For this reason, Data is allowed to store multiple values, all of which have their own global equation number and history values.

Values stored within Data are read and written to by access functions. The i$^{\text{th}}$ value at present time is accessed via

```
Data::value_pt(i)
```

and the t$^{\text{th}}$ history value with

```
Data::value_pt(t,i).
```

Write functions are similarly defined as

```
Data::set_value(i,val)
```

and

```
Data::set_value(t,i,val)
```

respectively.

4.2.2 Nodes

Nodes in Oomph-lib are derived from Data. They store nodal data within the Data values, in addition to this they store spatial position which is specified at a number of coordinates. The nodal positions are accessed using

```
Node::x(i)
```

and history values with

```
Node::x(t,i).
```

4.2.3 Elements

Elements in Oomph-lib are based upon a four-level inheritance structure[125], separating:

1. the basic functionality that is shared by all generalised elements.
2. the functionality that is shared by all finite elements.
3. the implementation of the finite element geometry i.e. the shape-function-based mapping between local and global coordinates.
4. the representation of the mathematics that describes a specific problem.

The distinction between the four levels of inheritance facilitates code reuse as for example, many geometric elements can be used for the solution of a number of different equations defined at the mathematics level. More information on the different levels of inheritance of Oomph-lib elements is provided in the online documentation[125]

4.2.4 Meshes

A Mesh has several minimum requirements that it must satisfy, these are listed in the online documentation (`http://www.Oomph-lib.org/`) as

- Construct the elements and the Nodes.
- Store pointers to the elements in the *Mesh::Element_pt* vector.
- Store pointers to the Nodes in the *Mesh::Node_pt* vector.

- Set the elements' pointers to their local Nodes.

- Set the positions of the Nodes.

Those which are most relevant to the development of the numerical package are discussed briefly here.

A Mesh stores the finite element representation of a domain. Such a representation consists of finite elements, which themselves consist of nodes. At construction, a Mesh will construct the elements and nodes that it contains. The pointers to the elements and nodes are stored in the vectors *Mesh::Element_pt* and *Mesh::Node_pt* respectively.

In order to function properly, the finite elements must be made aware of which nodes in the mesh are their local nodes. This is often achieved through calling the finite element member functions *construct_node* or *construct_boundary_node*. These functions create a node, pass a pointer to it to the finite element at the appropriate local node number, and return a pointer to the node for the Mesh to store.

The global position of the nodes must be set. This defines the global shape and size of the finite elements.

4.2.5 Problems

Within Oomph-lib, the problem class contains the meshes and additional objects, such as time-steppers required to perform the numerical calculations. Mesh construction, Newton solves, and time-stepping is performed by invoking high-level member functions of the problem class.

4.3 The use of Oomph-lib

The procedure for setting up a driver code with Oomph-lib is extensively detailed in the library's online documentation and the many tutorials found therein[121]. According to the coding conventions set out in the documentation, a driver code will consist of a name space that details parameters and static functions used throughout the code to formulate the physical system, a problem class which outlines the problem being solved, and the main function loop which is executed.

4.4 What has been added to Oomph-lib

In order to simulate the physical processes within the human cardiac system, several additions needed to be developed in line with the Oomph-lib state-of-the-art and integrated into the library. Broadly speaking there were four areas which were added to the library, each containing several sub-structures which had to be developed. These areas are anisotropic electrical diffusion models, computation of non-linear source terms and communication of these terms to the wider library, anisotropic solid mechanics and constitutive laws, and biophysically detailed mesh generation.

The additions most similar to existing Oomph-lib structures are the anisotropic electrical diffusion models, and the anisotropic solid mechanics and constitutive laws. These consist of classes similar in implementation to existing Oomph-lib classes. Anisotropic electrical diffusion models were implemented from scratch, and anisotropic solid mechanics and constitutive laws were implemented by, in part, inheriting from the existing PVD elements.

There were a number of major obstacles to implementing these additions.

The obstacle which required the most time to design, refine, and test was the addition of separate structures by which non-linear source terms are stored, computed, and communicated to the rest of the Oomph-lib classes. These structures needed to be applicable to general cardiac cell models, integrate into Oomph-lib finite elements, operate in both serial and parallel, operate correctly within distributed problems and meshes, efficiently handle memory, and implement operator splitting by means of calculating a partitioned solution alongside Oomph-lib. This functionality had to be implemented in a manner which minimally impacts the structure and form of driver codes and is in line with the existing Oomph-lib coding conventions.

A further addition which required extensive work to implement is the generation of biophysically detailed finite element meshes. These unstructured meshes can often be very large with many millions of nodes and elements and, in general, are not in a format which can be recognised by Oomph-lib. A method of generating Oomph-lib meshes for the simulation of both electrophysiology and mechanical deformation was implemented. Oomph-lib can implement 3D unstructured mesh generation using TetGen. A specific example driver code[123] implements the generation of solid and fluid meshes of an iliac bifurcation from MR/CT images which are converted using VMTK to files required by TetGen. This implementation however is specific to bifurcation-like geometries. Furthermore, since the solid mesh is generated from the MR/CT images of the fluidic region by a conversion code, this implementation only deals with the constant thickness of the vessel walls. In order to simulate anatomical cardiac meshes, a more general method of constructing Oomph-lib meshes from biophysically detailed geometries was required and was implemented. Since Oomph-lib mesh generation procedures require knowledge of element facets on the boundaries on a domain, a robust and efficient method for the identification of boundary facets had to be designed. This unstructured mesh generation was implemented in such a way that, in future work, the corresponding fluidic meshes on the interior of the vessel can be generated for electro-mechanical-fluidic simulations.

Anisotropic electrical diffusion elements were implemented from scratch. The bidomain equations are so dissimilar from all existing Oomph-lib elements that it was simpler to start without using an existing element as a framework. However, the monodomain equation is similar to the unsteady heat equation. The unsteady heat equation was therefore used as a guide for the implementation, although certain substantial changes were made.

The PVD equations represent some of the most complicated structures in Oomph-lib. These implement the solution of the PVD, specified to Cartesian coordinates, for compressible, incompressible, and nearly-incompressible solids. Several satellite elements handle the application of surface traction and communications of forces and pressures to and from fluid elements when used in FSI problems. In order to avoid re-implementing these features and to maximise code re-use, which is a core part of the Oomph-lib ethos, the anisotropic solid PVD classes inherit from the existing Oomph-lib PVD classes. Additions and changes to certain functions were made so that anisotropy can be communicated to the required points of computation. This code reuse means that the anisotropic PVD elements can piggyback on existing functionality which implements FSI.

All additions developed as part of this work are included in the source files located at[126].

4.4.1 Solving for electrical diffusion

Monodomain and Bidomain equations were implemented in the standard way for Oomph-lib. The equations were converted into the weak formulation and the residual and Jacobian contributions from a general element were formulated and implemented.

These finite elements can be used in the normal sense for Oomph-lib, used as a template argument to build a finite element mesh which is then used to solve the equations with or without time-stepping. As such, these equations can be used in monolithic problems without any additional changes. They have a forcing term which represents the transmembrane flux or fluxes of ions and can be set to some analytic function which represents an idealised cell model. Alternatively, if a cell model is implemented as an Oomph-lib element, the elements can be used to construct Oomph-lib multi-physics elements or be used in multi-physics, multi-domain discretisation simulations through using the Oomph-lib multi-physics machinery. It is a requirement, for the implementation of general cell models and operator splitting methods, that these elements be set up in such a way that they can also be solved as part of a wider operator splitting method.

Preparing for operator splitting

In order for general electrical diffusion models to be used with the operator splitting machinery (described later), a base class from which all electrical diffusion classes inherit was implemented as part of this work.

The *BaseCellMembranePotentialEquations* class defines several virtual functions which must be overridden in a diffusion equations class.

The function which defines the index of the DOF which represents the transmembrane potential is given by

```
virtual inline unsigned vm_index_BaseCellMembranePotential()
    const {return 0;}
```

This is virtual in case the element is used for multi-physics via multiple inheritance. In such cases, the index of the DOFs required by this element may have to be shifted to account for those of the other parent elements. Alternatively, the DOFs of the other parent elements may be placed after those of this one using the function

```
inline unsigned max_index_plus_one_BaseCellMembranePotential
    () const {return vm_index_BaseCellMembranePotential()+
    required_nvalue();}
```

This accesses the first DOF index which is not used by this class and is defined by the virtual function,

```
inline unsigned required_nvalue(const unsigned &n) const =0
```

which returns the number of DOFs this element requires. It is implemented as virtual since any derived diffusion model must specify how many DOFs it requires. For instance, the monodomain model has one DOF, whereas the bidomain model has two.

Finally, and most importantly, the equations which represent the diffusion model must be implemented. This is performed by filling the residual vector and Jacobian matrix with the appropriate values via the function

```
virtual void
    fill_in_generic_residual_contribution_BaseCellMembranePotential
    (
Vector<double> &residuals, DenseMatrix<double> &jacobian,
DenseMatrix<double> &mass_matrix, unsigned flag)
```

Details of the fill-in procedure of the residual and Jacobian are given throughout the Oomph-lib documentation and a brief resumé is provided in the relevant sections on the finite element method.

Additional functions that provide source terms and variables such as membrane capacitance or conductivity are defined. This serves two purposes. Firstly it reduces the amount of code which must be written when implementing a new conductance model. Secondly, when used in multiphysics such as when the tissue is deformed during muscular contraction, these functions may change. This can be illustrated by the variable χ which represents the membrane surface area per unit volume in the tissue in the monodomain equations. This variable will, in general, change during the elastic deformation of the tissue. Implementing such functions in the BaseCellMembranePotentialEquations class, therefore, means that a multiphysics version of these functions only needs to be implemented once in the form of a wrapper class.

Of course, this cannot necessarily capture all functionality required by a generic electrical diffusion model. All functionality required by the bidomain equation and monodomain equation has been implemented, any further functionality will have to be implemented as needed.

Relevant example drivers

The Oomph-lib website contains numerous example problems which detail how the library can be used to simulate a myriad of physical systems. Several of these example problems are important for the development of additions to the library which are used in the simulation of the heart. The problems discussed each provide an important step in library development.

The unsteady heat equation, which is already implemented in Oomph-lib, is similar to the monodomain equation. It is given by

$$\alpha\frac{\partial u}{\partial t} + f = \beta\frac{\partial^2 u}{\partial x^2}. \tag{4.1}$$

Both the monodomain equation and unsteady heat equation contain only a single DOF and so the number of DOFs required by their respective Oomph-lib equation classes is also one.

The monodomain equation contains a weighting term in the second spatial derivative which reflects variation in diffusion in the preferred directions within the tissue.

Monodomain equations

The monodomain model over a domain Ω with boundary $\partial\Omega$ is given by 2.57 and 2.58 where V is the transmembrane potential, $\chi(\underline{x})$ is the membrane surface area per unit volume of tissue, $C_m(\underline{x})$ is the membrane capacitance per unit membrane surface area, $i_{\text{ion}}(\underline{x})$ is the transmembrane ionic current per unit membrane surface area, $\mathbf{D}(\underline{x})$ is the conductivity tensor, and $\underline{n}(\underline{x})$ is the outer unit normal to the domain boundary $\partial\Omega$.

Implementing the monodomain equation using the unsteady heat equation as a framework required several changes. The parameters α and β must be allowed to vary spatially. β must, instead of being a scalar, contain the entries of a matrix which represents the conductivity tensor in the suitable coordinate basis. Additionally, β must also be incorporated into the second derivative. Finally, the variable χ must be introduced to the formulation and also be permitted to change spatially throughout the element.

The weak formulation of 2.57 is determined by multiplying by a test function ψ and integrating over the domain Ω

$$\int\limits_{\Omega} \left\{ \chi(\underline{x}) \left(C_m(\underline{x}) \frac{\partial V}{\partial t} + i_{\text{ion}}(\underline{x}) \right) \psi + (\mathbf{D}\nabla V) \cdot \nabla\psi \right\} \mathrm{d}\nu = \oint\limits_{\partial\Omega} \psi \mathbf{D}\nabla V \cdot \underline{n} \, \mathrm{d}s, \tag{4.2}$$

where $\mathrm{d}\nu$ is the volume element, and $\mathrm{d}s$ is the surface area element. The boundary condition 2.58 is then applied to arrive at the residual equation which is implemented by the monodomain equations class

$$\mathcal{R} = \int_{\Omega} \left\{ -\chi(\underline{x}) \left(C_m(\underline{x}) \frac{\partial V}{\partial t} + i_{\text{ion}}(\underline{x}) \right) \psi - (\mathbf{D}\nabla V) \cdot \nabla \psi \right\} \mathrm{d}\nu. \tag{4.3}$$

In the finite element formulation, this corresponds to the residual element contributions from the e$^{\text{th}}$ element enclosing the region Ω_e,

$$\mathcal{R}_k = \int_{\Omega_e} \left\{ -\chi(\underline{x}) \left(C_m(\underline{x}) \frac{\partial V}{\partial t} + i_{\text{ion}}(\underline{x}) \right) \psi_k - (\mathbf{D}\nabla V) \cdot \nabla \psi_k \right\} \mathrm{d}\nu, \tag{4.4}$$

and the Jacobian element contributions

$$\mathcal{J}_{kj} = \int_{\Omega} \left\{ -\chi(\underline{x}) \left(C_m(\underline{x}) \psi_j w + \underline{\dot{X}}(\underline{x}) \cdot \nabla \psi_j \right) \psi_k - (\mathbf{D}\nabla \psi_j) \cdot \nabla \psi_k \right\} \mathrm{d}\nu. \tag{4.5}$$

During residual and Jacobian fill-in procedures, the variables $\chi(\underline{x})$, $C_m(\underline{x})$, $\mathbf{D}(\underline{x})$, and $i_{\text{ion}}(\underline{x})$ are determined via spatially dependent functions which are specified by the user when writing the driver code. These functions are inherited from the BaseCellMembranePotentialEquations class.

Bidomain Equations

The bidomain equations contain several terms which are similar to the monodomain equation and solve for two different variables. Their implementation cannot be simply based on any existing Oomph-lib elements and they must be implemented in their entirety.

The contributions to the residual for the eth element are given by

$$\mathcal{R}_k^V = \int_{\Omega_e} -\chi(\underline{x})\left(C_m(\underline{x})\frac{\partial V_m}{\partial t} + (I_{\text{ion}}(\underline{x}) - I_1(\underline{x}))\right)\psi_k \\ - (\mathbf{D}(\underline{x})\nabla(V_m + \phi))\cdot\nabla\psi_k \,\mathrm{d}\nu \tag{4.6}$$

$$\mathcal{R}_k^\phi = \int_{\Omega_e} - (\mathbf{D}(\underline{x})\nabla V_m)\cdot\nabla\psi_k - ((\mathbf{D}(\underline{x}) + \mathbf{E}(\underline{x}))\nabla\phi)\cdot\nabla\psi_k \,\mathrm{d}\nu. \tag{4.7}$$

and the Jacobian entries

$$\frac{\partial\mathcal{R}_k^V}{\partial V_j} = \int_{\Omega_e} -\chi(\underline{x})\left(C_m(\underline{x})\psi_j w + \underline{\dot{X}}(\underline{x})\cdot\nabla\psi_j\right)\psi_k - (\mathbf{D}(\underline{x})\nabla\psi_j)\cdot\nabla\psi_k \,\mathrm{d}\nu \tag{4.8}$$

$$\frac{\partial\mathcal{J}_k^\phi}{\partial V_j} = \int_{\Omega} - (\mathbf{D}(\underline{x})\nabla\psi_j)\cdot\nabla\psi_k \,\mathrm{d}\nu. \tag{4.9}$$

$$\frac{\partial\mathcal{R}_k^V}{\partial \phi_j} = \int_{\Omega_e} - (\mathbf{D}(\underline{x})\nabla(\psi_j))\cdot\nabla\psi_k \,\mathrm{d}\nu \tag{4.10}$$

$$\frac{\partial\mathcal{J}_k^\phi}{\partial \phi_j} = \int_{\Omega} ((\mathbf{D}(\underline{x}) + \mathbf{E}(\underline{x}))\nabla\psi_j)\cdot\nabla\psi_k \,\mathrm{d}\nu. \tag{4.11}$$

4.4.2 Adding non-linear point source terms

So far, electrical diffusion elements have been implemented in the standard way for Oomph-lib elements. However, if operator splitting is to be employed then several additional functionalities are needed. In this case, the cell model, instead of being an Oomph-lib element or an ideal forcing function, is represented by an external container which must be in some way linked to the standard Oomph-lib objects. Among other requirements, the transmembrane potential has to be communicated to and from the cell model, active strain needs to be communicated from the cell model to PVD elements, the elastic strain may need to be communicated from PVD elements to the cell model, and physiologically detailed diffusion tensor and cell alignment information may need to be communicated from the cell model to the diffusion and PVD elements. A wrapper of the conductance element handles this communication of data between the Oomph-lib elements and the cell model containers. Additionally, a wrapper of the Mesh class sets up the correspondence between the cells and the nodes/elements in the mesh, stores the cell containers, and provides helper functions which facilitate operator splitting solutions as well as the construction of the cell mesh.

Previous iterations of the implementation

Several implementations of non-linear point source term implementations were attempted and after testing were further refined. These included attempts to introduce non-linear point source terms as Oomph-lib finite elements and in doing so hijack the Monolithic Newton solver to solve for any point-source variables. These attempts led to either unwieldy implementations or inefficient computation time. As such the final implementation represents the most efficient and elegant attempted solution.

The Cell Mesh

CellMeshBase operates as a wrapper to the Mesh class. It adds functionality required by a mesh to operate with the non-linear source terms and provides useful helper functions which aid in constructing cell models, setting up initial conditions, outputting, and performing partitioned time-stepping via operator splitting.

Instances of cell models are stored within *CellMeshBase* which handles their construction and the communication of variables to and from other multi-physics elements. Cell models are stored via pointers within a dynamically allocated vector. Cells are associated with nodes in the mesh in a one-to-one correspondence, this allows for operator splitting to be performed without the need to project the membrane potential from the cells to the nodes or vice-versa when the cells have the same resolution as the electrical diffusion mesh. As such, the value of the underlying node can simply be copied to the relevant cell, or the value in the cell can be copied to the relevant node.

In order to be used with an Oomph-lib mesh, the user must define the function *BuildCells* which handles the construction of cells at the nodes in the mesh. This is performed by calling the templated function *build_cell_at_node* for each of the nodes in the mesh. The template argument is the class of the cell model which is to be built. *build_cell_at_node* accepts two arguments, the node number in the mesh, and the number of backup data the cell model requires. Backup data is utilised by the cell model to store and restore solutions internally. These operations are useful when implementing adaptive operator splitting methods. *build_cell_at_node* returns a pointer to the constructed node so that any other setup required by the model, such as assigning cell type, setting the external electrical stimulus function, and setting fibre alignment and diffusion coefficients, can be performed. All other setups, such as constructing a correspondence with the correct node and elements in the mesh is automatically performed.

Any child class which inherits from *CellMeshBase* must call the *CellMeshBase* function *FinalizeMeshSetup* within its constructor. *FinalizeMeshSetup* builds the lookup tables for nodes and elements, calls *BuildCells*, and finally sets up for parallel computation if Oomph-lib has been built with *MPI*. *FinalizeMeshSetup* must be called in the child class constructor since *CellMeshBase* requires the nodes and elements of the mesh to already be created and so cannot be called in the *CellMeshBase* constructor.

Node-Element lookup tables are generated within *FinalizeMeshSetup* by a call to the function *BuildNodeElementTables*. This constructs the vector of pairs *Elements_ containing_ node*. The index of the vector is the global node number, and each element in the vector consists of the list of numbers of the elements which contain that node and the local node number of the node within those elements.

During calls to the function *build_ cell_ at_ node* in *BuildCells*, the function *add_ cell_ to_ node* is called in which the cell is provided information about its corresponding node, an element that node exists in, and the local and global coordinate of the node. This function relies on *Elements_ containing_ node* which is assembled only once after the elements and nodes have been constructed during the call to *FinalizeMeshSetup*. This expedites the construction of cell models at the nodes of the mesh since corresponding elements can be found quickly using *Elements_ containing_ node*.

When a cell model is built at a node, all elements which contain the node are provided with a pointer to the cell along with the local index of the node within the element. The elements are then able to access data in the cell model, for example, to access fibre alignment or active stress, and the cell model is able to access and modify data in the elements, such as when assigning initial membrane potential.

Finally, if Oomph-lib has been built with *MPI* enabled, the function *SetupDataIndices* is called. This prepares storage for the communication of cell solutions across processors. For large numbers of cells, it may be beneficial to solve them in parallel rather than in series. In such cases, the list of cells within the mesh is partitioned and each processor computes the solution to only a subset of the total cells. The solution stored on each processor must then be communicated to the other processors. In order to do so, vectors of local solutions must be assembled by each processor, which are then sent to all processors through a call to *MPI_Allreduce*. In some cases, such as caricature cell models consisting of very few simple equations, communication of data between processors may be more computationally expensive than calculating the solution to the cell model. For these circumstances the compiler flag *Oomph_FORCE_SERIAL_SINGLE_CELL_SOLVE* can be defined, in which case each processor computes the solution to all cells independently and serially. Taking time steps with all cells in the mesh either in parallel or in series is handled automatically through a call to Take_time_step_with_all_cells_in_mesh. This function accepts as its arguments the time-step Δt, and a pointer to the Oomph-lib problem class which is used to access the MPI distribution.

Large-scale simulations can consume a huge amount of memory. By default, a complete copy of the Oomph-lib Problem is stored on each processor. Therefore when simulations are run in parallel they consume roughly that volume of memory times the number of processors they are run on. The size of a problem can therefore be limited by the amount of memory available on each processor. Additionally, mesh adaptation does not benefit from parallelisation since each processor must adapt its own copy of the meshes within the problem. Oomph-lib has a system for dealing with this, called distribution. In distributed problems, the problem is initially built in serial. The *Problem* class member function *distribute* is called which in turn calls the *Mesh* class member function *distribute* for each mesh in the *Problem*. This, for each mesh, removes from each processor all but that processor's share of elements and nodes so that only a subset of the finite elements from each mesh are stored on each processor. In this way, the complete problem is only stored in memory once. As a result, distributed problems consume substantially less memory when run in parallel than their non-distributed versions. Furthermore, the time taken to perform mesh adaptation can be substantially decreased since each processor now only needs to perform the adaptation procedure on a subset of the total elements.

Simple problems within Oomph-lib can be fairly trivially distributed. More complicated problems require more care and work at the point the driver is implemented.

The wrapper class *CellMeshBase* adds additional member data and member functions to the Mesh class. Such additions must be handled properly when the problem is distributed. Therefore, in order to allow for larger problems using the *CellMeshBase* to be run in parallel, distribution has been re-implemented in *CellMeshBase*. Distributed problems more efficiently fill memory, however, since data must be communicated between the regions solved by different processors, additional computational overhead is introduced during the Newton step and single cell solve. This overhead may be offset to some degree by the introduction of additional processors, however, an optimal number of processors will likely exist for each problem. Further improvements may be made to a problem by minimising the size of the boundaries between the distinct distributed regions since this will minimise the communication required between the processors.

Since cardiac simulations are often characterised by small stiff sub-regions in the form of a propagation wave, it follows that a small number of processors may bear a disproportionately large fraction of the cost required to solve for a single time step. It is unclear however how this could be mitigated since the distribution of a problem is often costly, and would have to be performed several times as the wave front travels through the domain. This may not prove to be such an issue since cell models often do not require significantly more computation when experiencing upstroke as compared with other times unless adaptive time-stepping is implemented.

CellMeshBase can handle any number of different cell models. Since the cells operate fully independently of one another there is no conflict with data storage or performing operations such as time-stepping. If the cell model data was stored as DOFs within an Oomph-lib mesh then each node in the mesh would require an arbitrary number of DOFs. Such implementations may be possible within Oomph-lib, and indeed some elements, such as fluid meshes with continuous pressure allow for nodes with differing numbers of DOFs. However, the number of DOFs required by each node is known *a-priori* since it is clearly specified within the element class. This was one of the main contributing factors in the decision to extricate the cell models from the Oomph-lib finite element paradigm and instead store them within an entirely distinct, newly implemented structure.

As with Oomph-lib mesh classes, *CellMeshBase* requires a template argument which specifies the finite element class. In order to ensure that this finite element class is compatible with the *CellMeshBase* and cell models, this element must inherit from the class *BaseCellMembranePotentialEquations*. In order to handle Paraview output with more than one cell model, an alternative Paraview output function is provided for *CellMeshBase* derived meshes. The function *output_paraview_per_cell_model* will produce a vtu output file for each cell type the mesh contains. As arguments, it requires the directory in which the vtu files are to be generated, the generic root file name (e.g. *cell_sheet*, etc), the output number which is appended to the names of each of the files, and the number of points to be plotted along the edges of each finite element. *output_paraview_per_cell_model* returns a vector of pairs of strings, each element of the vector contains the cell model name and the corresponding vtu file to which that cell model's data was output. The user can then use this information in the driver to append the appropriate .vtu file to each .pvd file. A separate vtu file must be generated for each cell model within the mesh since each cell model may have an arbitrary number of variables with no relation to the variables in the other cell models.

The Fully Partitioned wrapper class

Data from the cell model is required for multi-physics simulations. Diffusion models require the diffusion tensor and the membrane potential from the cell models, and external elements such as *AnisotropicSolidEquations* require fibre and sheet alignment and active strain or stress. Additionally, the cell model requires data, such as membrane potential or strain, from these elements. This data must be suitably interpolated to the coordinates at which the cells exist, or where the external elements require it. In order to facilitate this, the wrapper class *FullyPartitionedCellEquations* is provided. This wrapper adds functionality which allows for cell data to be interpolated to Oomph-lib elements. It contains the pointers to cells associated with each local node in the element, as well as procedures for interpolating the data from the cells to any local coordinate within the element as well as adding new dummy integral points at the nodes of the element, so that external data may be interpolated directly to the coordinate of the cell via multi-physics.

To interpolate cell data for use by finite elements, the normal Oomph-lib interpolation procedure is undertaken whereby basis shape functions are evaluated at the interpolation point and the cell data at each node is compiled together with a weighted sum. Several functions for accessing specific data have been implemented, including *get_interpolated_cell_alignment* which calculates interpolated local fibre, sheet, normal directions, *get_ interpolated_ diffusion_ tensor* which calculates the diffusion tensor at a local coordinate, and *get_ interpolated_ active_ strain_from_ cell_ model* which interpolates active strain generated by the cell models at each node.

In order to access data from external elements at each cell, integral points are required to be aligned with the nodes of the element. This is because the Oomph-lib function *setup_ multi_ domain_ interactions* calculates the local coordinate in elements of external meshes at which local coordinates of each element lie only performs such operations at local integral points. This is because such multi-domain interactions so far have only been required at integral points for the calculation of contribution to residual and Jacobian elements which take the form of an integral arising from weak formulations.

To provide these additional integral points, a new quadrature class which is based on the Oomph-lib Gauss quadrature elements have been used. These contain additional integral points with local coordinates aligned with those of the nodes in the element. The integral weights associated with the additional integral points are zero so that their existence does not affect the value of integrals computed using the quadrature. The member function *ipt_ at_ node* returns the appropriate integral point for the requested local node number which can then be used by *Elemen-tWithExternalElement* procedures to access the required external data. Since these quadrature schemes are only used in meshes which contain specific elements, such as those which inherit from *MonodomainEquations*, *fill_ in_ generic_ residual_ contribution_ BaseCellMembranePotential* of these elements can be written such that only the genuine integral points associated with the integral scheme are iterated over.

FullyPartitionedCellEquations also handles generic Paraview output of cell data which is called by the *CellMeshBase* containing the element when required.

The Cell Model class

Often in cardiac cell models, different types of variables require different time-stepping schemes, such as forward Euler for ion concentrations, Rush-Larsen for gating variables, and dense linear solvers for mechanical contraction-related variables. It is often easier to hard code these solvers for each variable rather than defining the derivative and using a general time-stepper such as those used in Oomph-lib. There is also no guarantee that a cell model variable will be linear or explicit in its first-time derivative as is a requirement for Oomph-lib explicit time-steppers. This would restrict the use of implicit time-stepping to the cell models which would require the inversion of dense matrices and is often less efficient than explicit time-stepping. Furthermore, as is mentioned in the discussion of the *CellMeshBase* class, cell models can contain an arbitrary number of variables due to their differences such as the number and types of ion channel models. As such, it would be particularly difficult to engineer an Oomph-lib finite element which can implement, potentially many simultaneously, generic cell models.

CellModelBaseFullyPartitioned contains all the functions required for solving arbitrary cell models and linking their solution to Oomph-lib finite element meshes with operator splitting. It is intended that *CellModelBase-FullyPartitioned* allows for generic cell models to be implemented easily without major modifications. *CellModelBaseFullyPartitioned* stores the DOFs which represent the single-cell variables, such as ion concentrations and gating variables, as generic double precision variables, similarly to Oomph-lib *Data*/*Node* class. However, this data is stored with a difference in the ordering of the information in contiguous memory when compared to the Oomph-lib *Node/Data* class. The information in Oomph-lib *Node/Data* is stored in blocks corresponding to each variable, accessing the next pointer in the block returns the value at the next time step. This is not suitable for generic cell models, since these are often implemented as accepting an array which represents the current state of the cell variables. In order to facilitate passing the information as an array to the *CellModel* function *TakeTimestep*, it is required that the information is stored in blocks corresponding to each state. In this way, accessing the next pointer within each block returns the next variable at that backup. Information is stored in backup blocks within the generic cell model base class in order to facilitate adaptive operator splitting by storing and restoring solution states. Adaptive operator splitting may additionally necessitate the storing and restoring of solutions due to large time-steps attempts diverging or becoming unstable. In such cases, it is useful to have a safe backup of the simulation at some previous time to which the entire solution space can be reverted and reattempted with smaller time steps. The number of backups for the cell variables is specified by an argument in the cell class constructor and cannot be changed after the cell has been constructed. This number of backups is passed to the *CellModelBaseFullyPartitioned* constructor by the *CellMeshBase* function *build_ cell_ at_ node*.

CellModelBaseFullyPartitioned contains the function *get_ cell_ model_ name* which returns the name of the cell model. This is essential for the generic output of cell data. Since a mesh containing cells can feasibly contain any number of cell models, e.g. when the mesh contains several regions of the heart each with distinct modelling characteristics, it is important that any Paraview output be handled separately for each cell model. This is handled automatically by the *CellMeshBase* as has already been discussed.

Within the constructor of the cell class the vectors *Names_ Of_ Cell_ Variables* and *Names_ Of_ Output_ Data* must be defined to contain the names of all time-varying DOFs the cell model handles. This list must also contain the membrane potential since, during operator splitting, differing versions of the membrane potential are stored by each cell. The lengths of these vectors inform the base cell class how much memory to allocate for the cell model, as well as what to label the Paraview variables during the output of the solution.

The function *index_ of_ membrane_ potential_ in_ cell_ data* must be defined to return the index within the *Names_ Of_ Cell_ Variables* at which the membrane potential is stored. This is used for outputting and for functions which assign the membrane potential to the cell model or underlying node.

The function *get_ initial_ state_ variable* accepts as an argument the index of the variable for which the initial state is returned. The output of this function must be determined for each cell model and can vary depending on purpose and implementation. It is often used to return the state of cell model variables after a number of conditioning external stimuli have been applied and the cell model has reached a consistent resting state.

CellModelBaseFullyPartitioned contains several generic functions which must be overridden in order to implement a new cell model. *TakeTimestep* accepts as arguments the time-step to be taken, Δt, the current simulation time t, and the array of current cell variables. The function must be implemented in such a way that the array of variables is updated to reflect the values at time $t + \Delta t$. This time-stepping can be performed by a call to an external numerical library such as Mumps or CVODE, or by a user-implemented method such as Rush-Larsen or explicit Euler. *TakeTimestep* is never called by the driver code and is instead called by the function

Take_ Time_ Step which accepts only the time-step length as an argument. *Take_ Time_ Step* itself is often only called by a call to the *CellMeshBase* function *Take_ time_ step_ with_ all_ cells_ in_ mesh* which accepts as arguments the time-step length and the Oomph-lib Problem class. This function handles any potential parallelisation and distribution of the cell mesh.

The function *get_ output* provides a means by which the cell model can compute information which may be required by other processes. It accepts as arguments the state variables and an output vector. The output variables are specified by their names in the vector *Names_ Of_ Output_ Data*. *get_ output* is called automatically after each time the cell model takes a time step and the values are stored in an array within the cell model so that the function does not need to be called each time this additional information is required. The generated information is output during the generic Paraview output procedure.

The function *GetActiveStrain* is defined to specifically calculate the active stress or strain which is generated by the cell model. It is, by default, implemented to return a default value of 0, since most implementations use the active stress decomposition and a value of 0 specifies no active stress. Active stress or strain could be contained within the *get_ output* function however, since mechanical simulations are a specific purpose of this development, it is more convenient when implementing multi-physics elements for it to be specified as its own function.

Once these functions and variables have been defined the cell model is ready for use in single-cell as well as multi-physics simulations.

The cell model constructor must end with a call to the base cell class function *FinalizeConstruction*, which takes the sizes of the vectors *Names_ Of_ Cell_ Variables* and *Names_ Of_ Output_ Data* as well as the number of backup states required and suitably allocates memory for the cell model.

During adaptive operator splitting, it is often necessary to store and restore solutions. The cell model solution can be stored in one of the backup states through a call to *Store_State_In_Backup* which accepts an unsigned integer which specifies the backup the state is to be stored in. Backup 0 corresponds to the current solution so should not be written to since such an attempt will not result in the solution being saved. If the library is compiled with the -*DPARANOID* flag, attempting to back up to index 0 will result in an error being thrown. The current solution can be restored from one of the backups through a call to *Restore_State_From_Backup* which accepts the index of the desired backup as an argument. Similarly, this function will throw an error if the user attempts to restore from backup 0.

During adaptive operator splitting, since alternating solutions of varying step lengths are taken, the time according to the cell models will change differently to that of the Oomph-lib problem class. However, at the end of each operator splitting solve the time values must be identical in order for the operator splitting method to make physical sense. In order to facilitate checking if this is the case, the function *check_if_time_is_consistent* can be called to check if the argument matches the current value stored in the cell model and returns true if they are and false if they are not. By default this function checks if the values are within 10^{-9} of each other, however, this value can be modified by the user. Due to the numerical round-off, the time values may drift from one another. Over the course of long simulations with many time steps, this drift could potentially become significant. The function *synchronise_time* is therefore provided to allow for the user to force synchronisation of time according to the cell model to the value which is provided as an argument. Such a function should only be called once the operator splitting technique has rigorously been tested with *check_if_time_is_consistent* to ensure that the two values are within the specified tolerance and only diverge after many iterations of the operator splitting algorithm have been taken.

Most cardiac cells are sensitive to electrical stimuli. Such stimuli could result from electrodes applied to the tissue or from nerve cells which transmit an electrical impulse to the tissue. The *CellModelBaseFullyPartitioned* class contains a pointer to an external function which specifies such an applied stimulus. This function follows the usual form of source terms in Oomph-lib and must be specified by the user. It can then be passed to the cell, similar to the method of assigning source functions to Oomph-lib finite elements, by using the access function *stim_func_pt*. In the computation of the cell model timestep, the cell model can access this source term using the protected member function *get_stimulus* which accepts the current time, according to the cell, as an argument. The arguments representing spatial coordinates in the function *stim_func_pt* are automatically filled by the base cell class and are defined when the cell is constructed by a *CellMeshBase* class derived mesh.

If the cell is not built by a suitable mesh, for example when being used for single-cell simulations, the coordinates are all given the default value of 0 with the local and global coordinates set to the 3D origin. The user may override the coordinate values in the case of single cell simulations using the functions *set_local_coord* and *global_coord*. For safety, if these are called for a cell generated as part of a *CellMeshBase* then an error will be thrown since it is unlikely to be physically meaningful to move a cell once it has been built in a tissue mesh.

Fibre, sheet, normal alignment as well as electrical conductivity coefficients in these directions are stored within the *CellModelBase*. Although these quantities are not used by the cell model instead the diffusion or solid models, this storage decision makes sense from a practical perspective. When generating meshes from anatomical geometries, fibre alignment is provided node or element-wise. As such, storing and accessing this data is simplified by storing it within the cells and accessing it via interpolation using functions contained within the *FullyPartitionedCellEquations* class.

Several models of the dynamics of cardiomyocytes are implemented within the context of the numerical package in order to demonstrate its applicability to general physiological models. Three models were selected, the Coleman-Ni-Zhang (CNZ) human atria model, a version of the Tusscher-Noble-Noble-Panfilov (TNNP) human ventricle model, and a version of the HRd human ventricle model. Additionally, several caricature models were implemented for testing purposes.

4.4.3 Solving the diffusion equations with operator splitting

Due to its widespread application to the solution of simulations of myocardial function, the use of operator splitting methods is emphasised in the design of the handling of diffusion and cell equations by the numerical package.

One important characteristic of operator-splitting methods is the communication of data from the solution of one operator to the initial conditions of another. In this package, this communication is handled by the structures which conduct the solution of the cell equations.

For the operator splitting techniques outlined in 3.5.3, solution of the system of equations is performed by alternately solving the individual operators. After each operator is solved the solution is projected to the object which handles solution of the other differential operator. The procedure of solution and projection is outlined in figure 4.1. The objects which handle solution of the individual operators must keep track of time according to the solution to that operator. That is, the current time according to an operator only changes when that operator is solved. This is essential to ensuring that any applied source functions which are explicit in time are evaluated correctly during time-stepping.

It was a principle intention that the inclusion of operator splitting had a minimal impact on the structure of time-stepping in driver codes. Unless additional physics, such as solid and fluid mechanics, are to be solved separately from the electrophysiology problem there are very few changes required to perform time-stepping on the problem. The entire procedure for implementing Strang-splitting with the developed machinery is presented below.

If the Oomph-lib problem is to be solved with adaptive time-stepping, then it must be ensured that the total time which is solved over is exactly that required by the operator splitting method. This can be achieved with a while loop and capping the maximum time-step used by the duration left to be covered. This procedure is illustrated in the code snipped below.

```
const double t_pre = this->time_pt()->time();
double dt_current = dt*operator_splitting_weight;
const double t_post = this->time_pt()->time()+dt_current;

while(this->time_pt()->time()<t_post)
{
    dt_current = adaptive_unsteady_newton_solve(dt_current,
        error_target);
    dt_current = std::min(dt_current,(t_post-this->time_pt()->
        time()));
```

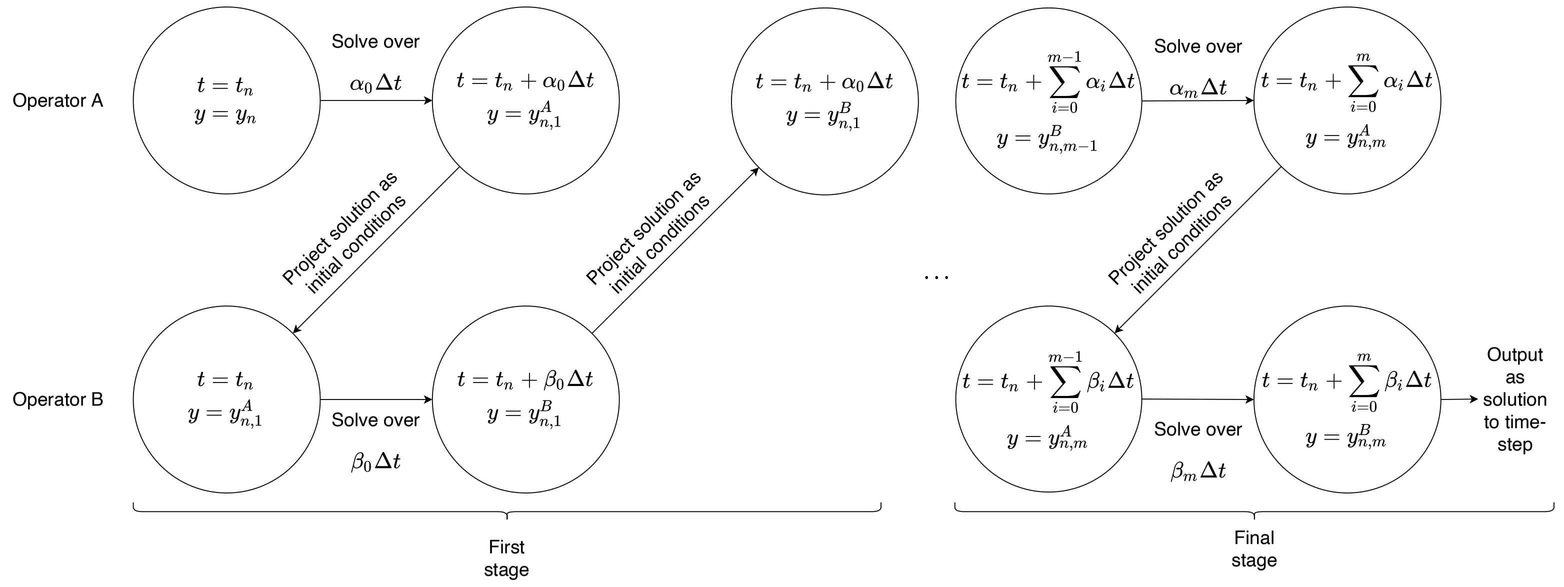

Operator A
Operator B
$t = t_n$
$y = y_n$
Solve over
$\alpha_0 \Delta t$
$t = t_n + \alpha_0 \Delta t$
$y = y^A_{n,1}$
Project solution as initial conditions
$t = t_n$
$y = y^A_{n,1}$
Solve over
$\beta_0 \Delta t$
$t = t_n + \beta_0 \Delta t$
$y = y^B_{n,1}$
Project solution as initial conditions
$t = t_n + \alpha_0 \Delta t$
$y = y^B_{n,1}$
First stage
…
$t = t_n + \sum_{i=0}^{m-1} \alpha_i \Delta t$
$y = y^B_{n,m-1}$
Solve over
$\alpha_m \Delta t$
$t = t_n + \sum_{i=0}^{m} \alpha_i \Delta t$
$y = y^A_{n,m}$
Project solution as initial conditions
$t = t_n + \sum_{i=0}^{m-1} \beta_i \Delta t$
$y = y^A_{n,m}$
Solve over
$\beta_m \Delta t$
$t = t_n + \sum_{i=0}^{m} \beta_i \Delta t$
$y = y^B_{n,m}$
Output as solution to time-step
Final stage

Figure 4.1: Illustration of solution-projection protocol when applying operator splitting. The object that handles each operator keeps track of solution time independently of the other. This way any source terms which are explicitly dependent on time are handled appropriately during the solution of each operator. In the context of solving cell models with the developed numerical library, the Oomph-lib problem class keeps track of time for the operator which corresponds to electrical diffusion, and the cell models themselves keep track of the time for the other operator.

```
}
```

It remains to be determined if the error estimates of Oomph-lib time-steppers are appropriate in this instance. Considering BDF time-steppers, the velocity of the DOFs is calculated during the shift of the time values. This is performed within the call to *adaptive_ unsteady_ newton_ solve* and so is performed after the membrane potential has been projected from the cell models. This may therefore overestimate or underestimate the error for the first adaptive time-step, although this aspect of time-stepping with operator splitting within Oomph-lib is yet to be rigorously investigated.

Crank-Nicolson is a time-stepping method which is commonly applied to solving the bidomain and monodomain models. It is synonymous with the Trapezoid (TR) time-stepper as it is often referred to in other literature. One might therefore assume that Crank-Nicolson can be used by building the diffusion elements with the TR time-stepper. Use of this time-stepper in Oomph-lib requires that the governing equation is explicit and linear in the time derivative and that the residual for a given DOF is written in the form $r = f(t, u) - dudt$. Initially, the TR time-stepper requires a call to *Setup_ initial_ derivative* so that the derivative at the first time step is correctly stored within the history values. The time derivative at the current time step is then calculated before time values are shifted by simply taking the difference with the previous value. This poses potential issues. Since the value at the previous time-step is being changed during the projection from the cell models, the function *Setup_ initial_ derivative* may have to be called every time the solution to the cell operator is projected to the diffusion DOFs. This could be very expensive. Additionally, through inspection of the implementation, the function *Setup_ initial_ derivative* appears only to work when the TR time-stepper is the only time-stepper used in the problem. This prohibits the use of this time-stepper alongside other sub-physics, such as FSI, since they will in general use their own, different, time-steppers. That is unless those meshes are removed from the problem whenever the electrophysiology sub-physics is being solved, however adding and removing meshes and rebuilding the global mesh are expensive procedures to undergo at every stage of calculating a time step.

During validation of the electrophysiology implementation with operator splitting, it was found that the TR time-stepper either does not perform as well as other time-steppers or fails to converge at all. This was the case when both the function *Setup_initial_derivative* was called before an unsteady Newton solve and when it was not.

A hard-coded monodomain model element which uses Crank-Nicolson by explicitly calculating the derivatives at the current and previous time-step was also implemented. This element does not require calls to *Setup_initial_derivative* since such derivatives are calculated on the fly. This is more expensive than the TR time-stepper since the derivatives are calculated more cheaply during shifting the time values. It was found, as shown in the validation data, that this explicit Crank-Nicolson element does converge appropriately and results in a smaller error than backward Euler, as is expected. In order to appropriately allocate history values to the DOFs, this element must be built with the BDF$\langle 1\rangle$ time-stepper. This element cannot currently be used with adaptive time-stepping since it has not been determined if the error measure provided by the BDF$\langle 1\rangle$ time-stepper is appropriate.

It was also found that higher-order time-steppers, such as BDF$\langle 2\rangle$, are less accurate than BDF$\langle 1\rangle$ when used within operator splitting. This could be because the value at the previous time-step, given by the solution to the cell model, actually represents a virtual solution from which the diffusion model should be solved. Use of this virtual solution alongside the genuine solutions at older time steps to approximate the time-derivative of the membrane potential may therefore be fundamentally incorrect. Indeed this appears to be the case since, as is shown in the validation data, that BDF$\langle 4\rangle$ is even less accurate than BDF$\langle 2\rangle$ which is less accurate than BDF$\langle 1\rangle$. Therefore, if higher-order solutions to the diffusion model are required, then explicit implementations of time-steppers, such as implicit RKF or single-step Obrechkoff methods, could be required. However, implicit RKF methods introduce additional DOFs to the elements which may negatively impact solution efficiency. Furthermore, the calculation of an error estimate will likely require the development of alternative dummy time-steppers.

4.4.4 Solving the electrophysiology problem with multiple domain discretisations

If it is required that there are different resolutions of diffusion elements and the cell point sources then changes to the method of solution are required. It was found that this is minimally implemented with two meshes and projection of the solution from one mesh to another via Oomph-lib multi-physics procedures.

Operator splitting is characterised by the assignment of the initial conditions of one differential operator from the solution given by solving the other operator. The normal Oomph-lib element with external element framework cannot necessarily be used directly to achieve this, since directly assigning the solution from one mesh to another and back again does not necessarily leave the solution unchanged. Doing this can result in spurious diffusion of the solution in the absence of solving any of the equations. This is illustrated in the figure 4.2 in which two meshes, one cell mesh, denoted $\mathcal{C}$ and one diffusion mesh, denoted $\mathcal{D}$, have misaligned nodes. It can be shown, through consideration of the values calculated through simple projection at integral points, that spurious diffusion can easily occur. The membrane potential values in the cell mesh are labelled V_1^* and V_2^* and the values in the diffusion mesh are labelled V_1, V_2, and V_3. Basis functions in the cell mesh are labelled ϕ_1^* and ϕ_2^*, and basis functions in the diffusion mesh are labelled ϕ_1, ϕ_2, and ϕ_3.

The first node in the cell mesh is aligned with the local coordinate s_1 in the first diffusion element, and the second node in the cell mesh is aligned with the local coordinate s_2 in the second diffusion element. Similarly, the second diffusion node is aligned with the local coordinate s_2^* in the cell element. Projecting from the diffusion mesh to nodes in the cell mesh gives the values

$$V_1^* = V_1\phi_1(s_1) + V_2\phi_2(s_1)$$

and

$$V_2^* = V_2\phi_2(s_2) + V_3\phi_3(s_2).$$

Then projecting from the cell mesh to the second node in the diffusion mesh gives the value

$$\begin{aligned} V_2 &= V_2^*\phi_2^*(s_2^*) + V_3^*\phi_3^*(s_2^*) \\ &= V_2(\phi_2(s_1)\phi_2^*(s_2^*) + \phi_2(s_2)\phi_3^*(s_2^*)) + V_1\phi_1(s_1)\phi_3^*(s_2^*) + V_3\phi_3(s_2)\phi_3^*(s_2^*) \end{aligned} \tag{4.12}$$

which is not necessarily equal to the original value at that node. Contributions from the neighbouring nodes in the diffusion mesh have been included.

In order to project the solution from one mesh to the other, the local

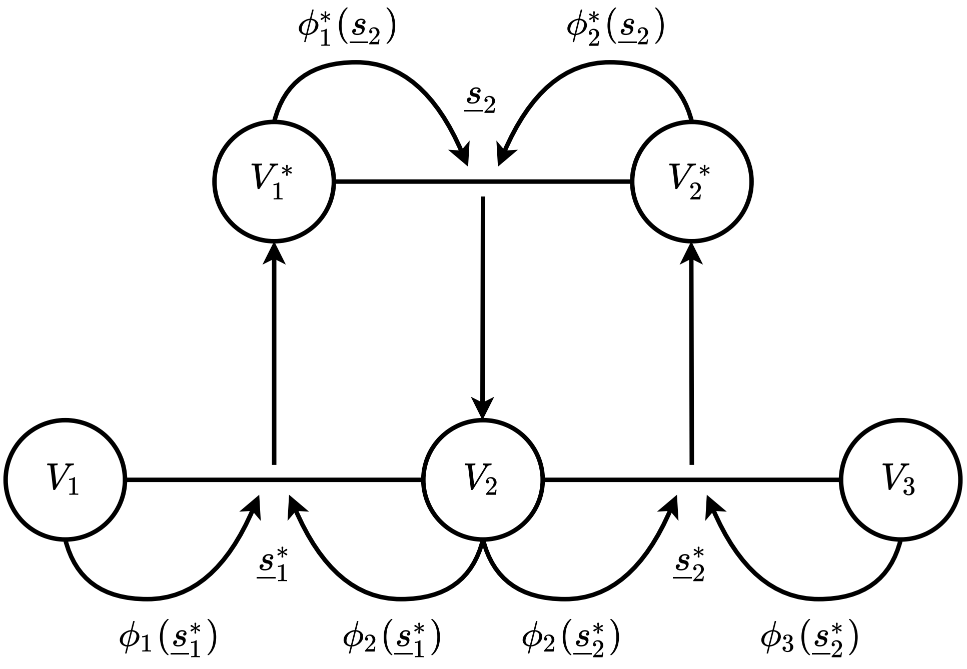

$\phi_1^*(\underline{s}_2)$
$\phi_2^*(\underline{s}_2)$
$\underline{s}_2$
V_1^*
V_2^*
V_1
V_2
V_3
$\underline{s}_1^*$
$\underline{s}_2^*$
$\phi_1(\underline{s}_1^*)$
$\phi_2(\underline{s}_1^*)$
$\phi_2(\underline{s}_2^*)$
$\phi_3(\underline{s}_2^*)$

Figure 4.2: Schematic of projection between meshes with misaligned nodes. Projection between meshes with misaligned nodes can produce spurious diffusion in the data. Illustrated is a 1D example in which the top row of the image represents an element with nodes $1*$ and $2*$ with membrane potential V_1* and V_2* respectively, and the bottom row represents two adjoining elements with nodes 1, 2, and 3 with membrane potential V_1, V_2, and V_3 at those nodes respectively. The elements occupy the same physical space but represent different domain discretisations. First, the membrane potential in the bottom elements is interpolated to the locations of nodes $1*$ and $2*$, these values are then re-interpolated to the location of node 2. In doing so the value at node 2 has potentially been polluted with data from nodes 1 and 3.

solution is sought, such that the difference between the local interpolated solution and the external interpolated solution is zero. This projection of the membrane potential is implemented in class wrappers, one for cell elements and one for diffusion elements.

Projection could be implemented by adding an additional projected membrane potential DOF to the elements. For the wrapper of cell elements, this is the chosen method, since the base class for the wrapper is set to the *NoDiffusionEquations* class which contains no DOFs. As a result, only one DOF is required, which is the value of membrane potential projected from the external diffusion mesh. It has the residual function

$$\mathcal{R}_k = -\int_{\mathrm{e}} \left(V^{\mathrm{Proj}}(\underline{s}) - V^{\mathrm{Ext}}(\underline{s})\right) \phi_k(\underline{s}) \, \mathrm{d}V \tag{4.13}$$

and Jacobian

$$\mathcal{J}_{kl} = -\int_{\mathrm{e}} \phi_k(\underline{s})\phi_l(\underline{s}) \, \mathrm{d}V \tag{4.14}$$

where $V^{\mathrm{Proj}}(\underline{s})$ is the projected membrane potential in the cell mesh which is to be solved for, and $V^{\mathrm{Ext}}(\underline{s})$ is the value of the membrane potential in the external mesh.

Since the projection is only performed when reassigning initial conditions at the start of each time step, this additional DOF is an unnecessary inclusion for the diffusion elements. Instead, the wrapper class provides alternative versions of *fill_in_contribution_to_residuals* and *fill_in_contribution_to_jacobian.* These alternatives, instead of solving for the membrane potential due to the base class diffusion model, solve for the value of the membrane potential as projected from the external cell mesh. A Boolean switch changes which residual and Jacobian fill-in is used. This removes the requirements of extra storage of additional DOFs and reduces computational overhead, since DOFs do not need to be pinned and unpinned during projection. It has the residual function

$$\mathcal{R}_k = -\int_{\mathrm{e}} \left(V(\underline{s}) - V^{\mathrm{Ext}}(\underline{s})\right) \phi_k(\underline{s})\,\mathrm{d}V = 0 \tag{4.15}$$

and Jacobian

$$\mathcal{J}_{kl} = -\int_{\mathrm{e}} \phi_k(\underline{s})\phi_l(\underline{s})\,\mathrm{d}V. \tag{4.16}$$

Both 4.13 and 4.15 are linear in the DOFs and so the Newton method as applied to the resultant linear systems will converge in a single iteration.

For the Bidomain equations solved in a multi-domain discretisation, projection of the membrane potential to the Bidomain mesh from the cell mesh and solution of the extracellular charge (which must be solved after reassignment of initial conditions for membrane potential) can be performed at the same time. In the same method as described above, the same wrapper class as for the Monodomain equation, the Bidomain finite elements are given an alternative residual and Jacobian fill-in procedure to perform projection from the cell mesh for the membrane potential. By leaving the equations for extracellular charge unchanged, this results in solving for an extracellular charge which corresponds to the reassigned initial conditions for the membrane potential.

The procedure for performing projection from the diffusion to the cell mesh would be as follows:

1. Pin all problem DOFs except for the projected membrane potential DOFs in the cell mesh.

2. Newtonian solve.

3. Restore pinned status of all problem DOFs and pin projected membrane potential DOFs.

The procedure for projection from the cell mesh to the diffusion mesh is as follows:

1. Pin all problem DOFs except for the DOFs of the diffusion elements.

2. Switch boolean in diffusion element such that residual and Jacobian fill in membrane potential entries for projected values.

3. Newtonian solve.

4. Switch Boolean in diffusion elements such that the normal diffusion model equations are solved.

5. Restore pinned status of all DOFs.

The above methodology also works for refinable diffusion meshes.

Situations where the diffusion mesh is refineable and the cell mesh is not could also be implemented by assigning cells to the nodes of macro elements in the diffusion mesh. The solution would then be interpolated from cells to the hanging nodes at the start of an operator splitting step. However, this method would require a partial re-implementation of the Fully Partitioned wrapper class for refinable elements. Such an implementation however would potentially be more efficient, since the projection step would not require a Newton solve of its own. Although the multi-domain implementation is less restrictive and allows for a much broader variety of multiple discretisation options without the need for additional specific versions to be written.

In future, a refineable version of *FullyPartitionedCellEquations* could be implemented which allows for cells to be built at the nodes at only the nodes of the macro elements. For a given element within the mesh, interpolation of information from the cells in the mesh will then be taken from that element's macro element instead. Similarly, the projection of data from and to external meshes can be performed in a similar fashion. Refinement and un-refinement of the mesh can then be performed without altering the number or arrangement of cells within the mesh since the mesh cannot be unrefined in such a way that cells are removed. This implementation however requires refineable versions of the *FullyPartitionedCellEquations* class and will likely require modifications to the *CellMeshBase* class so is left as a potential objective for future work.

4.4.5 Anisotropic solid mechanics

Anisotropy introduces new information which must be assimilated into the solid mechanics computation. For simple, prescribed preferential vectors and active stress/strain this could be achieved by passing relevant function pointers to the constitutive law or strain energy function. However, for physiological cell models and preferential vectors taken from anatomical scan data, a more robust and adaptable method for ingesting this information is required. This method is based on re-implementing the existing Oomph-lib isotropic solid mechanics elements and adding new member functions which allow for data to be sent to and from the *FullyPartitionedCellEquations* class. Due to this approach, driver codes which use the resultant *AnisotropicPVD-Equations* derived elements are the same as those which use the isotropic counterpart elements.

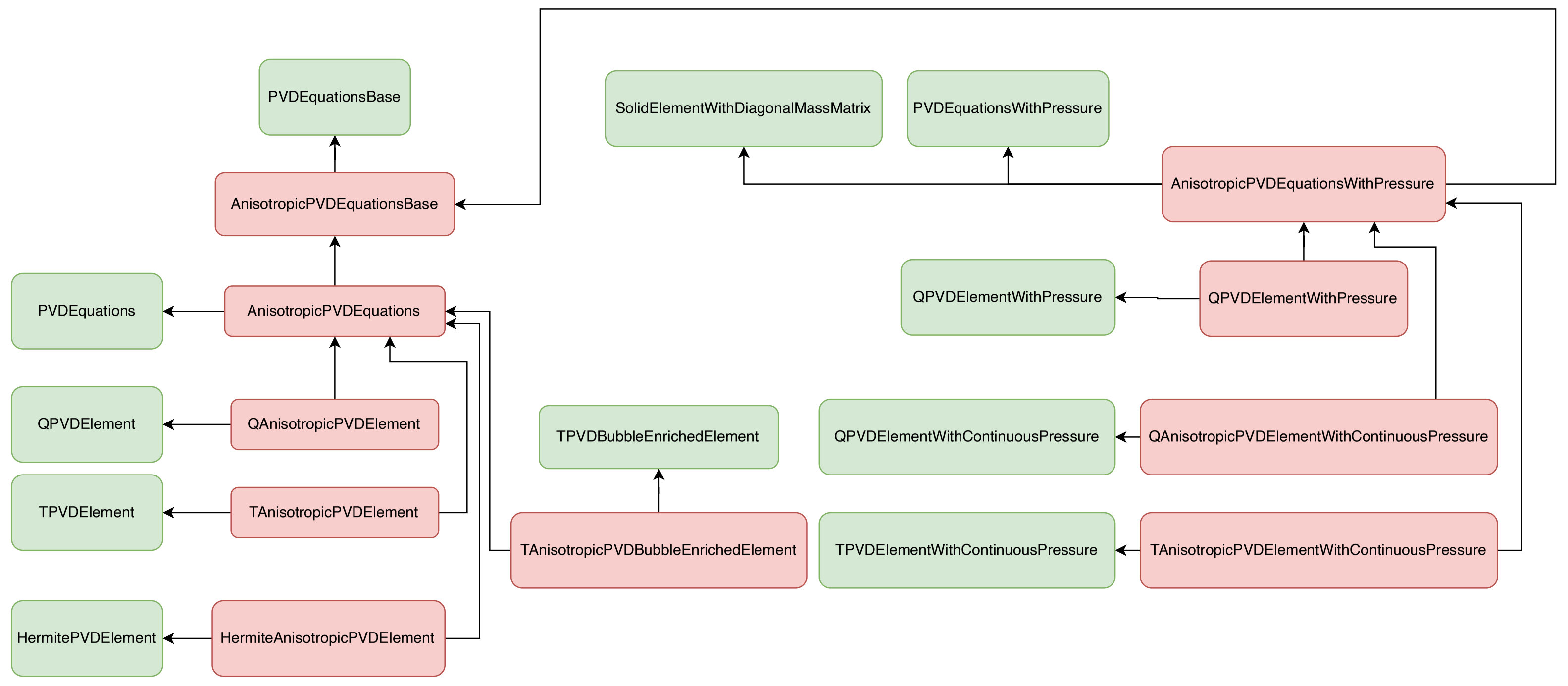

PVDEquationsBase
AnisotropicPVDEquationsBase
PVDEquations
AnisotropicPVDEquations
QPVDElement
QAnisotropicPVDElement
TPVDElement
TAnisotropicPVDElement
HermitePVDElement
HermiteAnisotropicPVDElement
SolidElementWithDiagonalMassMatrix
PVDEquationsWithPressure
AnisotropicPVDEquationsWithPressure
QPVDElementWithPressure
QPVDElementWithPressure
TPVDBubbleEnrichedElement
TAnisotropicPVDBubbleEnrichedElement
QPVDElementWithContinuousPressure
QAnisotropicPVDElementWithContinuousPressure
TPVDElementWithContinuousPressure
TAnisotropicPVDElementWithContinuousPressure

Figure 4.3: Schematic of the overview of the inheritance structure of the anisotropic solid mechanics classes and how they relate to the isotropic solid classes. Oomph-lib classes are displayed in green and classes added as part of this work are shown in red. Arrows point from a derived class to its base class.

Inheriting from the PVD elements

The newly implemented anisotropic solid mechanics elements are built upon the Oomph-lib non-linear isotropic solid mechanics elements. They inherit from the already existing classes and declare and define new methods and functions for implementing anisotropic features.

Anisotropic solid mechanics elements are implemented as a wrapper of the existing hyperelastic isotropic solid mechanics elements with several virtual functions overridden in order to implement the anisotropic components of the hyperelasticity models. Additional functions are provided which allow for data to be passed to the anisotropic solid elements, either through function pointers or through interfacing with external elements in multi-physics simulations.

Since additional data is required by many member functions, several functions with the same names as existing isotropic counterparts are re-implemented to allow for the addition of anisotropy by including additional function arguments.

Two versions of the function *get_stress* exist in the isotropic implementation

```
void get_stress(const Vector<double> &s,
                DenseMatrix<double> &sigma);
```

and

```
void get_stress(const DenseMatrix<double> &g,
              const DenseMatrix<double> &G,
              DenseMatrix<double> &sigma);
```

The first is called by output functions to calculate the stress at local coordinate s. The latter is called during the compressible residual and Jacobian calculation to calculate stress as a function of the components of the undeformed and deformed metric tensor components g and G. The first calculates the deformed and undeformed metric tensors and then calls the latter.

Alternative anisotropic versions of these functions were implemented. The first has no changes to arguments, however, the latter has additional arguments which represent the preferential vectors and the actives stress/strain

```
inline void get_stress(const DenseMatrix<double> &g,
            const DenseMatrix<double> &G,
            const Vector<Vector<double>>& A,
            const Vector<double> &V,
            DenseMatrix<double> &sigma);
```

where A is a vector, the i^{th} element of which is the i^{th} preferential vector, and V is the vector of active stress or active strain.

For incompressible solids, the derivative of the stress with respect to the contravariant deformed metric tensor is required and is calculated by the following function

```
inline void get_d_stress_dG_upper(const DenseMatrix<double> &g,
              const DenseMatrix<double> &G,
              const DenseMatrix<double> &sigma,
              RankFourTensor<double> &d_sigma_dG)
```

For anisotropic incompressible solids, this calculation requires the preferential vectors and active stress/strain and so an updated version of this function is implemented as the following

```
inline void get_d_stress_dG_upper(const DenseMatrix<double> &g,
              const DenseMatrix<double> &G,
              const Vector<Vector<double>>& A,
              const Vector<double> &V,
              const DenseMatrix<double> &sigma,
              RankFourTensor<double> &d_sigma_dG)
```

Similarly, there is a version of this function for nearly incompressible isotropic solids

```
inline void get_d_stress_dG_upper(const DenseMatrix<double> &g,
            const DenseMatrix<double> &G,
            const DenseMatrix<double> &sigma,
            const double &gen_dil,
            const double &inv_kappa,
            const double &interpolated_solid_p,
            RankFourTensor<double> &d_sigma_dG,
            DenseMatrix<double> &d_gen_dil_dG)
```

and so an implementation of this function which adds preferential vectors and active stress/strain is added

```
inline void get_d_stress_dG_upper(const DenseMatrix<double> &g,
            const DenseMatrix<double> &G,
            const Vector<Vector<double>>& A,
            const Vector<double> &V,
            const DenseMatrix<double> &sigma,
            const double &gen_dil,
            const double &inv_kappa,
            const double &interpolated_solid_p,
            RankFourTensor<double> &d_sigma_dG,
            DenseMatrix<double> &d_gen_dil_dG)
```

These functions all serve to pass the preferential vectors and active stress or strain to the anisotropic constitutive law which then calculates the required values according to the constitutive model it represents.

The residual and Jacobian fill-in functions are re-implemented, from the versions in *PVDEquations* and *PVDEquationsWithPressure*. The new implementations are functionally identical to the originals, except that when *get_stress* or *get_d_stress_dG_upper* is called, the alternative anisotropic version is called instead. Additionally, preferential vectors and active strain/stress are also calculated as needed.

Similarly, output functions are re-implemented so that stress can be properly calculated from the anisotropic formulation in the constitutive law.

Member functions which get the values of preferential vectors and active stress/strain are implemented. These take the form of source functions used throughout Oomph-lib and are given by

```
virtual inline void preferential_vectors(const unsigned& ipt,
            const Vector<double> &s,
            const Vector<double>& xi,
            Vector<Vector<double>>& A)
```

and

```
virtual inline void driving_strain(const unsigned& ipt,
            const Vector<double>& s,
            const Vector<double>& xi,
            Vector<double>& V)
```

respectively.

These call function pointers of the form

```
typedef void (*PreferentialVectorsFctPt)(const unsigned& ipt,
            const Vector<double>& s,
            const Vector<double>& x,
            Vector<Vector<double>>& A);
```

and

```
typedef void (*DrivingStrainFctPt)(const unsigned& ipt,
          const Vector<double>& s,
          const Vector<double>& x,
          Vector<double>& V);
```

respectively which must be defined and assigned by the user within a driver code as per source functions in other Oomph-lib finite elements.

When the preferential vectors and active stress/strain must be taken from a more complicated source than these simple source functions can provide, the virtual functions preferential_vectors and driving_strain are overridden so that they can take their values from the required location instead.

Despite these re-implementations, the code reuse achieved through implementing anisotropic solid mechanics as a wrapper of the existing hyperelastic solid elements results in comparatively little new code when compared to the size of the original isotropic hyperelastic solid mechanics files. This is in line with the code reuse ethos of Oomph-lib.

Methods for acquiring preferential vector and active strain or stress information are added to the anisotropic solid mechanics elements and standalone anisotropic versions of the constitutive law and strain energy classes are implemented in order to allow for information on anisotropy to be passed to them.

Anisotropic Constitutive laws

Anisotropic constitutive laws and anisotropic strain energy function classes are implemented from scratch and do not inherit from their existing isotopic counterparts. This is because the classes themselves are comparatively simple and already contain very few functions, the interfaces of all of which have to be modified in order to communicate the necessary information regarding anisotropy to the function.

The names of the member functions of ConstitutiveLaw which are called by the solid mechanics elements, namely *calculate_second_piola_kirchhoff_stress*, *requires_incompressibility_constraint* remain unchanged in the *Anisotropic-ConstitutiveLaw* classes. However, additional arguments A and V are added to *calculate_second_piola_kirchhoff_stress* which represent the anisotropic orientations within the solid and the active strain/stress generated by the cardiomyocyte model in each of those directions respectively. A is a vector of vectors of doubles, each of which contains the orientations which define the preferred vectors of the solid, and V is a vector of doubles which contains the active stress/strain in each of those directions. For brevity and simplicity, and since models of cardiomyocyte contraction can be defined to compute the force generated due to contraction or compute the sarcomere shortening ratio, V is allowed to represent either the active strain or active stress components. It is then required that a suitable corresponding anisotropic constitutive law which defines either the active stress or active strain formulation is used. Since no distinction in terms of the nature of the information which is passed from the cardiomyocyte model to the constitutive law can be made, there are no checks which ensure that a constitutive law which requires the active stress/strain is indeed receiving the stress/strain. It is left to the user to ensure that proper combinations of constitutive laws and cardiomyocyte models are used.

Indeed a function could be written that checks if the active stress/strain information required by the constitutive law matches that provided by the cell model. However, such a function has not been implemented as of yet.

Implementation of anisotropic constitutive laws which are derived from strain energy functions is similar to that of isotropic constitutive laws. The class *AnisotropicStrainEnergyFunctionConstitutiveLaw* computes the components of the second Piola-Kirchhoff stress tensor as well as other quantities required by the anisotropic solid elements through calls to functions contained within a pointer to the *AnisotropicStrainEnergyFunction* derived class used. *AnisotropicStrainEnergyFunction* contains several virtual functions which compute the strain energy in terms of the strain tensor or strain invariants, the derivative of the strain energy with respect to the strain, and derivatives of the strain energy with respect to the strain invariants. By default, derivatives are implemented using generic finite differencing schemes, although these can be overridden when an analytic form exists in order to improve performance.

Several constitutive laws commonly used within cardiac modelling and in the validation of numerical schemes have been implemented and are shown in figure 4.4. Both the active strain and active stress decompositions of these laws have been implemented.

The existing isotropic Mooney-Rivlin constitutive law is upgraded to include terms accounting for active stress/strain which arise from the contraction of cardiomyocytes, both the active stress and active strain decompositions are applied in separate classes.

Secondly, a constitutive law which contains a simple exponential term is implemented. Again, this model is implemented with both the active stress and active strain decompositions. Due to its simplicity, it is commonly used in validation rather than physiological simulation. Due to its prevalence, it is included in the library.

The simplest phenomenological constitutive law which is implemented is due to the Pole-Zero strain energy function. This model, introduced by Nash and Hunter 2000, seeks to account for observed strain limiting in the three principal axes of anisotropy within the myocardium. Both the active stress and active strain decompositions are implemented.

The Holzapfel Ogden constitutive law is widely used in studying structural deformation of the myocardium. Both the active stress and active strain decompositions are implemented. For the active stress decomposition, all of the incompressible, nearly incompressible, and compressible forms are implemented, whereas for the active strain decomposition only the incompressible form is implemented.

Communicating active stress/strain and cell alignment from the cell models

A multiphysics version of *AnisotropicSolidElement* is implemented

```
template<class SOLID_ELEMENT, class EXT_CELL_SOLVER_ELEMENT>
class AnisotropicSolidElementWithExternalCellElement :
    public virtual SOLID_ELEMENT,
    public virtual ElementWithExternalElement
{
...
};
```

This multiphysics version has an external *FullyPartitionedCellEquations* derived class which is defined by the second template argument *EXT_ CELL_ SOLVER_ ELEMENT*.

For instances when the active stress/strain must be taken from a cell model rather than a source function, the function *driving_ strain* must be overridden to take the active strain/stress from the cell model using the element with external element procedures.

There is no distinction made between active stress and active strain within the function arguments of either the anisotropic finite elements or the constitutive law class. Both active stress and active strain are passed around as the same variable since only one of the two is ever used at any time in modelling. It is the job of the user to ensure that the output of the cell model (either active stress or active strain) matches what the constitutive law expects to receive.

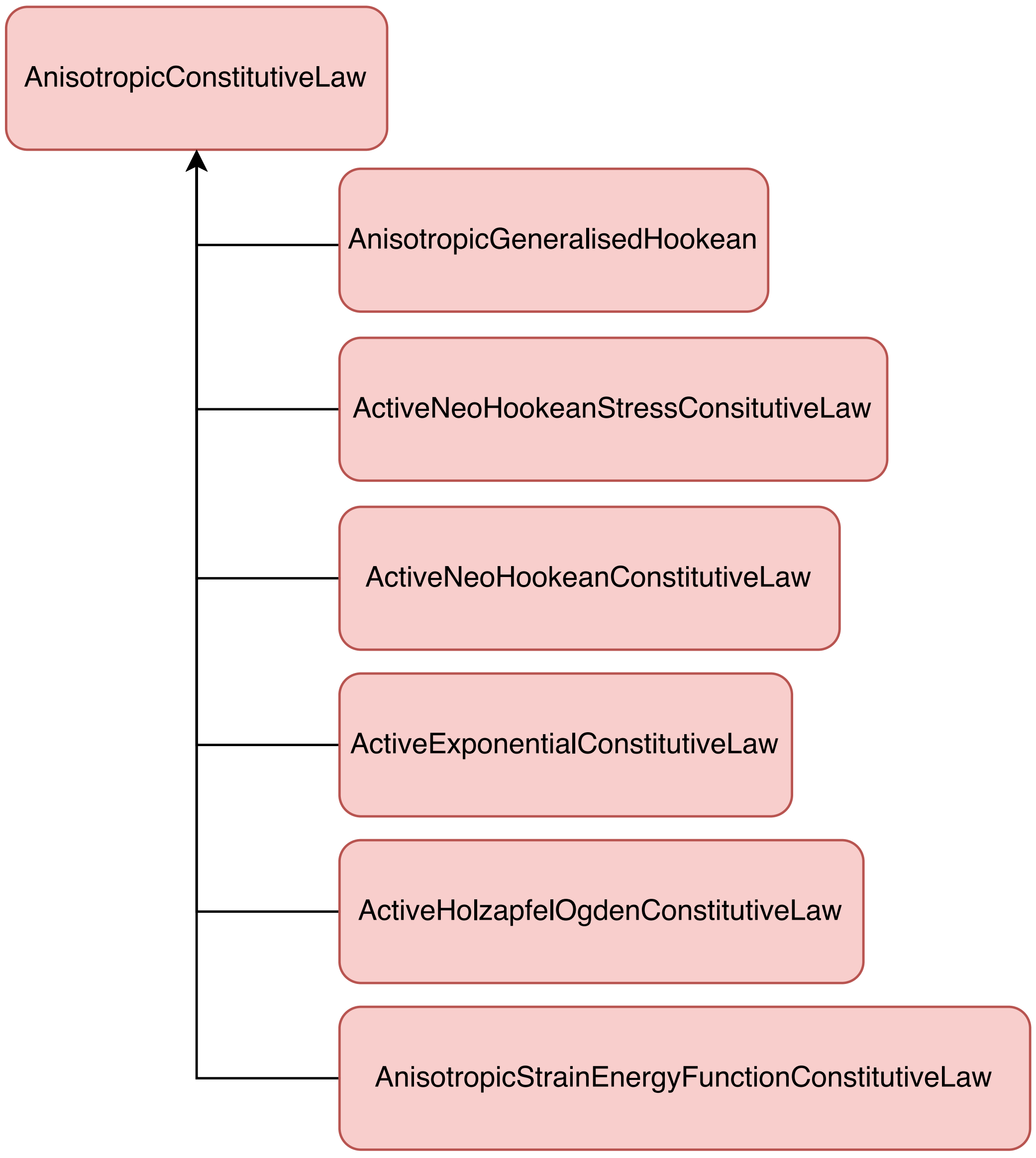

AnisotropicConstitutiveLaw
AnisotropicGeneralisedHookean
ActiveNeoHookeanStressConsitutiveLaw
ActiveNeoHookeanConstitutiveLaw
ActiveExponentialConstitutiveLaw
ActiveHolzapfelOgdenConstitutiveLaw
AnisotropicStrainEnergyFunctionConstitutiveLaw

Figure 4.4: Schematic of the inheritance structure of the anisotropic constitutive law classes.

Refineable anisotropic solid mechanics elements

A refineable version of the anisotropic solid mechanics elements has not yet been implemented. However, it would essentially follow the implementation of the Oomph-lib isotropic solid mechanics elements. The only change necessary would be to pass the additional function pointers,

```
    PreferentialVectorsFctPt Preferential_Vectors_fct_pt;
    DrivingStrainFctPt Driving_strain_fct_pt;
```

to the child elements.

4.4.6 Generating unstructured meshes from anatomically detailed geometries

Within Oomph-lib, generating ideal geometries with structured meshes is relatively easy. This process is detailed throughout the Oomph-lib documentation. However, generating geometries with unstructured meshes can be more challenging. This requires the Oomph-lib triangle mesh functions to be called wherein an external library Triangle or TetGen is invoked. Furthermore, mesh adaptation and large displacements require re-meshing and projection of any fields from the old mesh to the new one.

Defining more intricate geometries with curvilinear boundaries for the generation of unstructured meshes is similarly tedious to the process for simple geometries. Generation of unstructured meshes with non-ideal geometries is more difficult. The use of complex, unstructured, and non-idealised biophysically detailed meshes is essential for the study of cardiac electrophysiology. Such meshes are often generated from MR or CT scan data. Methods must therefore be developed to permit their use within the Oomph-lib computing environment if the additions to Oomph-lib produced as part of this thesis are to be used for biophysically detailed simulations. The methods and the resultant pipeline for the use of such meshes are introduced and detailed.

Since there is no method in place within Oomph-lib for handling the file types often used for bio-physically detailed meshes, a new method must be implemented which allows for this. Such a method has been implemented. Initially, the geometries are loaded into ParaView. The element and node data is then exported by viewing the geometry in a spreadsheet view and exporting when viewing the attributes "point data" and "cell data". It is these two files which are used to generate the unstructured mesh within Oomph-lib. Since the files used by the method are generated by Paraview, this method can potentially be used for any mesh which can be loaded into paraview.

The node file contains the global coordinate of the nodes as well as any other node data packaged with the geometry. The node data is read and the relevant information is saved. In addition to the global nodal coordinates, this information could include local coordinates, cell types, or fibre orientations.

Next, the element file is loaded. This file contains the element numbers as well as the global identifying numbers of each node contained within each element. Additional information could be contained within this file if cell type or fibre orientation data is defined per element instead of per node.

For each node, the elements which contain that node are recorded. This is essential for correctly constructing the elements as well as constructing boundary nodes.

Next, the list of unique facets is generated. The files generated by paraview do not contain boundary information, this is an issue since correct construction of an Oomph-lib mesh requires explicitly knowing which nodes are on the boundaries of the domain. This information can be generated from the element data. Since the elements are tetrahedral, every combination of three nodes within each element represents a unique facet of that element. In order to determine which of these facets are on the domain boundary the complete list of facets of all the elements is interrogated to look at facets which are not repeated. Non-repeated facets must be on a domain boundary since if a facet is on the inside of the domain it must also exist within another element.

In order to interrogate the element facets efficiently, a hash function is constructed which converts each facet into a unique identifying integer. These identifying integers are then used to generate an unordered map. The facets are looped over, their identifying integer is generated, and the corresponding element of the unordered map is incremented. After this procedure, if an element in the unordered map returns a value of one then the corresponding facet only appears once in the mesh and so must be a boundary facet.

The hash function used requires that the global identifying numbers of the nodes are ordered from smallest to largest, otherwise, the global node numbers could be rearranged and as such produce three identifying integers for each element. Once the node numbers are ordered in ascending order, a unique identifying number is constructed by appending the digits of the node numbers to form one long integer. In order to ensure that the hash function produces unique identifying numbers the node numbers have zeros added to the end until they are of a set length in terms of digits. This set length should be large enough to accommodate the number of nodes in the mesh, for example, if there are 1×10^5 nodes, the node numbers should be buffered with zeros until they are 6 digits long. The resultant integer produced for each facet is guaranteed to be unique for a unique combination of three nodes in the facet.

In situations when an electrophysiological mesh is generated from a biophysically detailed geometry, it is possible that elements are created with a local-to-global coordinate mapping with an inverted Jacobian. To handle such cases, the function *check_for_and_deal_with_inverted_jacobian* is provided which checks for inverted Jacobian and appropriately swaps node numbers within the element if needed.

A drawback of this mesh generation procedure is that only tetrahedral elements with 2 nodes per edge can be generated, as such only linear basis functions can be applied to the finite elements. Although it could be possible to add more nodes to the elements in between the corner nodes, this has not been implemented. As such higher-order spatial approximations of the solution are not possible on meshes generated with this method.

Good quality and usefully labelled meshes such as those provided in [127] are very useful when generating meshes from anatomical geometries. This large data set containing a cohort of 24 four-chamber heart meshes derived from CT scans is provided along with a natural local coordinate system describing the left ventricle. The coordinates are the fractional distance from the endocardial to epicardial surfaces, the fractional distance from the apex to the base, and the angle clockwise from the septum around the ventricle wall. The use of such coordinates permits easy identification of nodes and element facets which are on the endocardial and epicardial surfaces, as well as being helpful in identifying the ring of nodes and elements surrounding the aortic and mitral valves - which is essential for a full description of the blood pool within the vessel. Such surface identification is required when building meshes in Oomph-lib since the application of boundary conditions assumes knowledge of the surfaces of the mesh. Additionally, should only a segment of the left ventricle be required for localised simulations or for toy problems during development, it is possible to only include nodes and elements within a specified subdomain. Nodes which are on the endocardial and epicardial surfaces can be identified by their transmural coordinate. Similarly, nodes at the ventricle base or apex can be identified. Suitable boundary conditions or applied tractions or stimuli for the electrical and solid simulations can then be applied to these regions individually. This virtual cohort provides cell fibre, sheet, and sheet-normal alignments for the elements in the left ventricle which can then be interpolated to the cells at the nodes. These alignments and cell types within the left ventricle were generated by the authors by applying rule-based methods which are functions of the local coordinates: transmural and apicobasal ratios, and the azimuthal angle[128]. Such rules are extensively discussed in the literature and have been shown to closely reflect the physiological arrangement of the ventricles. However, for patient-specific modelling, scan data of fibre and sheet alignments would likely produce more reliable results. Such scan data is not available in the cohort dataset used in the development of this mesh generation but can be determined through diffusion MR imaging[129].

Chapter 5

Validation

In order to determine that the implemented elements correctly simulate their governing equations, several validation tests were undertaken. For the monodomain equation, validation in 2D was performed for homogeneous conductivity and for source functions which provide known analytic solutions. Validation was performed on an undeformed mesh and so the ALE formulation and any pullback of components of the equations are ignored. Capacitance is also assumed to be unity since, given the assumption of homogeneous conductivity, the capacitance can be absorbed into the conductivity tensor. A general method for the validation of numerical models for physical systems is that the solution converges to a finite value as the spatial and temporal resolutions become infinitely fine. To test for this, in all validation cases an additional test was performed wherein the spatial and temporal resolutions are refined and the resultant solutions are inspected for convergence.

For certain forms of the stimulus function, analytic solutions of the monodomain equation exist. For this validation, source functions of the form $I_{\text{stim}} = \alpha(\underline{x}) + f(t)$ were chosen. The resultant equation was solved using the monodomain element implementation in the library with a source function to represent the source term.

As a further validation, a cell model was employed which is coupled to the monodomain elements with operator splitting techniques. The two methods were then compared to the analytic solution, and as such the error of each method was determined. For clarity, the solution which uses Oomph-lib style finite elements with a source function and Oomph-lib time-stepping is referred to as the Monolithic solution, whereas the solution which involves the separation of the source terms from the finite elements through the use of operator splitting is referred to as the Partitioned solution. The Partitioned solution was computed for the operator splitting methods Lie-Trotter, Strang-Splitting, and AM-32.

Additionally, the partitioned solution was computed with the adaptive operator splitting methods: Palindromic Lie-Trotter, Strang-Milne, and Symmetric Milne-32. For these methods, an adaptive time-stepping procedure was applied to the solution of the diffusion equation.

The solution to the Monodomain equation was computed for several Oomph-lib times-steppers: the trapezoidal time-stepper, BDF$\langle 1\rangle$, BDF$\langle 2\rangle$, BDF$\langle 4\rangle$, and a hard-coded Crank-Nicolson implementation of the Monodomain model. Results demonstrate the assertion that time-stepping methods with more than one history value are incompatible with operator-splitting methods. The global error was observed to increase as the order of the time-stepper was increased where an increase in time-stepper order corresponds to an increase in the number of history values. Additionally, these validation results indicate that the Oomph-lib trapezoidal time-stepper may not be applicable to operator splitting either since the results given by it disagree with those of an explicitly coded Crank-Nicolson monodomain element.

In the application of adaptive operator splitting methods, only $\langle 1\rangle$ was used for solving the diffusion equations. This is because only the BDF$\langle 1\rangle$ was identified as being suitable for use with operator splitting and because the explicitly coded Crank-Nicolson implementation of the monodomain equation does not provide a measure of solution error.

For both the Partitioned solutions, for consistency, the maximum time-step allowed for the solution to the non-linear source term was set to $\Delta t_{\max} = 0.01$ and solved in the method set out in section 3.5.3.

5.1 Monolithic and Non-Adaptive operator splitting

The non-dimensionalised monodomain equation including the source function is of the form

$$\frac{\partial V}{\partial t} + \alpha(x,y) + f(t) = \sigma_x^2 \frac{\partial^2 V}{\partial x^2} + \sigma_y^2 \frac{\partial^2 V}{\partial y^2}$$

with the insulating boundary conditions $\nabla V \cdot \underline{n} = 0$ on $(x,y) = (0,y)$, $(x,y) = (L,y)$, $(x,y) = (x,0)$, and $(x,y) = (x,L)$ where $\underline{n}$ is the outward pointing normal from the domain Ω on the boundary. Through the separation of variables, the analytic solution is given by

$$\begin{aligned} V = \sum_{j=0}^{\infty}\sum_{i=0}^{\infty} A_{ij} \cos\left(\frac{i\pi x}{L}\right) \cos\left(\frac{j\pi y}{L}\right) \exp\left(-\frac{\pi^2}{L^2}\left(i^2\sigma_x^2 + j^2\sigma_y^2\right) t\right) \\ + \gamma(x,y) + \int_0^t -f(\tau)\,\mathrm{d}\tau \end{aligned} \tag{5.1}$$

Where it is assumed that $\alpha(x, y)$ takes the form

$$\alpha(x, y) = \sum_{j=0}^{\infty} \sum_{i=0}^{\infty} \Lambda_{ij} \cos\left(\frac{i\pi x}{L}\right) \cos\left(\frac{j\pi y}{L}\right)$$

for given Λ_{ij}. $\gamma(x, y)$ is therefore given by

$$\gamma(x, y) = -\frac{L^2}{\pi^2} \sum_{j=0}^{\infty} \sum_{i=0}^{\infty} \frac{\Lambda_{ij}}{\sigma_x^2 i^2 + \sigma_y^2 j^2} \cos\left(\frac{i\pi x}{L}\right) \cos\left(\frac{j\pi y}{L}\right).$$

The initial conditions A_{ij} are then given by

$$A_{ij} = \frac{L^2 \Lambda_{ij}}{\pi^2(\sigma_x^2 i^2 + \sigma_y^2 j^2)} + \frac{L^2}{4} \int_0^L \int_0^L V_0(x) \cos\left(\frac{i\pi x}{L}\right) \cos\left(\frac{j\pi y}{L}\right) \mathrm{d}x \, \mathrm{d}y.$$

Weights in the source term are given in table 5.1 Weights in the initial

	j				
i	0	1	2	3	4
0	0.0	0.0	0.0	0.0	0.0
1	0.0	-0.05	0.05	0.01	-0.02
2	0.0	0.28	0.09	0.28	0.8
3	0.0	0.03	-0.03	-0.03	-0.06
4	0.0	-0.08	0.3	0.002	0.08

Table 5.1: Weights of spatially dependent source term, Λ_{ij},

conditions were given in table 5.2. Conductivities were set to $\sigma_x = 1.0$ and $\sigma_y = 0.24$.

$f(t)$ is the same as for the 1D validation test and so is the corresponding term in the analytic solution with the exception of a scalar change in magnitude of 0.05.

$$f(t) = -\Gamma \sin(\omega t) \exp(-\lambda t)$$

	j				
i	0	1	2	3	4
0	0.6	0.210	-0.12	-0.1	0.0
1	0.3460	-0.9	0.4	0.001	-0.12
2	0.1237	0.37	0.1	0.0	0.8
3	0.26	0.98	0.01	2.1	-0.7
4	0.1	0.01	0.4	-0.6	0.416

Table 5.2: Weights of the initial conditions, A_{ij},

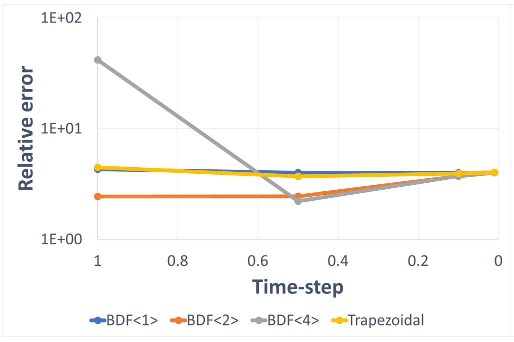
is chosen with $\omega, \lambda, \Gamma \in \mathbb{R}$. This results in the corresponding term in the exact solution

$$\int_0^t f(\tau)\,\mathrm{d}\tau = \frac{\Gamma}{\omega^2+\lambda^2}\left(\exp^{(-\lambda t)}\left(\omega\cos(\omega t)+\lambda\sin(\omega t)\right)-\omega\right).$$

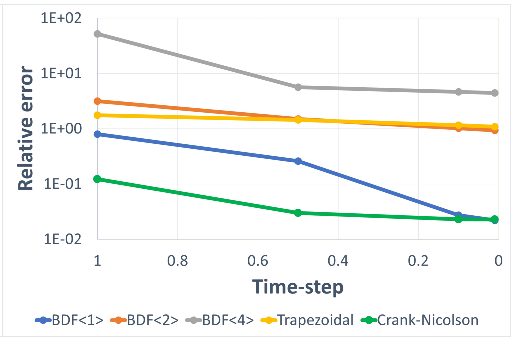
The values are set to $\omega = 0.1$, $\lambda = 0.5$, and $\Gamma = \frac{1}{20}$.

The error, as in the previous validation tests, is determined as the difference between the calculated solution and the analytic solution. The solution was computed for both the Monolithic and Partitioned methods for several time-steppers and operator-splitting techniques. The error, as in the previous 1D validation test, is determined as the difference between the calculated solution and the analytic solution. The solution was computed for both the Monolithic and Partitioned methods for several time-steppers and operator-splitting techniques. The results for the monolithic solution are presented in figures 5.1 (a) and 5.2 (a).

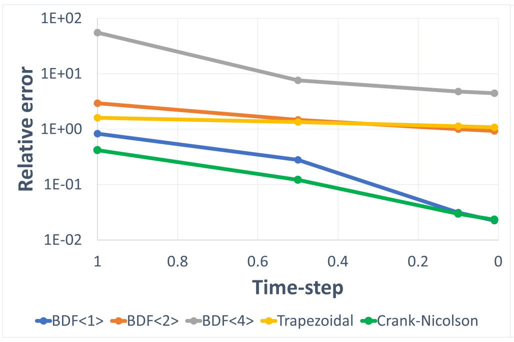
Figure 5.1 (a) shows that for the monolithic solution, the error does not decrease monotonically for large time-step values and for the BDF$\langle 2\rangle$ and BDF$\langle 4\rangle$ methods. This is because large time steps render the solution unstable for higher-order time-stepping methods. For the trapezoidal and BDF$\langle 1\rangle$ methods, the error is of lower order so larger time-steps do not produce as large an error. For Strang-splitting (Figure 5.1 (b)), Lie-Trotter (Figure 5.1 (c)), and AM32 (Figure 5.1 (d)), the error from applying Crank-Nicolson and BDF⟨1⟩ converges monotonically. However, the error from applying the Trapezoidal, BDF⟨2⟩, and BDF⟨4⟩ methods does not decrease monotonically which indicates that these time-stepping methods may not be suitable for use with adaptive operator splitting.



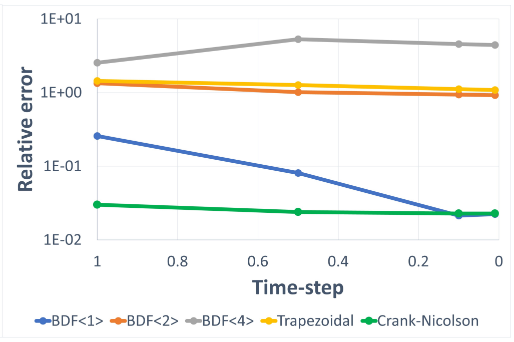
(a)



(b)



(c)



(d)

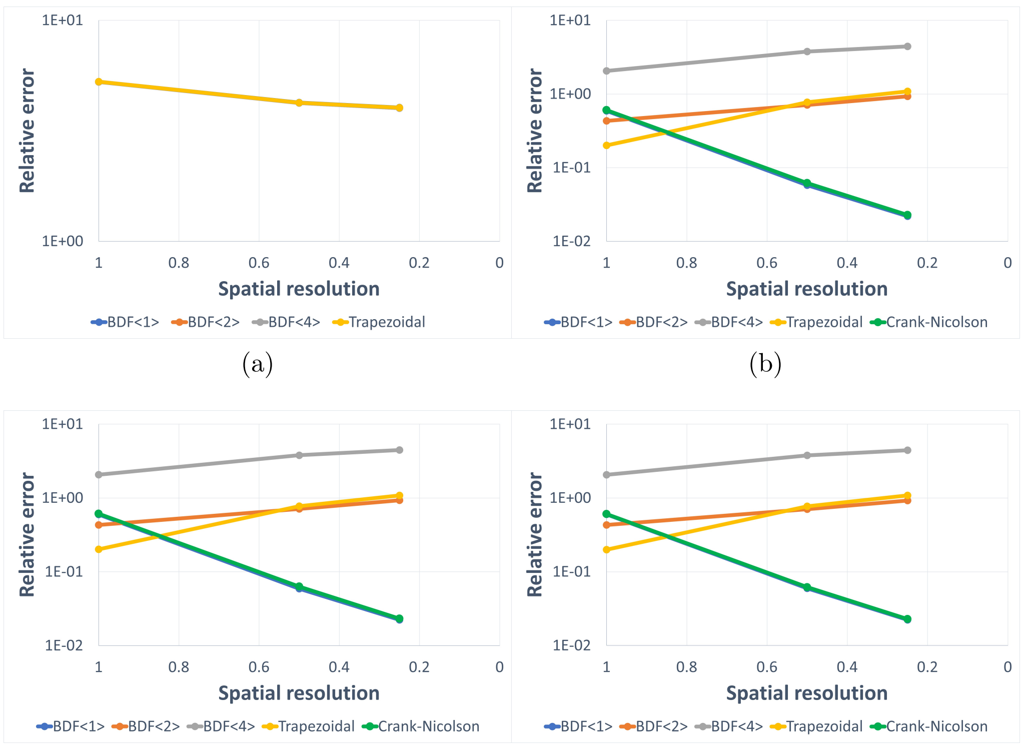
Figure 5.1: Relative error of the validation test with a fixed spatial resolution and varying time-step. Each panel presents the error for either the monolithic method or an operator-splitting method. Each method is applied with several time-stepping methods which are applied to time-stepping either the combined equation in the monolithic case, or each of the monodomain and source equations in the operator splitting cases. (a) is Monolithic, (b) is Strang-Splitting, (c) is Lie-Trotter, (d) is AM32.

Figure 5.2 (a) shows that, for the monolithic solution with a fixed time-step, the error converges monotonically as the spatial resolution is decreased as expected. However, for the operator splitting methods: panels (b), (c), and (d) only the error of the BDF$\langle 1 \rangle$ and Crank-Nicolson methods converge monotonically with decreasing spatial resolution, as was observed with decreasing time-step. The error from applying other time-stepping methods, Trapezoidal, BDF$\langle 2 \rangle$, and BDF$\langle 4 \rangle$, does not converge monotonically as spatial resolution decreases, further indicating that these time-stepping methods may be unsuitable for use with operator splitting. The results in panels (b), (c), and (d) are very similar, indicating that Strang-Splitting, Lie-Trotter, and AM32 respond similarly to changes in spatial resolution for a fixed time-step.

5.2 Adaptive operator splitting

The monodomain equation with non-linear source terms was also solved with adaptive operator splitting methods. These methods are discussed in the relevant section 3.5.3.

Adaptive operator splitting was applied to the system. Within the solution to the non-linear source equations, the maximum time step was imposed as was the case for the non-adaptive partitioned solutions in the previous section 5.1. The monodomain equation was solved with both adaptive and non-adaptive time-stepping in order to observe the effects on the results. It was demonstrated in figures 5.1 and 5.2 that BDF$\langle 2 \rangle$, BDF$\langle 4 \rangle$, and the trapezoidal time-stepper appear to be unsuitable for use with operator splitting. As such, higher-order time-steppers were not used since their use with operator splitting methods was shown to increase the local error measure. Additionally, since the explicitly coded Crank-Nicolson formulation does not



(a) (b) (c) (d)

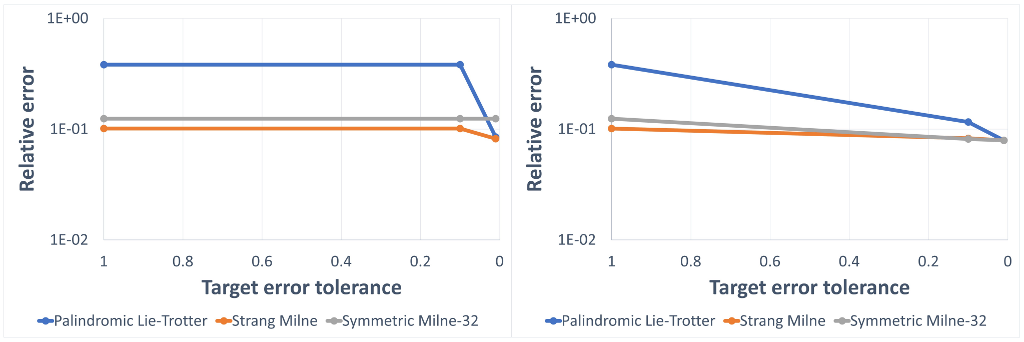
Figure 5.2: Relative error of the validation test with a fixed time-step and varying spatial resolution. Each panel presents the error for either the monolithic method or an operator-splitting method. Each method is applied with several time-stepping methods which are applied to time-stepping either the combined equation in the monolithic case, or each of the monodomain and source equations in the operator splitting cases. (a) is Monolithic, (b) is Strang-Splitting, (c) is Lie-Trotter, (d) is AM32.

provide an error measure, only BDF⟨1⟩ is used for adaptive solutions to the monodomain equation.

The error tolerance of the adaptive operator splitting and adaptive time-stepping of the diffusion equation was varied in order to demonstrate convergence. The error tolerance of the adaptive time-stepping was set to the same value as for the adaptive operator splitting error tolerance.

All initial conditions, source terms, and simulation parameters were the same as for the previous section 5.1.

Comparison of figure 5.3 (a) and figure 5.3 (b) shows that the application of adaptive time-stepping to solving the monodomain and source term equations improves the convergence of the methods as target error tolerance is decreased.



(a) (b)

Figure 5.3: Relative error in the validation simulation when adaptive operator splitting is applied. For a fixed spatial resolution, the target error tolerance in the operator splitting is varied and the time-step is allowed to change adaptively as the operator splitting algorithm progresses. Three adaptive operator splitting methods were applied: Palindromic Lie-Trotter, Strang-Milne, and Symmetric-Milne-32, and BDF⟨1⟩ was used in all cases. Panel (a) is the error when the monodomain equation and source equation were solved with a fixed time-step, and panel (b) is the error when the monodomain equation and source equation were solved with adaptive operator splitting.

Comparison of figure 5.4 (a) and figure 5.4 (b) demonstrate that applying adaptive time-stepping to solving the monodomain and source term equations has little effect on convergence as the spatial resolution is decreased.

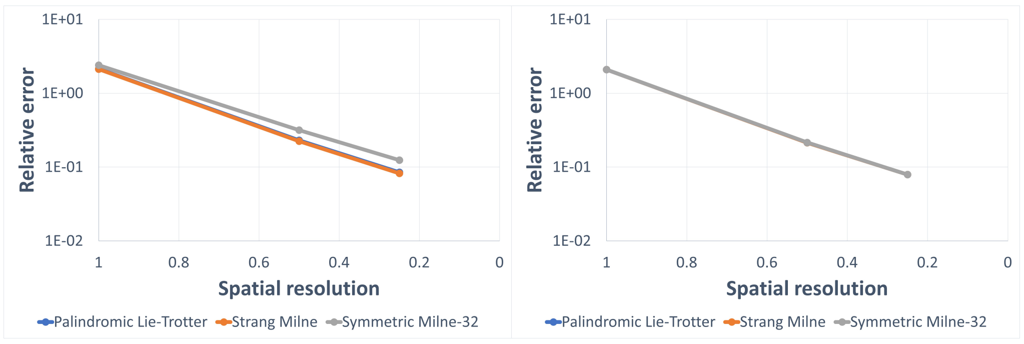


Figure 5.4: Relative error in the validation simulation when adaptive operator splitting is applied. For a fixed target error tolerance, the spatial resolution in the operator splitting is varied and the time-step is allowed to change adaptively as the operator splitting algorithm progresses. Three adaptive operator splitting methods were applied: Palindromic Lie-Trotter, Strang-Milne, and Symmetric-Milne-32, and BDF⟨1⟩ was used in all cases. Panel (a) is the error when the monodomain equation and source equation were solved with a fixed time-step, and panel (b) is the error when the monodomain equation and source equation were solved with adaptive operator splitting.

5.3 Considerations for time-stepping with operator splitting

When using operator splitting it is clear that the use of certain time-steppers to solve the monodomain equation produces larger errors than expected. Several methods are used when solving the monodomain equation. The BDF time-steppers, BDF⟨1⟩, BDF⟨2⟩, and BDF⟨4⟩ are applied. These are packaged within Oomph-lib. BDF⟨1⟩ and BDF⟨2⟩ support error estimation and adaptive time-stepping procedures. The trapezoidal time-stepper, often referred to as the Crank-Nicolson method was also used. Crank-Nicolson is a widely used time-stepping method when solving the monodomain equation. Through investigation, it appears that the trapezoidal time-stepper, as it appears within Oomph-lib, cannot be used to implement the Crank-Nicolson method when used with operator splitting. When compared to an explicitly coded version of Crank-Nicolson in which the derivatives are calculated explicitly in the residual equation, the trapezoidal time-stepper consistently produces results which are significantly different. This is in all cases where either *setup_ initial_ derivative* or *setup_ initial_ derivative* and *actions_ after_ timestep* are called for the time-stepper. This is likely an issue arising from the way in which the trapezoidal time-stepper calculates the residual function at the previous time step. This is performed by simply calculating $f(t_{n-1}) = \frac{V(t_{n-1})-V(t_{n-2})}{\Delta t}$ rather than explicitly performing the calculation, in the monodomain equation, this explicit calculation would correspond to calculating $f(t_{n-1}) = \nabla \cdot \mathbf{D}\nabla V(t_{n-1})$. Under the normal circumstances in which the trapezoidal time-stepper is designed to operate, such a simplification is mathematically sound and effectively reduces the computational load required to use the trapezoidal time-stepper. However, since this calculation is performed after the solution to the cell equations, $V^C(t_{n-1})$, is projected to the diffusion DOFs the resultant quantity $\frac{V(t_{n-1})^C-V(t_{n-2})}{\Delta t}$ is not necessarily equal to $\nabla \cdot \mathbf{D}\nabla V^C(t_{n-1})$ since $V^C(t_{n-1})$ was not determined by solving the equation $\frac{\partial V}{\partial t} = \nabla \cdot \mathbf{D}\nabla V$. This discrepancy and the unsuitability of the trapezoidal time-stepper in solving cardiac electrophysiology with operator splitting are highlighted in the results sections.

Chapter 6

Use of the numerical package

Many example codes and tutorials have been written to inform users of Oomph-lib's many features and to work as a self-study guide[121]. Similarly, several example code snippets are provided throughout this section which details the use of different features of the additions made to Oomph-lib as a part of this thesis.

6.1 Defining a point-source model

Defining point-source models is simple. The process is outlined for a version of the Fitzhugh-Nagumo model. The model needs to define how many variables there are, compute appropriate initial conditions, define how the variables are updated by a time-step, compute any additional variables required for output or required by other elements, and compute active strain/stress. Cell models can be defined in header files or in a drive code itself and must inherit from *CellModelBaseFullyPartitioned*.

Within the model constructor, the names of the cell variables are set and the function *FinalizeConstruction* is called which invokes the base class, *CellModelBaseFullyPartitioned*, to assign appropriate memory for the cell variables. The index within the cell data at which membrane potential is stored is defined by the protected function *index_ of_ membrane_ potential_ in_ cell_ data* which is called by the *CellModelBaseFullyPartitioned* class, and hence protected. The value returned must match the label of membrane potential in *Names_ Of_ Cell_ Variables* in order for output labels to be correct.

Several functions are required to be overridden. In order for the model to set initial conditions the protected member function. *get_ initial_ state_ variable* is defined which assigns the value of cell variables and membrane potential respectively. For this example model, the initial conditions must be calculated numerically for the specified model parameters. Therefore, these functions check if the initial conditions have been calculated, if they have not then the function *calculate_ initial_ conditions* is called which uses the Newton method to calculate the initial conditions of the model to be those which guarantee a steady state.

The protected member function which defines how time steps are performed is given by *TakeTimestep* which calls either the explicit time-stepping or implicit time-stepping function depending on the value of a boolean flag.

The explicit time-stepping function is given by *ExplicitTimestep* which applies forward Euler to calculate the new variable values, and the implicit time-stepping function applies the Newton-Raphson method to solve the equations with backward Euler.

The additional functions which calculate the optional output and active strain or stress are given by *get_ output* and *GetActiveStrain*, which in biophysically detailed models are used to compute any information required by external models.

6.2 Calculating conduction velocity in a fibre of tissue

One of the simplest applications of the library is in the simulation of electrophysiology using a structured mesh representation of idealised geometries. To demonstrate this functionality, this section presents code snippets illustrating the process of generating a fibre of tissue, building and populating the mesh with instances of a cell model, and performing a series of external stimuli after which the conduction velocity within the fibre is measured.

The pacing external stimulus is given by the function *cell_source_func* which defines the stimulus applied to a point in the tissue at a given time. A pointer to this function is passed to the cells within the stimulus region within the construction of the cell mesh. The cell mesh class is defined by the following

```
//Define the cell mesh
template<class CONDUCTANCE_MODEL>
class CellAugmentedOneDMesh : public virtual OneDMesh<
    CONDUCTANCE_MODEL>,
                                        public virtual CellMeshBase<
                                            CONDUCTANCE_MODEL>
{
public:
  //Upon construction call the constructor of the OneDMesh
  CellAugmentedOneDMesh(const unsigned &Nx,  const double &Lx,
      TimeStepper* time_stepper_pt) :
    OneDMesh<CONDUCTANCE_MODEL>(Nx, Lx, time_stepper_pt),
    CellMeshBase<CONDUCTANCE_MODEL>(){this->FinalizeMeshSetup();}
  //Override the function called to build the cells in the mesh
  void BuildCells() override{
    //Loop over the nodes in the mesh
    const unsigned long n_node = this->nnode();
    for(unsigned long l=0; l<n_node; l++)
    {
      CELL_MODEL* cell_pt = this->template build_cell_at_node<
          CELL_MODEL>(l, l);
      const double x = this->node_pt(l)->x(0);
      cell_pt->set_cell_type(CELL_MODEL::LVEPI);
      if(x<Lx*0.6){cell_pt->set_cell_type(CELL_MODEL::LVMCELL);}
      if(x<Lx*0.25){cell_pt->set_cell_type(CELL_MODEL::LVENDO);}
      if(x<=source::stim_region){cell_pt->stim_func_pt() = &
          source::cell_source_func;}
    }
  }
};
```

which constructs the fibre mesh and then defines how the cells are built within it. The macro *CELL_MODEL* can be any generic cell model.

The problem constructor is very similar to those found in other Oomph-lib example driver codes. The time-stepper is defined, the meshes are built and added, the global mesh is built, equation numbers are assigned, the time-step is initialised, initial values are set for an impulsive start, and the initial conditions are assigned to the cells.

The major difference in this driver code to Oomph-lib example drivers is in time-stepping. Since, in this example, time-stepping is performed with Strang splitting, a call to *unsteady_newton_solve* is insufficient. Strang-Splitting is implemented by invoking *unsteady_newton_solve* as well as two additional *CellMeshBase* member functions: *Take_time_step_with_all_cells_in_mesh* and *SetNodeValuesToCellValues*. The implementation of Strang-splitting is given by the function

```
//Take a time-step using Strang splitting
void Timestep_StrangSplitting(const double& dt)
{
    // Cell timestep
    Cell_Mesh_pt->Take_time_step_with_all_cells_in_mesh(dt/2.0,
        this);
    // Project the solution to the nodes
    SetNodeValuesToCellValues();
    // Do a diffusion solve
    unsteady_newton_solve(dt);
    // Project the solution to the cells
    SetCellValuesToNodeValues();
    // Cell timestep
    Cell_Mesh_pt->Take_time_step_with_all_cells_in_mesh(dt/2.0,
        this);
    // Project the solution to the nodes (for outputting)
    SetNodeValuesToCellValues();
}
```

Here the cell models contained with the cell mesh take a half-time-step, the membrane potential values are projected to the finite element mesh, *unsteady_newton_solve* is called to compute the solution to the monodomain equation over a full-time-step, the membrane potential values are projected back to the cell models, the cell models contained with the cell mesh take a half-time-step, and finally, the membrane potential values are projected to the finite element mesh.

In the time-stepping loop, this function is invoked in the same way as *unsteady_newton_solve* is in other driver codes:

```
//The main time-stepping loop
for(unsigned istep=0;istep<nstep;istep++){problem.
    Timestep_StrangSplitting(dt);}
```

The functions *SetNodeValuesToCellValues* and *SetCellValuesToNodeValues* call generic functions which automatically handle the projection of membrane potential to and from the cells and finite element mesh respectively.

6.3 Adaptive operator splitting methods

The use of adaptive operator splitting introduces several changes to the driver code. Adaptive operator splitting requires the use of additional backup memory in the cell models. This is allocated and accessed by calling high-level functions during the construction of the problem and in the time-stepping procedure.

The first change made is to the amount of data allocated to the cell model. This is changed by modifying the argument of the cell mesh member function *build_cell_at_node*. In the above example, this argument was set to one since only one copy of the cell variables was required. However, in order to implement adaptive operator splitting, two additional copies are required: one for the value at the start of the time-step, and one for the solution to the lower order operator splitting scheme. The modified line of code in the cell mesh is then

```
CELL_MODEL* cell_pt = this->template build_cell_at_node<
    CELL_MODEL>(1, 3);
```

which informs the cell model constructor that three values are required instead of one.

A new time-stepping function is required to implement adaptive operator splitting. In this example, Strang-Milne is implemented. This function operates similarly to *adaptive_unsteady_newton_solve* in that it returns the next suggested time-step. It differs however in that if the measured operator splitting error is larger than the specified tolerance, the solution is reverted to the start of the time interval and the next suggested time step is returned. In *adaptive_unsteady_newton_solve* the function will invoke itself again with the next suggested time-step if the error of the previous time-step is too large.

```
double adaptive_Timestep_strang_milne(const double& dt){
    //Store initial conditions in backup 1.
    StoreStateInBackup(1);
    const double dt_pre = this->time_pt()->time();
    //Solve by Strang-splitting
    Timestep_StrangSplitting(dt);
    //Store the solution due to Strang splitting in backup 2.
    StoreStateInBackup(2);
    //Restore solution 1.
    RestoreStateFromBackup(1);
    this->time_pt()->time() = dt_pre;
    //Solve by Milne device, this solution is stored in backup 0.
    Timestep_StrangSplitting_Milne_Device(dt);
    //Calculate local error
    double error = ...;
    //Calculate new time-step
    double dt_next = ...;
    if(error > error_tolerance)
    {
      RestoreStateFromBackup(1);
      this->time_pt()->time() = dt_pre;
    }
    return dt_next;
}
```

The solution is computed for Strang-splitting and the Milne device counterpart to Strang-splitting. The operator splitting error measure will depend on the adaptive operator splitting method used. This error is calculated by the cell models by comparing the solution in backup 0 to the solution in another backup. The default implementation in the cell model base class calculates the relative error but can be overridden to suit other requirements.

Both Strang-splitting and the Strang-splitting Milne device represent valid operator splitting methods, the difference between their solutions is used to calculate the error in the solution due to Strang splitting. The main difference between these operator splitting methods and that used by the previous, non-adaptive, example is the use of *adaptive_ unsteady_ newton_ solve* rather than *unsteady_ newton_ solve*, which applies adaptive time-stepping to calculate the solution to the diffusion equation over the entire interval given by Δt. This is implemented in the following

```
//perform an adaptive unsteady newton over the entire interval [t
    , t+dt] inclusive
void adaptive_unsteady_newton_solve_over_complete_interval(const
    double& dt)
{
     const double t_end = this->time_pt()->time()+dt;
     double dt_current = dt;
     while(this->time_pt()->time()<t_end)
     {
        dt_current = adaptive_unsteady_newton_solve(dt_current,
             error_tolerance);
        dt_current = std::min(dt_current, t_end-this->time_pt()->
             time());
     }
}
```

which applies *adaptive_ unsteady_ newton_ solve* iteratively until the desired time $t + \Delta t$ has been reached and is not overstepped.

Chapter 7

Applying adaptive operator splitting methods for the efficient solution of cardiac electrophysiology models

Simulations of highly nonlinear and complex systems such as the cardiac tissue provide insights into the understanding of mechanisms underlying the complex dynamics of the system. Given the large scale and complexity of the system, this gives rise to the need to develop fast, efficient, stable and parallelizable numerical solvers for obtaining accurate solutions to the model. In this study, a temporally doubly adaptive method for the efficient solution of general cardiac electrophysiology models was developed. By testing and comparing the developed methods by using representative cardiac models, it was found that the application of adaptive operator splitting significantly improves the efficiency of solving the monodomain model of cardiac electrophysiology, and overall this improvement is model-independent. The developed method is highly parallelisable and capable of more efficiently and robustly solving cardiac cell and tissue simulations than Strang-Splitting. It is further demonstrated that adaptive time-steppers applied to the cell model often select time steps smaller than those commonly used in similar numerical studies, indicating that commonly used time steps in solving cell models may not be applicable to adaptive operator splitting. These results demonstrate that adaptive operator splitting schemes could improve the efficiency of numerical schemes used to solve electrophysiological models of the myocardium.

Biophysically detailed cardiac models consist of a large number of ODEs representing kinetics of ion channels, ion exchange and pumps, as well as intracellular Ca^{2+} handling, that are coupled with a small number of diffusive PDEs that are representative for intercellular membrane potential diffusion [130] [131] [132] [133]. The ODEs act to model source terms (i.e., the generation of cellular membrane potential) in the PDEs. The solution of the PDEs in turn affects the dynamical behaviour of the ODEs. Efficiently calculating the solution to this coupled system is an important field of active research since simulations of cardiac tissue are often very large and require enormous amounts of computational resources to solve the cardiac model equations [105]. Several methods have been developed which seek to tackle this problem, however, issues remain in balancing efficiency with accuracy [104]. There are a few examples of methods which utilise predictor-corrector error estimation to ensure the solution has converged to a suitable tolerance whilst minimising computational cost, even in highly efficient and specialised methods [108]. Additionally, the majority of adaptive methods proposed are based on qualitative and heuristic estimation of the stiffness of the cell equations with no general measure of error [134] [135] which is not guaranteed to achieve an acceptable error in the final solution. These methods are generally used since a heuristic estimation of stiffness is often sufficient to ensure convergence and often requires only a single evaluation of the cell equations. However, a quantitative measure of the error in a solution is important for verification purposes. Furthermore, when simulating new untested models, the robust convergence criteria provided by predictor-corrector methods substantially improve the trustworthiness and reproducibility of results over heuristic adaptive schemes.

In the field of computational cardiology, most of the widely used numerical methods have remained unchanged since their conception. These methods were generally developed for solving large-scale multi-physics problems on machines with comparatively little memory and computational power. For example, the partitioned method of using Rush-Larsen alongside forward Euler has been used for solving equations governing intracellular ion cycling kinetics and channel gating dynamics since it was developed in 1978 [96]. A number of attempts have been made to improve upon it with new algorithms, such as that of Sundnes [136], and Perego [137], however, the original Rush-Larsen method was demonstrated to outperform these newer methods for the majority of cell models tested [98].

However, with more abundant, faster memory and more powerful processors, alternative methods can be implemented which make use of predictor-corrector temporal adaptivity. During the time course of excitation, cardiac tissue simulations generally encounter both stiff and non-stiff time periods, during which the models are or are not numerically difficult to solve respectively. Non-temporally-adaptive methods provide no guarantee that they will suitably resolve the solution and may result in large errors during stiff periods. In general, a time stepper can take larger time steps when the governing equations are non-stiff but must take many smaller time steps when the equations become stiffer. Most cardiac cell models are only stiff during periods of rapid change in membrane potential during the phase-0 depolarisation phase, which occurs during only a relatively small time window within the action potential duration of the upstroke. Furthermore, at any given time during large-scale tissue simulations, only a relatively small fraction of the total population of cells in the tissue, corresponding to the excitation wavefront, are undergoing rapid change in membrane potential and therefore most cells can be solved with coarser time steps. A parallel method which allows for autonomous adaptivity for solving equations of each cell in the population tissue model could therefore result in improvements in the time taken to complete the simulation [138] [139].

The rapid change in the membrane potential during the upstroke phase in cell models is not the only source of error in solving cardiac model equations. Due to a large number of variables within cell models, the equations of sub-cellular and tissue-scale dynamics are partitioned so that each may be independently solved more efficiently. This splitting of the governing differential operators however introduces a further source of error [140] [141] which may be controlled through the application of predictor-corrector techniques to the operator splitting stage [18].

A recent study has made use of adaptive time stepping to solve the cell and the diffusion equations [142] in order to reduce error and improve stability. However, this method implements explicit methods for both sets of equations in order to improve scalability and memory use. Furthermore, a fixed super-time-step length is used and so the method may not be optimal for non-stiff periods of computation.

In this study, an alternative method where adaptive predictor-corrector algorithms [115] [114] [18] are instead applied to both the operator splitting stage and the cell equations is proposed and has been developed. The resultant temporally doubly adaptive method is highly parallelisable and capable of more efficiently and robustly solving cardiac simulations than the other methods tested. It is also shown that the adaptive cell solver not only allows for a more efficient solution but is required in order to ensure a numerically accurate and stable result when adaptive operator splitting is applied. It is further demonstrated that adaptive cell time-steppers often select time steps smaller than those commonly used in similar numerical studies.

7.1 Methods

7.1.1 Cardiac cell models

The general form of a cell model follows from that given in section 2.2. In order to demonstrate the model independence of the adaptive operator splitting schemes, two cell models are applied. Both the HRd 2011[103] and an SQT1 variant of the TNNP 2006 left ventricle model[130] which was developed[111] to simulate the N588K mutation on KCNH2-encoded hERG and is capable of sustaining persistent spiral waves were applied. Both cell models are applied to 1D simulations in order to determine conduction velocity. For 2D simulations, in order to facilitate the generation of re-entrant waves, only the SQT1 TNNP model was applied.

Diffusion Model

The monodomain equation is used to model electrical diffusion through the tissue, it is given by 2.57 with boundary conditions given by 2.58.

7.1.2 The operator splitting stage

Operator splitting is performed as per the description in 3.5.3. All operator splitting methods used are detailed there. Several operator-splitting methods are applied: Strang-splitting, palindromic Lie-Trotter, Strang-Milne, and symmetric Milne-32.

These splitting methods were selected since they all do not require backward time integration as they are second-order or less. It has been demonstrated that operator splitting methods of order three or greater could potentially be applied to some parabolic equations[143], including the monodomain and bidomain equations[144]. However, the application of these methods can introduce instability due to ill-posed backward time-integration of the parabolic PDEs. In order to remove the risk of instabilities due to backward integration confusing results, these methods are omitted. Adaptive operator splitting methods used are as follows. Palindromic Lie-Trotter which produces a second-order accurate solution and second-order accurate error measure. Strang-Milne is also second order and is produced by applying the Milne device to Strang-splitting. Symmetric Milne-32 is a combination of two methods, one of which is very similar to AM32 which is second order, which together can be used to produce an error measure.

7.1.3 Solving the monodomain equation

Two methods were applied to solve the monodomain equations. The more basic method of solving the monodomain equation was the commonly used Crank-Nicolson method [99]. Crank-Nicolson is unconditionally stable and second-order accurate[99] making it suitable for use with the potentially coarse time steps imposed by the adaptive operator splitting scheme and is given by 3.17

A more advanced method used to solve the monodomain equation was an adaptive backward Euler time-stepper. Here, the time-derivative was approximated using the backward Euler method, and the rate of change of each DOF was used to determine a predicted value from which an error approximation was calculated.

The weak form of the diffusion equations is obtained by multiplication with a test function and integration over the domain Ω. The weak form of the diffusion equation is then solved via the finite element method within the Oomph-lib open source multi-physics library[16] using a custom extension, and the resultant large linear system is solved at each iteration of the Newton method using the Mumps solver and openMPI. The Crank-Nicolson formulations of the Monodomain and Bidomain equations have been implemented as extensions to Oomph-lib.

7.1.4 Solving the cell equations

Due to the nature of adaptive operator splitting schemes, arbitrarily coarse time-steps could be imposed on the solver of the cell model. For such coarse time-steps, the commonly used Forward Euler with Rush-Larsen (FERL)[96] is unlikely to give a sufficiently accurate or stable solution.

For the cell models used, it is known that a maximum time-step of approximately 0.01 ms is required when they are solved with FERL. Therefore, when FERL is applied to solving them, a maximum time-step of 0.01 ms could be enforced and the method of solving cell equations outlined in section 3.5.3 would be applied.

Alternatively, adaptive time-stepping methods could be used. A commonly used adaptive method for solving single cell equations is that of Qu and Garfinkel [135]. Here, the time-interval is subdivided into K smaller intervals dependent on the magnitude of $\frac{\partial V_m}{\partial t}$. Larger magnitudes indicate potentially stiffer equations and a smaller Δt is required. It has been widely used [145] [146] [147] to provide more stable solutions to single cell equations than FERL. Since the adaptive time-step is chosen before any derivative evaluations are made, no partial solutions are discarded and fewer derivative functions are evaluated when compared to predictor-corrector adaptive methods. It was observed that this method diverged for many of the cell models tested but, when a finite solution was found, converged to a solution much faster than other adaptive methods tested. During the time-step selection of this method, single-cell variables other than transmembrane potential are ignored and so may result in large errors. Additionally, the method is not general and so is not guaranteed to converge for a general single cell model. It was found that a very small maximum time step value was required in order to ensure convergence for some models. This resulted in a much less efficient method when compared to predictor-corrector type adaptive time-steppers.

Heun-Euler (HE) is an explicit Runge-Kutta method. It compares the solutions given by the first order forward Euler (FE) and second order Heun methods. In the implementation of HE, checks for non-finite-values are made after the initial FE step. When non-finite-values are encountered, the solution is reverted to the start of the time interval and the time-step is halved. If the time-step is not discarded for the reason of encountering non-finite-values, then an error measure is calculated. This is used to inform the next time-step length, if it is below the target error tolerance, or to adjust and repeat the current time-step length if the error measure is above the target error tolerance. HE requires an additional function evaluation to compute the solution to each time interval when compared with FE. In terms of derivative evaluations, the efficiency of HE is double that of FERL for a successful time-step. To solve the cell equations over a given interval of length Δt, choice of the new time-step at each iteration of HE is made to ensure that the single cell equations are solved over exactly Δt. Each cell within the tissue is free to autonomously select appropriate time-steps independently of the other cells. Again, as in the operator splitting stage, a new time-step-length is chosen by the widely used method of [117].

7.1.5 The combined methods

To perform the target pattern and spiral wave simulations there were four combined methods used: Strang-splitting, which represents the de-facto method to which the others were compared, and the adaptive operator splitting methods, palindromic Lie-Trotter, Strang-Milne, Symmetric-Milne-32.

Strang-Splitting was implemented with the cell equations solved using a combination of forward Euler and Rush-Larsen depending on the cell variable solved. The resultant combined method is refered to here as SSFE. The adaptive operator splitting methods were implemented with the cell equations solved using Heun-Euler and are referred to here by the acronyms PLTHE, SMHE, and SM32HE. Adaptive operator splitting was applied with relative error tolerances of 0.1, 0.01, and 0.001.

The diffusion equations in all cases were solved using the Crank-Nicolson formulation.

To perform measurements of conduction velocity within a heterogeneous fibre, temporal adaptivity was applied to solving the monodomain equation and the cell equations were solved using the forward Euler and Rush-Larsen methods with a hard limit on the maximum time-step allowed. For these methods, as has been discussed previously, Crank-Nicolson is not suitable since the implementation within Oomph-lib does not provide an error measure. Instead the backward Euler time-stepper is applied. The combined methods for these simulations are referred to simply by their operator splitting name: Strang splitting, Palindromic Lie-Trotter, Strang-Milne, and symmetric-Milne-32. For these methods, the maximum error tolerance permitted for the adaptive time-stepping solution to the monodomain equation was set to the tolerance permitted in the operator splitting solve. As for the target pattern and spiral wave simulations, error tolerances of 0.1, 0.01, and 0.001 were applied.

7.1.6 Benchmark simulations

Adaptive operator splitting schemes are compared to Strang-splitting using four benchmark simulations: pacing of a heterogeneous fibre of cells, target pattern waves in a 2D square geometry, and re-entrant waves in a heterogeneous 2D square geometry.

1D fibre simulations

Conduction velocity is measured in a one-dimensional heterogeneous fibre so the results given by the adaptive operator splitting schemes can be compared to that of Strang-splitting. The fibre is of length 15 mm and consists of 100 2-node, one-dimensional, finite elements each of length 0.15 mm. Each node in the mesh corresponds to a model of a ventricular cell for a total of 101 cells in the fibre. The selected spatial resolution of 0.15 mm is similar to measured ventricular cell length which is approximately 0.8 mm-0.15 mm[148]. Cells in the mesh are of varying cell-type so as to represent spatial heterogeneity observed transmurally through the ventricular wall. As was used in other similar studies[149][150][151][152], the fibre comprises of 25% endocardial, 35% mid-myocardial, and 40% epicardial cells. The fibre is stimulated at the left most edge, that containing the endocardial cells, in a region of width 0.6 mm for a duration of 2 ms with current amplitude -52 pA/pF. This stimulus was applied a total of 10 times, one every 600 ms, before the 11th stimulus wave was measured for conduction velocity.

Conduction velocity is measured as a function of the conductivity D for each numerical scheme in order to compare results. The membrane potential of two cells, one at 25% and one at 75% along the length of the fibre, is measured after the 11th stimulus. The CV is calculated as $\Delta x/\Delta t$ where Δx is the distance between the two cells and Δt is the time delay between when $\mathrm{d}V_m/\,\mathrm{d}t$ was largest for the measured cells.

2D simulation of target pattern excitation waves

Target pattern excitation waves are simulated in a heterogeneous 2D tissue. Tissue of dimensions 9×9 mm and spatial resolutions $150 \times 150\,\mu m$ as used in previous studies [149] [153] is stimulated in the lower left corner in a patch of size 1×1 mm. The tissue is heterogeneous with 25% endocardial, 35% midmyocardial, and 40% epicardial left ventricular cells from the left most edge to the right. The simulation is run for 800 ms in order to capture both the excitation wave and subsequent repolarization of the tissue.

2D simulations of target pattern waves

A piece of 2D cardiac tissue is generated by extending the fibre mesh described previously perpendicular to the fibre by 50 mm, producing a rectangular mesh of dimensions 15×50 mm which is discretised by a spatial resolution $150 \times 150\,\mu m$ as was used in the original presentation of the SQT1 model [111]. Spatial heterogeneity is asserted in the direction of the short edge as described for the 1D fibre simulation. Conductivity in the fibre direction was the same as in the 1D simulations described previously, and anisotropic conduction was simulated by setting conductivity in the sheet direction to $D = 0.1\, m^2 s^{-1}$, which gave a conduction velocity of $0.5\, ms^{-1}$ which is close to observed values[154]. Re-entry is initiated by a standard S1-S2 protocol, wherein the S1 is applied to the endocardial region along the left-most edge of the substrate in order to invoke a planar excitation wave which propagates towards the epicardial region. The S2 stimulus is then applied to a region of tissue in the mid-myocardium region which is within the vulnerable window of the model.

7.2 Results

7.2.1 Measuring conduction velocity in a transmural left ventricle fibre for the HRd ventricle model

The conduction velocity within the fibre for each operator splitting method is presented in figure 7.1. The solution due to Strang-splitting with a time-step of $0.01\, ms$ is used as the benchmark to which the results due to the adaptive operator splitting methods are compared due to the prevalence and justification of this choice of time-step within the literature. Adaptive operator splitting methods were applied with a relative error tolerance in the operator splitting stage of 0.1. Of the adaptive operator splitting methods, Strang-milne in general produced the least error, with a relative error of less than 5% for conduction velocities greater than $1.2\, m/s$, figure 7.2. However, for conduction velocities less than $1.2\, m/s$ errors rose substantially up to approximately 15% for a conduction velocity of $0.1\, m/s$. The other adaptive operator splitting methods resulted in higher errors than Strang-Milne.

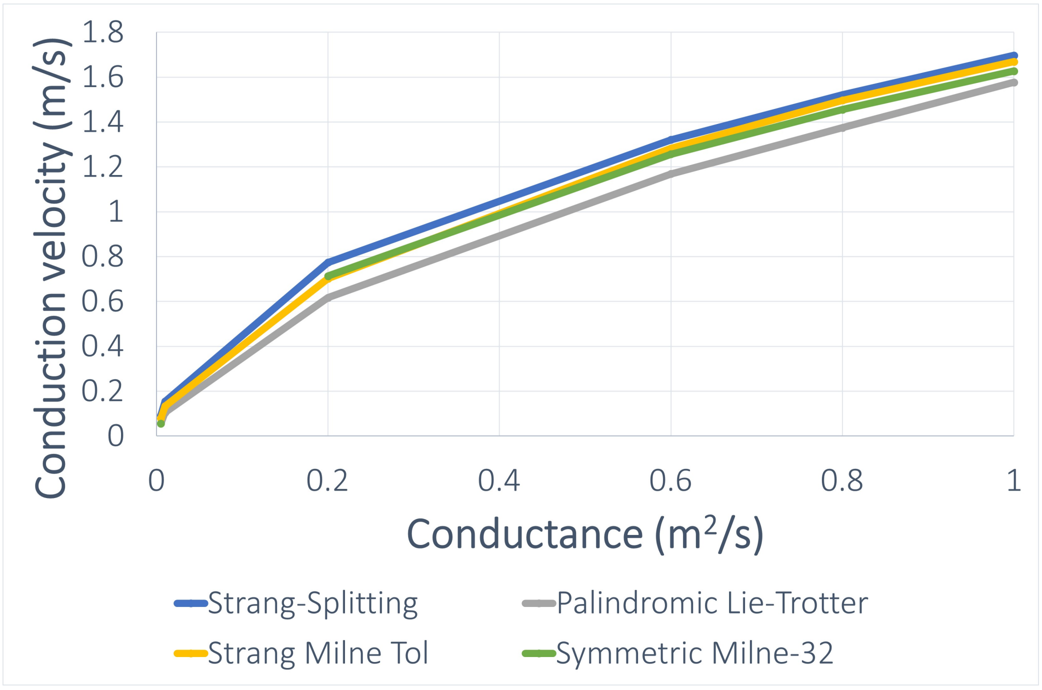


Figure 7.1: Conduction velocity is measured in a simulated heterogeneous transmural left ventricular fibre of the HRd cell model of length 15 mm. Strang splitting with a time step of 0.01 ms is compared to adaptive operator splitting methods each with a relative error tolerance of 0.1. The simulation is run for each method with conductances ranging from 0.05 m^2/s to 1.0 m^2/s and the resulting conduction velocity is measured for each.

It was found that for smaller values of relative error tolerances, the adaptive time-stepping heuristic used to solve the monodomain equation failed to converge. Furthermore for symmetric Milne-32 at a conductance of 0.01 m^2/s^2 the monodomain adaptive time-stepping failed in the same way for a relative error tolerance of 0.1. These failures in the adaptive time-stepping procedure to solve the monodomain equation occurred in all instances immediately following the application of the first external stimulus to the fibre. This could be because the backward Euler time-stepper is of relatively low order and could indicate that a higher-order adaptive time-stepper for solving

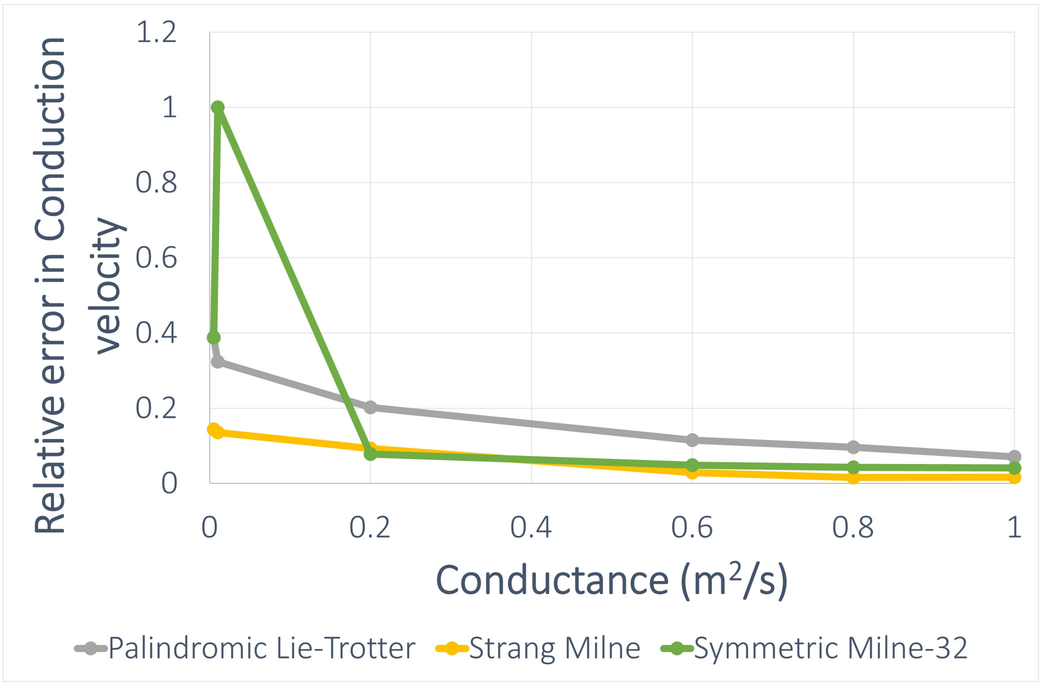


Figure 7.2: Relative error is calculated for the conduction velocity measured in a simulated heterogeneous transmural left ventricular fibre of length $15\,mm$. Strang splitting with a time step of $0.01\,ms$ is used to calculate the solution against which the relative error is calculated. The simulation is run for each method with conductances ranging from $0.05\,m^2/s$ to $1.0\,m^2/s$ and the resulting conduction velocity is measured for each.

the diffusion equations may be required.

All adaptive operator splitting methods with a relative error tolerance of 0.1 took less time to complete each simulation than Strang-splitting with a time-step of 0.01 ms.

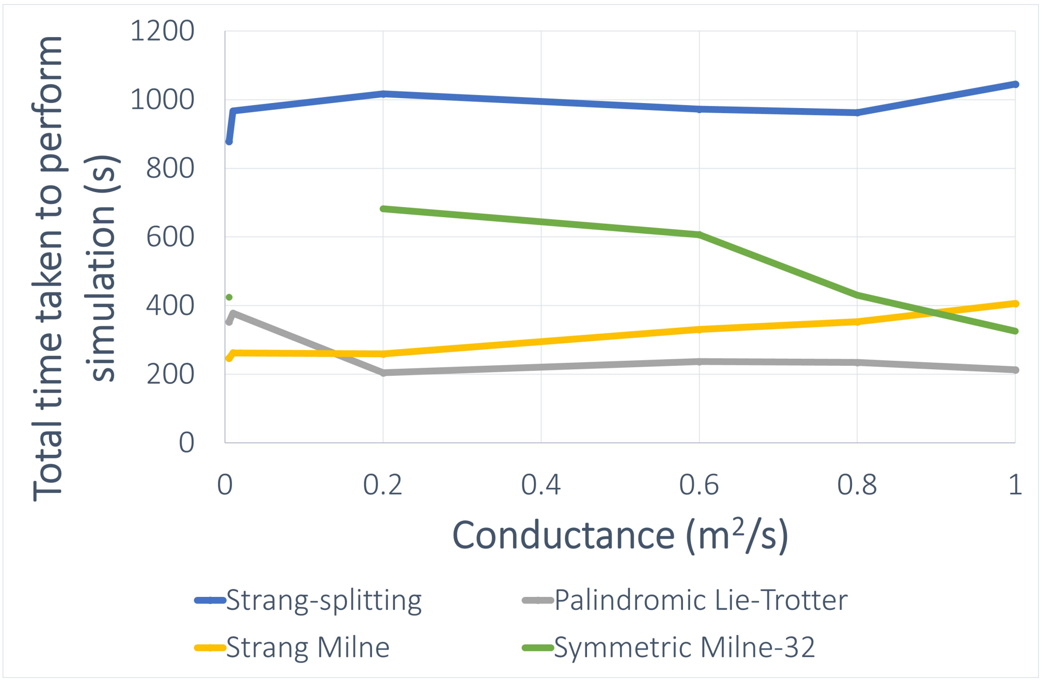


Figure 7.3: Total time taken to perform simulation of heterogeneous left ventricular fibre of length 15 mm. Strang splitting with a time step of 0.01 ms is compared to adaptive operator splitting methods each with a relative error tolerance of 0.1. Adaptive operator splitting methods took less time than Strang splitting with a time-step of 0.01 ms.

The Rate of Advance (ROA) is defined as the duration of the simulation in milliseconds achieved divided by the time taken to perform that simulation in seconds. ROA for the methods tested is presented in 7.4. All adaptive methods had a larger ROA than Strang splitting with a time-step of 0.01 ms.

The fraction of the entire computation time required for solving the monodomain equation and the cell equations is compared. The application of adaptive operator splitting methods appears to shift the computational load

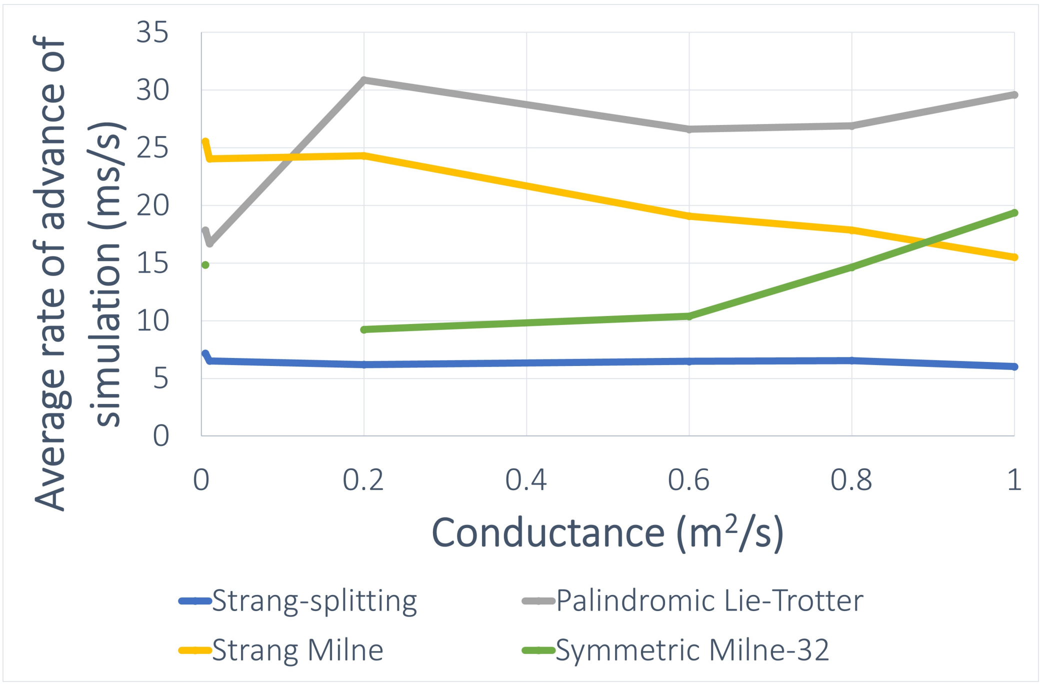


Figure 7.4: The average rate of advance of the simulation of a heterogeneous left ventricular fibre of length 15 mm. Strang splitting with a time step of 0.01 ms is compared to adaptive operator splitting methods each with a relative error tolerance of 0.1. The average rate of advance is defined here as the total simulation time (ms) divided by the total time taken to perform the simulation (s).

from solving the cell equations to solving the monodomain equation. Figure 7.5 shows that the fraction of the entire computational time applied to solving the monodomain equation was significantly higher for the adaptive operator splitting methods than for Strang-splitting.

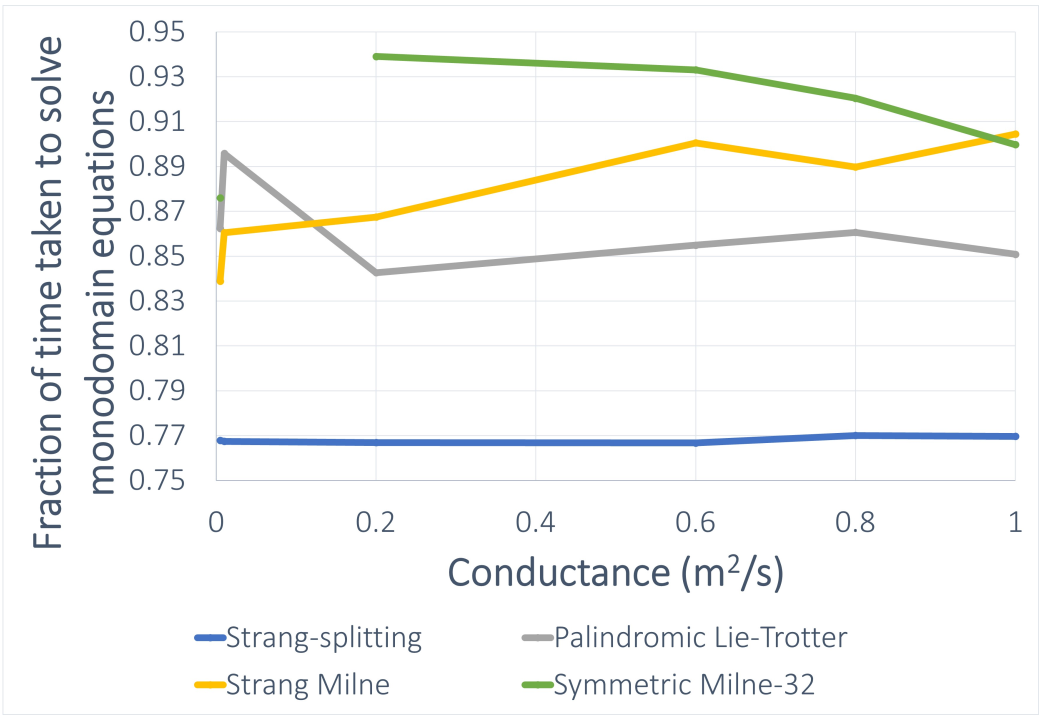


Figure 7.5: Fraction of the total simulation time taken to solve monodomain equation for a heterogeneous left ventricular fibre of length 15 mm. For every operator splitting method, each time-step requires computing the solution to the cell model equations and the monodomain equation. Since an adaptive time-stepping method is applied to solving the monodomain equation at each stage, the number of solves of the monodomain equation is not fixed and varies between time-steps and between methods. For Strang-splitting, the fraction of time spent solving the monodomain equation is relatively constant at around 0.77, however for the adaptive operator splitting methods a greater fraction of the total time taken to execute the simulation is spent solving the monodomain equations.

It is shown in figure 7.6 that the Strang-Milne and Palindromic Lie-Trotter methods require substantially fewer solves of the monodomain equation when compared to Strang-splitting with a times-step length of 0.01 ms.

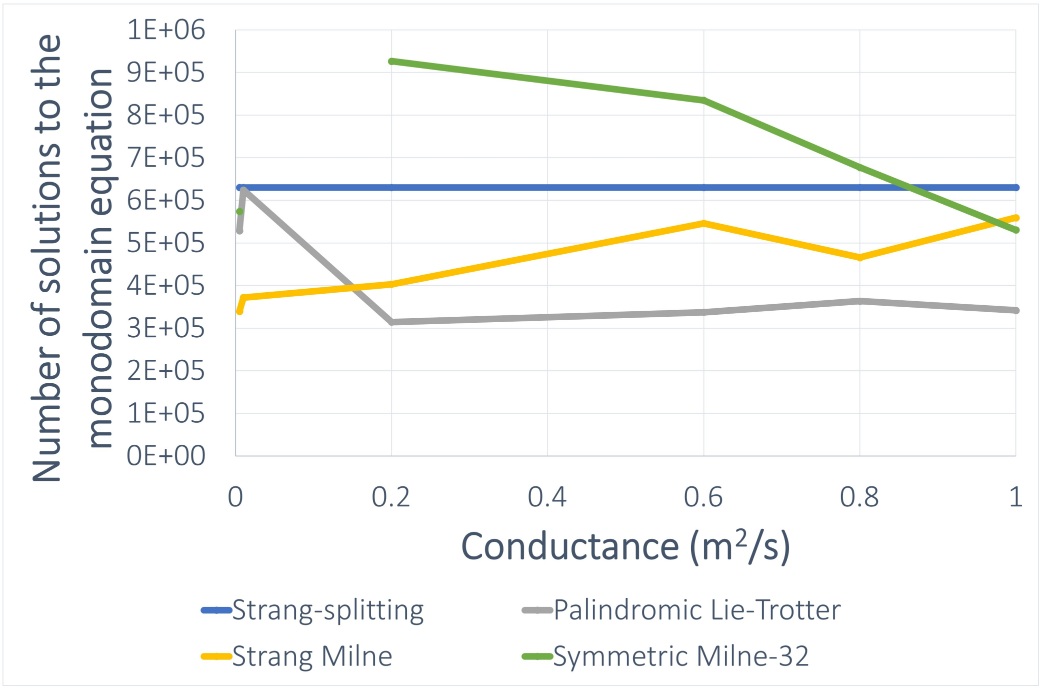


Figure 7.6: Total number of solves of the monodomain equation when simulating a heterogeneous left ventricular fibre of length 15 mm. Since an adaptive time-stepping method is applied to solving the monodomain equation at each stage, the number of solves of the monodomain equation is not fixed and varies between time steps and between methods. The total number of solves of the monodomain equation will, in general, depend on the number of solves required for computing each time-step and the size and number of time steps taken by the method in computing the solution. For Strang splitting, the number of solves of the monodomain equation is relatively constant, however, for the adaptive operator splitting methods, the number of solves of the monodomain equation varies significantly as the conductance is changed.

7.2.2 Target pattern wave simulations

In the 2D tissue model, the numerical schemes were first tested for simulating excitation waves of target pattern using both the HRd and TNNP06 models for cellular excitability. Results are shown in Figure 7.8. Snapshots are shown of target pattern excitation waves at $10\,ms$ (left panels) and $20\,ms$ (right panels) after the simulation computed from the HRd model using Strang-splitting with forward Euler and Rush-Larsen (SSFERL) ($\Delta t = 0.01\,ms$; Figure 7.8 A-B), Strang-Milne with Heun-Euler (SMHE) (Figure 7.8 C-D), Symmetric-Milne-32 with Heun-Euler (SM32HE) (Figure 7.8 E-F), and Palindromic-Lie-Trotter with Heun Euler (PLTHE) (Figure 7.8 G-H). In simulations, the excitation waves were evoked by a local stimulus applied to the left bottom corner of the tissue. With both HRd and TNNP06, simulation results on the wave-shape (i.e., smoothness of the curved wavefront) due to both SMHE, SM32HE, and PLTHE with target operator splitting error less than or equal to 0.01 matched those due to SSFERL with $\Delta t = 0.01\,ms$ (there is no variation throughout each column of Figure 7.8). The same simulation was run for the TNNP cell model, (Figure 7.7). It was found that although PLTHE was faster than both SMHE and SM32HE for an operator splitting error of 0.1, it produced oscillations in the initial wave-front which smoothed out as the wave travelled through the tissue. Furthermore, the time taken for PLTHE increases significantly faster than either SMHE or SM32HE as tighter restrictions on operator splitting error are imposed. PLTHE was not run for TNNP06.

The parallelisability of the numerical schemes was tested in solving the excitation of target pattern waves. Results in simulating the HRd model are shown in Table 7.1 and results of simulating the TNNP model are shown in Table 7.2. Target pattern waves were simulated for varying numbers of processors from 1 to 6. Computing a line of best fit of the form $T = AN^B$, where T is the total time taken to compute the simulation, N is the number of processors, and A and B are the parameters to be found, produces an estimation of how the computation time scales with the number of proces-

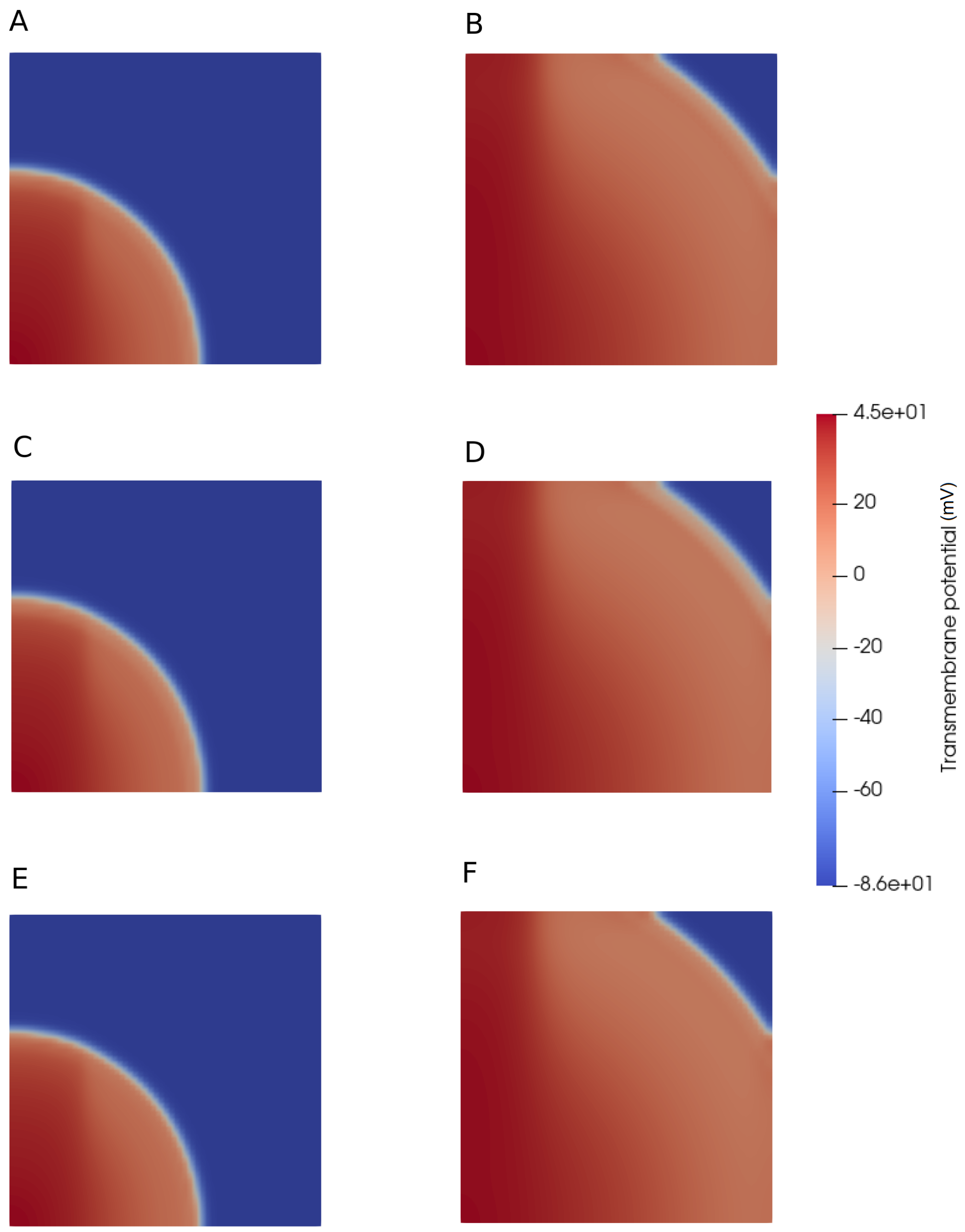


Figure 7.7: Snapshots of a target pattern wavefront through a 2D sheet consisting of the TNNP cell model. Results are shown for SSFERL with $\Delta t = 0.01\, ms$, and SMHE and SM32HE with target operator splitting error 1E-3. A, C, E: results for SSFERL, SMHE, SM32HE respectively at time $t = 10\, ms$. B, D, F: results for SSFERL, SMHE, and SM32HE respectively at time $t = 20\, ms$.

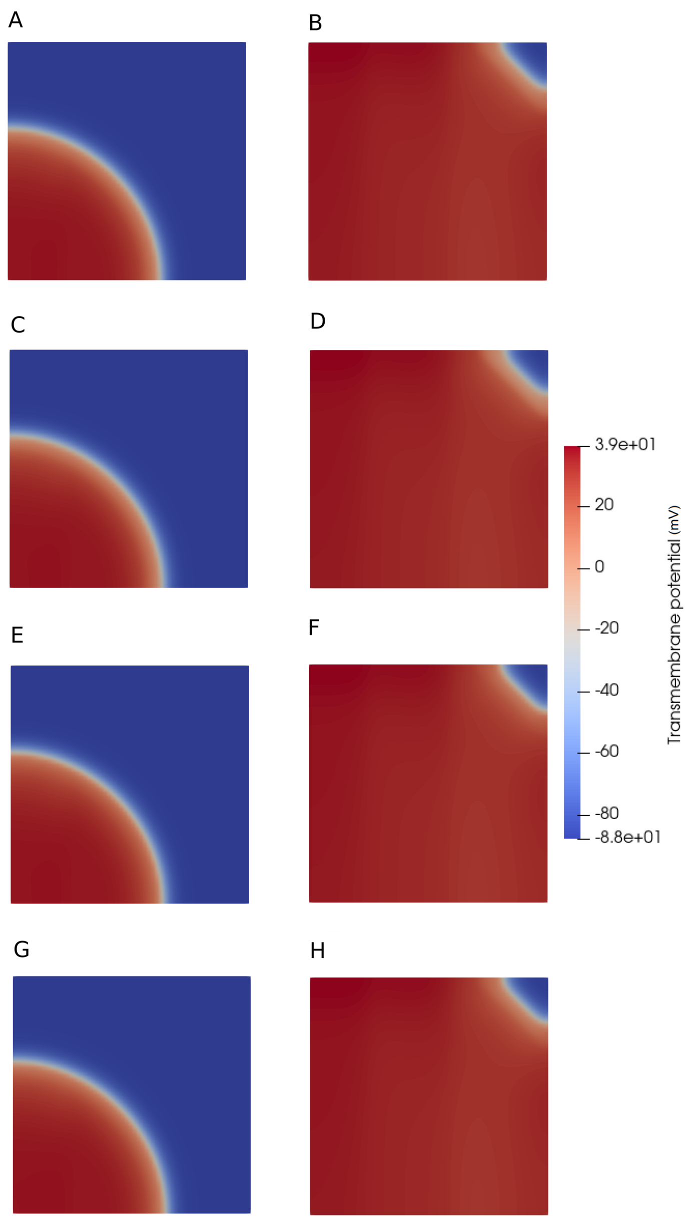


Figure 7.8: Snapshots of simulated target pattern excitation wave in a 2D sheet of HRd model with different operator splitting methods. Results were obtained by using SSFERL with $\Delta t = 0.01\,ms$, and SMHE, SM32HE, and PLTHE with target operator splitting error 1×10^{-3}. A, C, E, and G: results for SSFERL, SMHE, SM32HE, and PLTHE respectively at time $t = 10\,ms$. B, D, F, H: results for SSFERL, SMHE, SM32HE, and PLTHE respectively at time $t = 20\,ms$.

sors. More negative values of B indicate that the method scales better with increasing numbers of processors.

In simulating HRd, Strang-splitting produces $B = -0.47$, whereas Strang-Milne, Symmetric-Milne 32, and Palindromic-Lie-Trotter produce $B = -0.74$, $B = -0.635$, and $B = -0.50$ respectively with an operator splitting error of 0.001. This indicates that Strang-Milne is the most scalable of the methods tested, whereas palindromic-Lie-Trotter only slightly outperforms Strang-Splitting.

Num procs	Time Taken (s)									
	SSFERL	SMHE			SM32HE			PLTHE		
	Δt (ms)	OS error			OS error			OS error		
	0.01	0.1	0.01	0.001	0.1	0.01	0.001	0.1	0.01	0.001
1	3696	403	467	616	N/A	436	645	379	543	1130
2	2403	206	239	333	N/A	234	386	195	314	708
3	2061	142	177	256	N/A	170	288	136	233	552
4	1765	114	141	212	N/A	136	240	102	192	498
5	1571	95	122	187	N/A	117	221	84	176	467
6	1695	76	99	159	N/A	109	215	74	171	484

Table 7.1: Time taken to perform target pattern wave simulation with several operator-splitting methods for the HRd model. All operator splitting methods were applied with varying numbers of processors from 1 to 6. Adaptive operator splitting methods were applied with operator splitting errors of 0.1, 0.01, and 0.001. Strang-splitting was applied with a time-step of 0.01 ms. SM32HE did not converge to a finite solution for operator splitting error 0.1.

In simulating TNNP, Strang-splitting produces $B = -0.55$, whereas Strang-Milne and Symmetric-Milne 32 produce $B = -0.96$ and $B = -0.94$ respectively with an operator splitting error of 0.001. This indicates that Strang-Milne is the most scalable of the methods tested and is significantly more scalable than Strang-Splitting, however, there was little variance between Strang-Milne and symmetric-Milne 32.

The ROA for variant schemes for both HRd and TNNP models was inves-

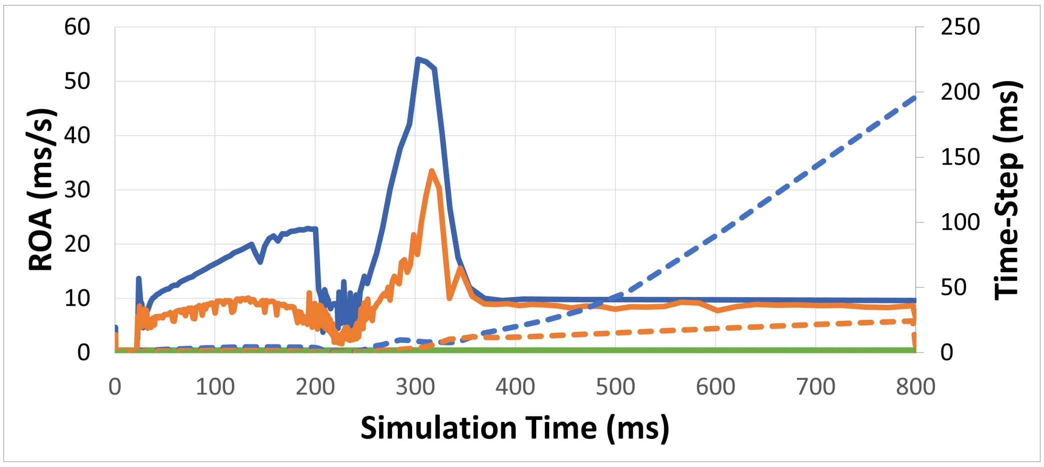
Num processors	Time Taken (s)						
	SSFERL	SMHE			SM32HE		
	Δt (ms)	OS error			OS error		
	0.01	0.1	0.01	0.001	0.1	0.01	0.001
1	3188.6	2054.2	2017.6	2117.5	2002.2	2040.5	2135.6
2	2090.7	1017.6	1015.7	1076.7	1008.8	1028.3	1088.2
3	1600.2	665.7	694.2	717.5	671.3	686.2	735.4
4	1428.6	503.5	514.4	547.4	506.2	520.4	565.8
5	1258.6	402.4	414.8	446.1	410.1	418.5	458.1
6	1201.5	337.0	356.6	379.2	343.4	350.5	396.6

Table 7.2: Time taken to perform planar wave simulation with various methods for the TNNP model. All operator splitting methods were applied with varying numbers of processors from 1 to 6. Adaptive operator splitting methods were applied with operator splitting errors of 0.1, 0.01, and 0.001. Strang-splitting was applied with a time-step of $0.01\,ms$.

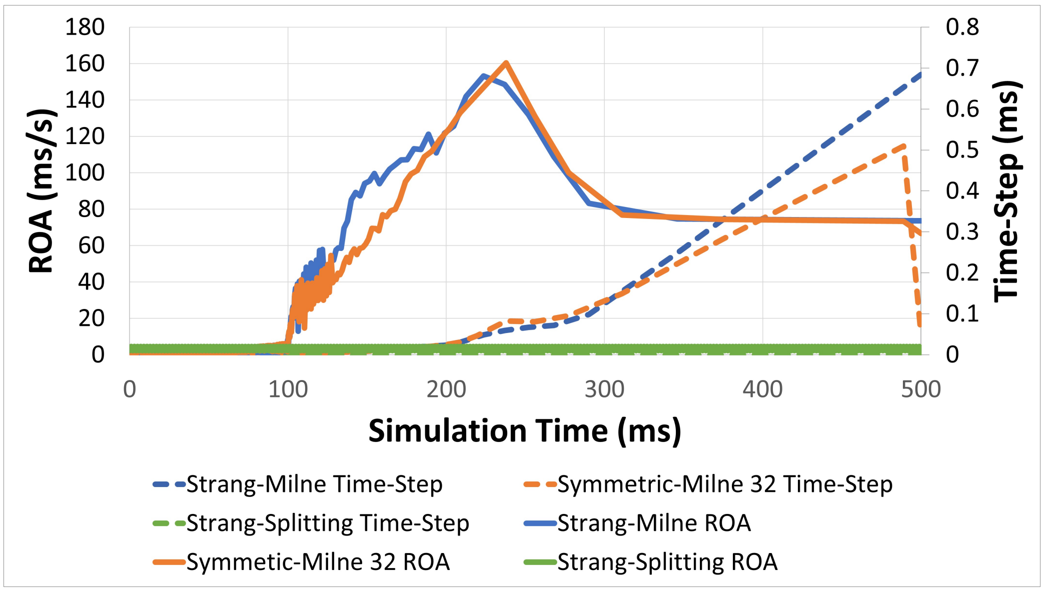
tigated. Results are shown in figure 7.10 and figure 7.11 for HRd and figure 7.9 for TNNP. Figure 7.10 A shows that for target pattern simulation, an OS error of 1E-3, ROA of SSFERL was greater than that of SMHE, SM32HE, and PLTHE whilst the wave-front was within the substrate ($t < 25\,ms$), although ROA of SMHE, SM32HE, and PLTHE was significantly higher than SSFERL once the wave-front left the substrate.

Similarly, with the TNNP cell model, figure 7.9 A demonstrates that SSFERL with $\Delta t = 0.01\,ms$ is faster than both SMHE and SM32HE while the wavefront is within the tissue. However, SSFERL is significantly slower than SMHE and SM32HE once the wavefront leaves the tissue. PLTHE was not used in solving the simulation with the TNNP cell model.

Additionally, it was found that with TNNP ROA was not necessarily improved by larger time-step values. Towards the end of the simulation (after approximately $525\,ms$) the ROA of both SMHE and SM32HE drops precipitously as the time-step increases and remains low until the end of the simulation. This occurs at a stage when very little spatial variation in the



(a)



(b)

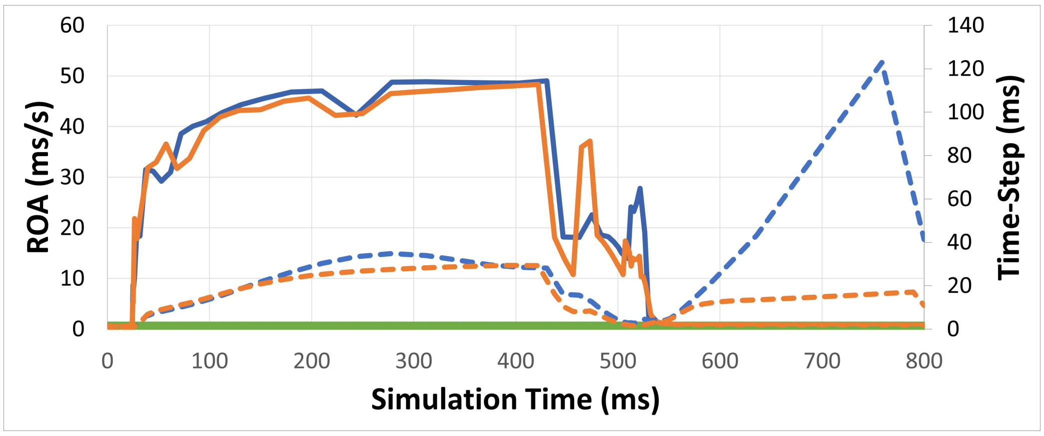
Figure 7.9: Rate of advance (ROA) of TNNP simulations of planar wave and spiral waves. A: Rate of advance of planar waves, ROA is plotted on the right y-axis against simulation time for SSFERL with $\Delta t = 0.01\,ms$, SMHE, SM32HE, PLTHE with OS error 1E-3. Time-step taken Δt is also plotted on the right y-axis against simulation time for each individual method. B: Rate of advance of spiral waves, ROA is plotted on the right y-axis against simulation time for SSFERL with $\Delta t = 0.01\,ms$, SMHE, SM32HE, PLTHE with OS error 1E-3. Time-step taken Δt is also plotted on the right y-axis against simulation time for each individual method.

solution exists and so little OS error is produced. This drop in ROA may be explained by stiffness in the single-cell equations necessitating smaller time-step lengths during the single-cell solve stage. However, it was observed that the reduced ROA persists even after the cells are recovered and their underlying variables are varying only slowly. This could be an indication that when applied to certain cell models, such as TNNP, this method may benefit from an upper limit on time-step length. Contrasted with the ROA of HRd, shown in Figure 7.10 A, which reaches a steady, high value as the time-step increases.

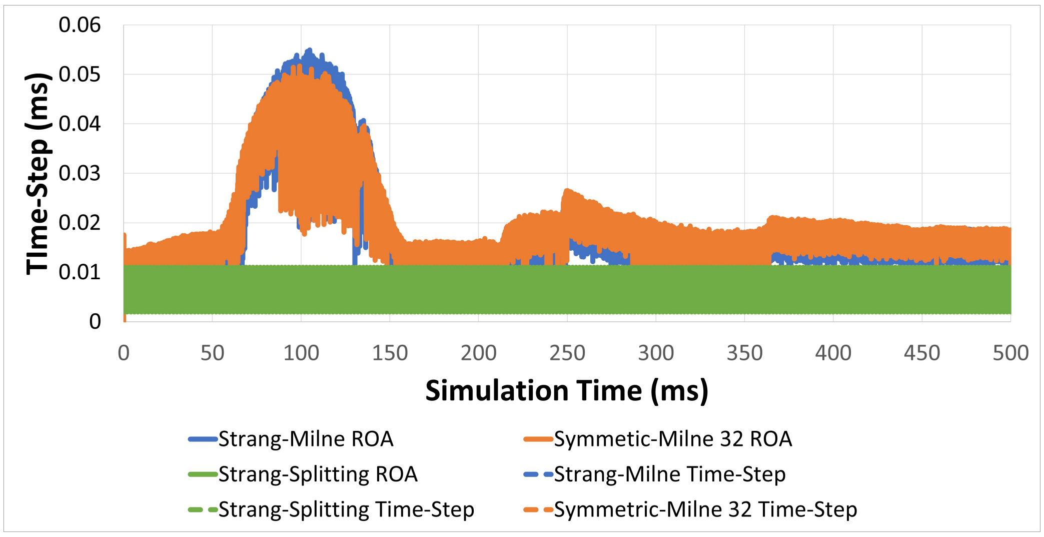
It was observed that SM32HE was not guaranteed to converge for a target operator splitting error of 0.1. For example, when applied to solving target pattern waves through a heterogeneous 2D substrate of HRd model cells, SM32HE stalled approximately 302ms into the simulation. Discontinuities which had emerged between the three regions resulted in excessively stiff conditions which required arbitrarily small time steps to solve and caused the simulation to stall. It was found that the break discontinuities caused the adaptive single-cell solver to take arbitrarily small time steps. A target error of 0.1, in this case, corresponds to a 10% relative error in the solution, it is therefore not believed that this negatively reflects the method since smaller target errors did not produce such discontinuities in the same simulation.

7.2.3 Spiral wave simulations

Similar investigations were also conducted in the 2D tissue model with spiral excitation waves. It was found that re-entrant waves were not generated in the substrate for the HRd model for any method tested. Results for HRd are shown in Table 7.3. This data shows that when applied to large 2D tissue simulations of the HRd model, SMHE, SM32HE, and PLTHE are faster than



(a)



(b)

Figure 7.10: Rate of advance (ROA) of HRd simulations of target pattern wave and spiral waves. A: The Rate of advance of target pattern waves, ROA is plotted on the right y-axis against simulation time for SSFERL with $\Delta t = 0.01\,ms$, SMHE, SM32HE, PLTHE with OS error 1E-3. Time-step taken Δt is also plotted on the left y-axis against simulation time for each individual method. B: Rate of advance of spiral waves, ROA is plotted on the right y-axis against simulation time for SSFERL with $\Delta t = 0.01\,ms$, SMHE, SM32HE, PLTHE with OS error 1E-3. Time-step taken Δt is also plotted on the right y-axis against simulation time for each individual method.

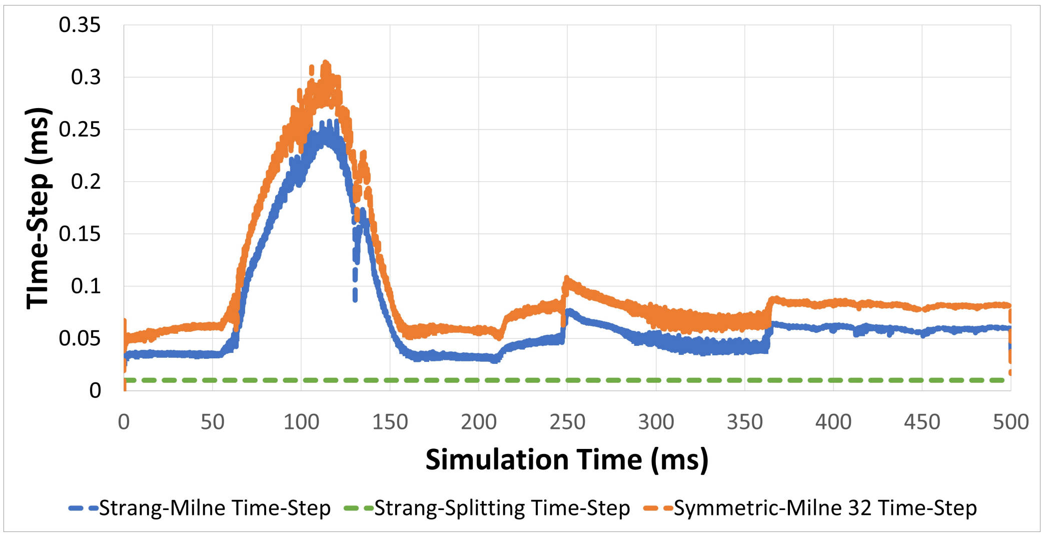


Figure 7.11: Time-step taken by several operator splitting methods in simulating HRd simulations of spiral waves. Strang-Splitting is simulated with a constant time-step of $0.01\,ms$, Symmetric-Milne 32 and Strang-Milne are used with an operator splitting error of 1E-3.

SSFERL with $\Delta t = 0.01\,ms$ for all tested target operator splitting errors. Table 7.3 A shows that all of SMHE, SM32HE, and PLTHE are faster in total than SSFERL with $\Delta t = 0.01\,ms$ for all tested OS errors. It is further shown in Table 7.3 B that SM32HE with OS error 1E-3 is faster than SSFERL even while the wave-front is within the tissue, whereas SMHE and PLTHE both with the same OS error were not. This may indicate that SM32HE provides a more reliable speed-up when simulations contain many cells than for smaller simulations. Complete numerical results for all methods and OS errors tested are shown in Table 7.3. Note that re-entrant wave simulations were only run on 6 processors.

Similarly, when applied to the TNNP cell model, SMHE and SM32HE were faster than SSFERL with a time-step of $\Delta t = 0.01\,ms$ for all tested OS errors. PLTHE was not run for TNNP. It was observed that for TNNP, re-entrant waves of delayed after depolarisations were generated for SSFERL

Time Taken (s)

SSFERL	SMHE			SM32HE			PLTHE		
Δt (ms)	OS error			OS error			OS error		
0.01	0.1	0.01	0.001	0.1	0.01	0.001	0.1	0.01	0.001
55252	3126	5092	12188	2324	4554	10027	1731	7540	29853

Table 7.3: Time taken to perform spiral wave simulation with various methods for the HRd model. All alternative methods (SMHE, SM32HE, and PLTHE) outperformed SSFERL with $\Delta t = 0.01\,ms$ for all chosen operator splitting errors.

with $\Delta t = 0.01\,ms$ and both SMHE and SM32HE with OS error 1E-2 and 1E-3, however, no re-entrant waves were generated for OS error of 1E-1. Table 7.4 A shows that, overall both SMHE and SM32HE were significantly faster than SSFERL with $\Delta t = 0.01\,ms$ for all OS errors tested. Additionally, Table 7.4 B further shows that at all stages of the simulation, SMHE and SM32HE progress faster than SSFERL. Complete numerical results for TNNP re-entrant wave simulations are shown in table 7.4.

Time Taken (s)

SSFERL	SMHE			SM32HE		
Δt (ms)	OS error			OS error		
0.01	0.1	0.01	0.001	0.1	0.01	0.001
45326	10321	15236	31162	10667	11947	27676

Table 7.4: Time taken to perform spiral wave simulation with various methods for the TNNP model. All alternative methods (SMHE, SM32HE, and PLTHE) outperformed SSFERL with $\Delta t = 0.01\,ms$ for all chosen operator splitting errors.

It was found that, during re-entrant wave simulation of TNNP, ROA of SMHE and SM32HE with OS error 1E-3 (figure 7.9 B) remain on average higher than that of SSFERL with $\Delta t = 0.01\,ms$ for the majority of the simulation. During this simulation, the time-step of the adaptive methods peaks between $0.25 - 0.3\,ms$, and so the drop in performance associated with larger time-step values observed in target pattern simulations was not observed.

It was found that a slight difference between the solution of SSFERL with $\Delta t = 0.01\,ms$ and SMHE and SM32HE with OS error 1E-3 emerged by the end of the $500\,ms$ simulation. Interrogating identical points within the two solutions shows (data not shown) that this is associated with a drift ($< 5ms$) in upstroke times between the two solutions.

7.3 Discussion

For simple, non-physiological test cases, it was found that the error was large when adaptive operator splitting methods were applied in the absence of a time-adaptive solution of the diffusion model and when the maximum time-step was enforced in the solution of the point-source model. For the same, non-physiological, test cases it was found that inclusion of time adaptive time-stepping to the solution of the diffusion model significantly improved the accuracy of the computed solution. In the context of a physiologically detailed model of a transmural fibre in the human left ventricle it was found that adding temporal adaptability to the solution of the diffusion equation was not sufficient to ensure that measured conduction velocities were accurate, when measured against the widely accepted Strang-splitting method, with an over-refined time-step. Additionally, for the symmetric-Milne-32 method, the time adaptive time-stepper used to solve the monodomain model failed to converge for specific conductances and the diffusion time-stepper failed to converge for all adaptive operator splitting methods tested when the required error tolerance was reduced further. This failure always occurred immediately following the application of an external electrical stimulus. This could indicate that temporal adaptability in the diffusion model alone, although sufficient to ensure accuracy for non-stiff test cases, is not sufficient to ensure that stiff physiological models converge when solved with adaptive operator splitting even when a maximum time-step in the solution of the cell models was applied.

The adaptive Heun-Euler method to solving the cell models was applied to test the response of simulation results to temporal adaptability in the solution of the point-source cell models. Adaptive operator splitting with temporal adaptability in the cell model and non-adaptability in the diffusion model was tested in 2D sheets of tissue simulating target pattern waves and induced re-entrant spiral waves. In these cases, it was found that execution time was substantially reduced when compared to Strang-splitting, by a factor of approximately 10 for smaller simulations and a factor of approximately 5 for larger ones. In contrast, the application of temporal adaptability in the diffusion model alone produced a speed-up by a factor of approximately 5 in the best case. This indicates that solving cardiac models with adaptive operator splitting methods would benefit from temporal adaptability in both the diffusion model and the cell models.

In general, it was found that the application of adaptive operator splitting can significantly improve the efficiency of solving the monodomain model of cardiac electrophysiology, and overall this improvement is model-independent. However, modifications in the form of a limit on the adaptive time-step size may further improve the method for some cell models such as TNNP06 which appeared to take considerably longer to solve for larger time-steps even when solved with adaptive time-stepping. The parallelizability of methods of adaptive operator splitting tested was greater than methods using non-adaptive operator splitting. It was also found that for small-scale simulations of target pattern waves, while the wave-front was within the substrate, a target operator splitting error of 0.01 or less resulted in methods which were slower than SSFERL but for large re-entrant wave simulations adaptive methods were faster at all stages of the simulation. It was demonstrated that for large target operator splitting error, PLTHE was faster than both SMHE and SM32HE in both small target pattern wave and large re-entrant wave simulations. However, for a smaller target operator splitting error, PLTHE was significantly slower than SMHE and SM32HE, this is likely due to the lower accuracy of PLTHE. Furthermore, for large target operator splitting errors, solutions obtained from PLTHE exhibited more spurious oscillations than the solutions obtained by either SMHE or SM32HE which could indicate that PLTHE is a less stable method.

It is clear that the use of adaptive operator splitting methods benefits from the additional application of adaptive time-steppers to solve both the diffusion model and the cell model. However, more work is needed to determine the most appropriate adaptive time-steppers for solving the individual sub-physics in this context.

When smaller relative error tolerances are enforced, the adaptive time-stepping heuristic for solving the monodomain equation often failed to converge for suitably large time steps. The Oomph-lib implementation of BDF methods produces a measure of the error in the solution. This error is calculated from the rate of change of the variables. It is unclear how this rate of change should be approximated in the case of operator splitting, where the initial conditions of the current time step are prescribed by the solution to the cell model equations. Adaptive operator splitting methods may be worth revisiting with a more suitable time-stepping method applied to solving the diffusion equation. It was found that applying BDF methods of more than one step to solving the monodomain equation with operator splitting resulted in an increased error. It is therefore unlikely that higher-order BDF methods will increase the accuracy of adaptive operator splitting methods. Higher-order time-steppers which use only the current values as initial conditions should be investigated. Commonly implicit methods are applied to the monodomain and bidomain equations for reasons of stability. However, high-order explicit Runge-Kutta methods may present a possible alternative. The inclusion of explicit time adaptive solution of the diffusion equations may further improve the numerical efficiency of this method [142]. Such time-stepping methods already exist within Oomph-lib, do not introduce additional DOFs, and produce reliable error measures for adaptive time-stepping.

Alternatively, implicit Runge-Kutta methods could be implemented. These, as for the Crank-Nicolson method, would require re-implementing a specific formulation of the monodomain and bidomain equations. Since implicit Runge-Kutta methods contain additional variables which must be calculated by solving a system of equations, implementing them within Oomph-lib will likely introduce additional DOFs to the specific implementations of the monodomain or bidomain equation elements. These additional DOFs will likely increase the numerical cost of solving the monodomain and bidomain equations. However, this effect may be mitigated by choosing the most suitable time-stepping methods and perhaps by developing and applying a preconditioner. Furthermore, the use of implicit Runge-Kutta methods will require a dummy time stepper which exists to perform essential Oomph-lib operations such as shifting time-values, but also to calculate the error based on the additional DOFs required by the Runge-Kutta method.

It has been demonstrated in another study[144] that third-order operator splitting methods can be used to solve the monodomain and bidomain equations. In order to avoid instabilities, an implicit Runge-Kutta method was applied to solve the monodomain and bidomain equations, and a high-order explicit Runge-Kutta method was applied to solve the cell model equations. Using these methods, it may be possible to apply adaptive third-order operator splitting and produce more stable and reliable results while improving efficiency.

Chapter 8

Future work

During this body of work, numerous areas were identified for future work. Initial thoughts for scope and implementation are noted in this study but represent significant areas for further development of oomph-lib and should be considered as such.

8.1 Validation of the remaining areas of the library

It became apparent during this study that validation of several areas of the numerical platform represent a large amount of further analysis, modelling, testing and resultant further validation. These areas are the bidomain elements, anisotropic solid elements, and validation of unstructured mesh generation from anatomical geometries.

Validation of these aspects of the library, as with all areas, is essential in order for the platform to be applied in cardiac research.

8.2 Implementation of Monodomain and Bidomain elements with alternative time-stepping techniques

Crank-Nicolson has long been established as a suitable method for the solution of cardiac electrophysiology simulations. However, it was demonstrated here that the accuracy of applying adaptive operator splitting methods is greatly improved when used in conjunction with adaptive time-stepping techniques to solve the monodomain equation. For the time-steppers which already exist as part of Oomph-lib, the backward Euler time-stepper was the only implicit, adaptive time-stepper which was found to be suitable for use with operator splitting techniques. However, issues were encountered with regard to convergence and accuracy for certain adaptive operator splitting techniques. To address this, alternative high-order adaptive time-stepping methods should be investigated. These methods should not require data from more than one previous time-step, since it has been shown in this body of work that methods such as high-order BDF time-steppers do not converge when used alongside operator splitting. Fractional time-stepping methods such as Runge-Kutta are a potential alternative, however, these require the introduction of additional DOFs which would negatively impact the efficiency of the method. Formulation of the residual and Jacobian formulations for the additional DOFs would be required. The resultant blocks of the Jacobian which correspond to the DOFs within each element will be very dense. However, Runge-Kutta methods provide error measures which are dependent entirely on the additional DOFs. This could result in a much more efficient adaptive time-stepping technique than using the Oomph-lib backward Euler time-stepper since history values before the current time interval are not taken into account.

Alternatively, it may be possible to construct higher-order time-stepping methods by exploiting the form of the diffusion equations which are used to simulate cardiac electrophysiology. There exist high-order time-stepping methods which only require data at the endpoints of the temporal interval. These methods improve the approximation of the Partial Differential Equation (PDE) over the time interval from a linear interpolation for Crank-Nicolson to arbitrarily higher order interpolations without the introduction of additional DOFs.

The general first order linear PDE,,

$$\frac{\partial u}{\partial t} = f(t, u(t)) \tag{8.1}$$

can be solved over the interval $[t_n, t_n + h]$ by constructing an approximation, $\mathcal{F}(t, u(t))$, to the integral of the function $f(t, u(t))$ over the interval. General N-point approximations can be constructed to have the form

$$\int\limits_{t_n}^{t_n+h} f(t, u(t)) \approx \sum_{i=0}^{I} \frac{\partial^i f(t_n, u(t_n))}{\partial t^i} \mathcal{T}_{i1} + \frac{\partial^i f(t_n + h, u(t_n + h))}{\partial t^i} \mathcal{T}_{i2} \tag{8.2}$$

where $\mathcal{T}_{i1}$ and $\mathcal{T}_{i2}$ are weights. However, 2-point approximations which use data only at the end-points of the interval have the advantage that no additional DOFs are introduced to the resulting numerical method.

The following approximation to 8.1 can then be constructed

$$u(t_n + h) - u(t_n) \approx \sum_{i=0}^{I} \frac{\partial^i f(t_n, u(t_n))}{\partial t^i} \mathcal{T}_{i1} + \frac{\partial^i f(t_n + h, u(t_n + h))}{\partial t^i} \mathcal{T}_{i2}. \tag{8.3}$$

This method is a special case of single-step Obrechkoff methods[155].

Single-step Obrechkoff methods can be of arbitrarily high order[155], and do not introduce the need for additional initial conditions as is required by multi-step methods, such as BDF. Furthermore, unlike Runge-Kutta, Obrechkoff methods do not necessitate additional DOFs. Obrechkoff methods do however introduce stricter requirements on the temporal differentiability of the terms in the PDE. In the context of the Monodomain and Bidomain equations, it is shown later that this requirement of temporal differentiability of terms in the governing equations is equivalent to stricter requirements for spatial differentiability and therefore requires the use of higher-order finite elements. Additionally, several terms must either be assumed as Natural boundary conditions or, if they cannot be assumed to be natural boundary conditions, added to the formulation at the boundaries of the domain.

A general Obrechkoff method is given by

$$\sum_{j=0}^{k} \alpha_i y_{n+j} = \sum_{i=1}^{l} h^i \sum_{j=0}^{k} \beta_{ij} y_{n+j}^{(i)}. \tag{8.4}$$

Obrechkoff methods of arbitrary order can be constructed by suitably choosing the values of k and l. Single-step methods are given when $k = 1$. It has been shown that the error constant decreases more rapidly for increasing l than increasing k. For values of $k > 1$ the method has more than two steps which have been shown to be unsuitable for use with operator splitting. Restricting $k = 1$ produces all single-step Obrechkoff methods which are of the form

$$\alpha_0 y_n + \alpha_1 y_{n+1} = \sum_{i=1}^{l} h^i (\beta_{i0} y_n^{(i)} + \beta_{i1} y_{n+1}^{(i)}). \tag{8.5}$$

Crank-Nicolson is a special case of this method, where $l = 1$, $a_0 = 1$, $a_1 = 1$, $\beta_{10} = 0.5$, and $\beta_{11} = 0.5$. These single-step methods can be constructed by considering approximations of the right-hand side of the differential equation

8.1 over the interval $[t_n, t_n + h]$ and then calculating the definite integral of the resultant approximation over the interval. The term $f(t, y(t))$ can be interpolated within the interval and, if such an interpolation uses only information at the end-points of the interval, t_{n+1} and t_n, then the resultant approximation will be a single step method, rather than a fractional or multi-step method.

Crank-Nicolson can be constructed in this way by choosing linear interpolation over the interval.

By considering a higher order cubic Hermite approximation which also interpolates the derivatives of $f(t, y(t))$ at the end-points of the interval, the fourth order Obrechkoff method which corresponds to $k = 1$, $l = 2$ can be determined where the corresponding weights of this method in equation 8.5 are $\alpha_0 = \alpha_1 = 1$, $\beta_{10} = \beta_{11} = 0.5$, $\beta_{20} = \frac{1}{12}$, and $\beta_{21} = -\frac{1}{12}$. In general, it may not be practical to compute terms containing $\frac{\partial f}{\partial t}$. However, the form of the Monodomain and Bidomain equations can be exploited to calculate them analytically in terms of higher-order spatial derivatives of the membrane potential.

For the Monodomain equation, $\frac{\partial f}{\partial t}$ takes the form

$$\frac{\partial f}{\partial t} = \underline{\nabla} \cdot \left(\frac{\partial D}{\partial t} \underline{\nabla} V + D \underline{\nabla} \left(\underline{\nabla} \cdot D \underline{\nabla} V \right) \right) \tag{8.6}$$

The resultant single-step Obrechkoff time-stepping method is of fourth order and contains derivatives of up to fourth order. Conversion to the weak form reduces the requirements on the spatial differentiability of the solution to second order which can be approximated by basis functions which already exist within Oomph-lib.

This method could improve the order and accuracy of numerical solutions when compared to that of Crank-Nicolson or backward Euler. When used in conjunction with a higher order operator splitting method or with an adaptive operator splitting method, it may allow for larger splitting time steps to be used since the error associated with the diffusion solve could be lower. However, it is possible that the error in such solutions may be dominated by the operator splitting error, in which case the application of such higher-order single-step time-steppers to solving the diffusion equations would be unlikely to result in any improvement to efficiency or overall accuracy.

Implementation of simulations using these Obrechkoff methods requires higher order basis functions than Crank-Nicolson in order to approximate the terms involving repeated derivatives. Within Oomph-lib, this corresponds to introducing additional nodes to the finite elements in the mesh which is used to solve the diffusion equations. Within the framework which was developed for handling cell models coupled with diffusion equations, this could result in changes to the spatial resolution of the cells and introduce more cells to the tissue. Such changes may affect the results of the simulation. This issue can be resolved by using separate meshes for solving the diffusion equations and the cell models individually. A diffusion mesh with high-order basis functions can then be used along with a coarser cell model mesh with linear interpolating basis functions.

The newly developed additions to the Oomph-lib library contain elements which can handle such separation of the diffusion equations and cell models into separate meshes. These elements utilise the Oomph-lib multi-physics procedures as well as additional DOFs to facilitate the projection of membrane potential between the meshes.

The separation of the meshes which are used for solving the separate physics presents the possibility of using an adaptive mesh for the solution of the diffusion equations. This isn't practical when the cell models and diffusion equations are handled within a single mesh, since the introduction of additional cells would potentially alter the solution. An adaptive diffusion mesh may provide a more accurate solution in the vicinity of the wave front and a more efficient solution in regions of tissue where there is little spatial variation in the cell model dynamics. However, since there is a significant numerical overhead associated with mesh refinement and un-refinement, it is possible that the cost of such methods may be greater than any potential efficiency gains.

This single-step Obrechkoff time-stepping method could be applied to performing adaptive time-stepping through consideration of an associated error measure. Since the use of single-step Obrechkoff time-stepping in Oomph-lib would require an explicitly implemented finite element, generic Oomph-lib time-steppers can not be used to compute the associated error. Instead, a placeholder time-stepper which computes the error will need to be implemented. This requirement is the same as for other similar implicit time-stepping techniques, such as implicit Runge-Kutta.

8.3 Additions of blood flow

Blood flow within the heart is multi-scale, the simulation of which requires several numerical and modelling approaches. Larger pools of blood that exist within the vessels are often modelled with Navier-Stokes or non-Newtonian models and simulated with FEM. Smaller vessels, such as capillaries, can be modelled as a branching network of one-dimensional tubes. Large areas of the circulatory system are often modelled with lumped sum parameter models, where complex organs or structures involving many processes are instead represented by a few variables representing the macro-scale behaviour. For a complete model of the circulatory system to be implemented, these distinct modelling approaches must be implemented along with methods of coupling them together.

8.3.1 Valve boundary conditions

Commonly used FEM approaches for simulating blood flow, such as Navier-Stokes and generic non-Newtonian fluid elements already exist within Oomph-lib. However, inlet and outlet conditions for the heart chambers require significant modelling. Blood flow in the inlets and outlets of the ventricles is regulated by valves. These valves, when operating normally, prevent blood from flowing back through the system. This improves the efficiency of the heart, since the regurgitation of blood back into the vessels during muscular relaxation is prevented, and is essential for proper heart function. Defects within the valves are known to cause a plethora of serious health risks.

The most realistic approach to modelling the heart values is through FSI and solid-solid interaction. Heart valves consist of leaflets which are in contact with one another when the valve is shut and separate as blood is pushed through them. The valve leaflets are connected to the papillary muscles via the chordae tendinae which together help to regulate the opening and closing of the valves. Under normal conditions, blood pressure within the heart chambers and contraction of the papillary muscles orchestrate to ensure proper opening and closing of the valves to ensure that blood is permitted to flow in only one direction.

A full FEM approach to modelling this system is very complex and would require the addition of several significant physics models, such as solid contact modelling, which do not currently exist within Oomph-lib. Instead, reduced parameter models of the valves could be considered. Such reduced parameter models have been successful in improving the realism of FEM models of blood flow within the heart chambers[156], however, their addition to Oomph-lib requires several modelling considerations.

A common approach of coupling reduced parameter heart valve models to FEM blood models is to allow the valve model to enforce parallel blood flow in only one direction. Such models assume that there is no regurgitation and also ignore the complex and well-documented interaction between the valve leaflets and the surrounding blood. However, their use has been shown to improve the predictive capabilities of blood flow simulations. If the valve boundaries are not aligned with the global basis vectors, then Lagrange multipliers will be needed to enforce parallel inflow-outflow to the fluidic elements. Implementation of this could take the form of a fluid face element which applies Lagrange multipliers to ensure that fluid flow only occurs normal to the element face.

Another modelling method is to apply lumped sum parameter models which represent the cross-sectional area of the valve opening and its dynamics in response to changes in fluid pressure across the valve. Such models are advantageous over simpler positive parallel flow boundary condition approximations since they account for the time-dependent nature of the valve leaflet dynamics, and thus the admittance of the valve, in response to blood pressure. Such models offer a compromise between detailed, computationally intensive finite element models of valve leaflets and simplified, more easily implemented and computed positive flow conditions.

For anatomical geometries, it is a significant problem to determine how external fluid boundaries should be identified automatically e.g. inlet, outlet. A potential approach is to manually label facets in the anatomical geometry.

These labels can then be read during the generation of the mesh and the elements treated accordingly.

8.4 Windkessel type models

The fluid dynamics within the heart are highly dependent on the conditions in the extra-cardiac space. For example, high impedance in the lungs is known to adversely affect the function of the left atrium and left ventricle, and atherosclerosis can affect the blood pressure within the left ventricle and thus impede proper contraction. There exist a large number of numerical models which approximate blood flow and conditions within distinct regions of the body. Windkessel models are lumped sum parameter models which contain several terms, each representative of a separate process or part of the circulatory system[157]. Although phenomenological, their use allows for predictive modelling of the effects of drugs and conditions which affect the heart or extra-cardiac systems. Windkessel models can vary in complexity and can represent several distinct processes within the circulatory system[158][159].

Implementation of Windkessel models within Oomph-lib would require the development of additional elements which are not used as part of a finite element mesh. Instead, the DOFs of any Windkessel elements would represent the variables in the Windkessel models and would have to be coupled to the FEM formulation in some manner.

This could be achieved by attaching Windkessel-coupling face elements to the inlet and outlet faces of the fluidic finite element mesh. These face elements would have to be developed to communicate various aspects of the FEM fluidic simulation, such as pressure and volume of fluid flow, to the lumped sum Windkessel models.

The inclusion of Windkessel models could allow for the simulation of lumped sum closed-loop cardiovascular simulations. These could then be further developed to introduce FEM modelling of other important areas of the cardiovascular system, such as blood flow in the pulmonary capillaries, or blood flow through the brain or placenta. Adding the capability of coupling FEM models of disparate areas of the body through lumped sum parameter models, such as those represented by Windkessel models, would represent an important and essential development in the functionality of Oomph-lib.

Chapter 9

Summary and conclusions

This work represents significant additions to the Oomph-lib numerical library as well as an investigation into the applicability of alternative operator splitting techniques to the simulation of cardiac models. Other areas were researched and are noted in Chapter 8 as further areas for study and later inclusion in oomph-lib

The application of alternative operator splitting methods to cardiac electrophysiology models was investigated. Currently, Strang-splitting represents the *de-facto* method for finding partitioned solutions to cardiac electrophysiology models. However, the application of Strang splitting requires manual tuning of the time step to ensure that error and stability in the resulting solution are acceptable. Furthermore, since models for cardiac electrophysiology represent electrical oscillators connected within a syncytium, both stiff and non-stiff periods are passed through during any simulation. A fixed time-step operator splitting method, such as Strang-splitting, may therefore not represent the most efficient method for performing such simulations. Three adaptive operator splitting methods were investigated in conjunction with adaptive time-stepping in the electrical conduction model, and adaptive time-stepping in the cell model. These methods demonstrated improved efficiency in the solution of the cardiac models but presented potential issues with accuracy and reliability. These issues could be addressed through the additional application of higher-order time-stepping to the electrical diffusion and point-source cell models in further research.

An efficient framework for coupling the partitioned solution of many point-source non-linear differential equations to the FEM solution of PDEs within Oomph-lib has been developed. Such a framework is essential for cardiac simulations where such partitioned solvers are commonly applied. However, this framework could also be applied to other physical systems where the decoupling of individual terms in the governing equations allows for a more efficient solution. Elements used in solving for anisotropic electrical diffusion models were implemented. These models were developed to operate fully within the pre-existing Oomph-lib environment and to adhere to the coding practices of the library. However, substantial additional functionality was also added which facilitates communication between these anisotropic electrical diffusion elements and non-linear point-source models. This functionality takes the form of a wrapper class for a finite element which, along with an associated wrapper for the Oomph-lib mesh class, performs the necessary spatial interpolation and communication of information to and from the point-source terms and finite elements. A generalisation of the pre-existing isotropic solid mechanics elements which solves for anisotropic solid deformation driven by an internal source of stress or strain was also developed. This extension is built upon the existing isotropic solid mechanics elements, from which the newly implemented anisotropic solid mechanics elements inherit. This inheritance-based implementation allows for the swapping of the new anisotropic solid mechanics elements for the original isotropic elements without changing driver codes. Additional terms which represent active stress or strain and the preferential vectors within the anisotropic solid are implemented in a generic fashion implemented throughout the pre-existing Oomph-lib elements. As such, these newly implemented anisotropic solid mechanics elements extend the functionality of the existing library without introducing added complexities when implementing driver codes. These additions to the Oomph-lib library were successfully applied in the study of adaptive operator splitting methods and were found to scale well with parallel computing. The body of work noted in this thesis represents large additions to the functionality of Oomph-lib, that while these are applicable to various areas of physical modelling, specifically create a computational platform for the simulation of biophysically detailed cardiac tissue models with general cell models. This platform consists of significant additions to Oomph-lib and allows for the novel application of Oomph-lib to cardiac modelling

Bibliography

[1] Gregory A. Roth, George A., et al. "Global Burden of Cardiovascular Diseases and Risk Factors, 1990–2019: Update From the GBD 2019 Study". In: *Journal of the American College of Cardiology* 76.25 (2020), pp. 2982–3021. ISSN: 0735-1097. DOI: `https://doi.org/10.1016/j.jacc.2020.11.010`. URL: `https://www.sciencedirect.com/science/article/pii/S0735109720377755`.

[2] Mark Michael Gallagher et al. "Distribution and prognostic significance of QT intervals in the lowest half centile in 12,012 apparently healthy persons". en. In: *Am. J. Cardiol.* 98.7 (Oct. 2006), pp. 933–935.

[3] Akira Funada et al. "Assessment of QT intervals and prevalence of short QT syndrome in Japan". en. In: *Clin. Cardiol.* 31.6 (June 2008), pp. 270–274.

[4] ESC Councils. "2015 ESC Guidelines for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death". In: *European Heart Journal* 36 (2015), pp. 2793–2867.

[5] Ciprian Rezuş et al. "QT interval variations and mortality risk: is there any relationship?" en. In: *Anatol. J. Cardiol.* 15.3 (Mar. 2015), pp. 255–258.

[6] Jason Gencher, Bishoy Deif, and Jason D. Roberts. “Short QT Syndrome”. In: *Electrocardiography of Inherited Arrhythmias and Cardiomyopathies: From Basic Science to Clinical Practice*. Cham: Springer International Publishing, 2020, pp. 41–50. ISBN: 978-3-030-52173-8. DOI: `10.1007/978-3-030-52173-8_3`. URL: `https://doi.org/10.1007/978-3-030-52173-8_3`.

[7] O. Anttonen et al. “Prevalence and Prognostic Significance of Short QT Interval in a Middle-Aged Finnish Population”. In: *Circulation* 116.7 (2007), pp. 714–720. DOI: `10.1161/CIRCULATIONAHA.106.676551`. eprint: `https://www.ahajournals.org/doi/pdf/10.1161/CIRCULATIONAHA.106.676551`. URL: `https://www.ahajournals.org/doi/abs/10.1161/CIRCULATIONAHA.106.676551`.

[8] Calum J Redpath et al. “Rapid genetic testing facilitating the diagnosis of short QT syndrome”. en. In: *Can. J. Cardiol.* 25.4 (Apr. 2009), e133–e135.

[9] Francesca Margara et al. “In-silico human electro-mechanical ventricular modelling and simulation for drug-induced pro-arrhythmia and inotropic risk assessment”. In: *Progress in Biophysics and Molecular Biology* 159 (2021). Mechanobiology of the Cardiovascular System, pp. 58–74. ISSN: 0079-6107. DOI: `https://doi.org/10.1016/j.pbiomolbio.2020.06.007`. URL: `https://www.sciencedirect.com/science/article/pii/S007961072030064X`.

[10] Richard B. Colquitt, Douglas A. Colquhoun, and Robert H. Thiele. “In silico modelling of physiologic systems”. In: *Best Practice & Research Clinical Anaesthesiology* 25.4 (2011). New Approaches in Clinical Research, pp. 499–510. ISSN: 1521-6896. DOI: `https://doi.org/10.`

1016/j.bpa.2011.08.006. URL: https://www.sciencedirect.com/science/article/pii/S1521689611000656.

[11] Houman Savoji et al. “Cardiovascular disease models: A game changing paradigm in drug discovery and screening”. In: *Biomaterials* 198 (2019). Organoids and Ex Vivo Tissue On-Chip Technologies, pp. 3–26. ISSN: 0142-9612. DOI: https://doi.org/10.1016/j.biomaterials.2018.09.036. URL: https://www.sciencedirect.com/science/article/pii/S0142961218306811.

[12] Jiahe Xi et al. “The estimation of patient-specific cardiac diastolic functions from clinical measurements”. In: *Medical Image Analysis* 17.2 (2013), pp. 133–146. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2012.08.001. URL: https://www.sciencedirect.com/science/article/pii/S1361841512001004.

[13] Nic Smith et al. “euHeart: personalized and integrated cardiac care using patient-specific cardiovascular modelling”. In: *Interface Focus* 1.3 (2011), pp. 349–364. DOI: 10.1098/rsfs.2010.0048. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsfs.2010.0048. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsfs.2010.0048.

[14] D.A. Nordsletten et al. “Coupling multi-physics models to cardiac mechanics”. In: *Progress in Biophysics and Molecular Biology* 104.1 (2011). Cardiac Physiome project: Mathematical and Modelling Foundations, pp. 77–88. ISSN: 0079-6107. DOI: https://doi.org/10.1016/j.pbiomolbio.2009.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0079610709000789.

[15] James Southern et al. "Multi-scale computational modelling in biology and physiology". In: *Progress in Biophysics and Molecular Biology* 96.1 (2008). Cardiovascular Physiome, pp. 60–89. ISSN: 0079-6107. DOI: `https://doi.org/10.1016/j.pbiomolbio.2007.07.019`. URL: `https://www.sciencedirect.com/science/article/pii/S0079610707000673`.

[16] Matthias Heil and Andrew L Hazel. "oomph-lib–an object-oriented multi-physics finite-element library". In: *Fluid-structure interaction*. Springer, 2006, pp. 19–49.

[17] Ismail Adeniran, Juldes Hancox, and Henggui Zhang. "In silico investigation of the short QT syndrome, using human ventricle models incorporating electromechanical coupling". In: *Frontiers in Physiology* 4 (2013). ISSN: 1664-042X. DOI: `10.3389/fphys.2013.00166`. URL: `https://www.frontiersin.org/articles/10.3389/fphys.2013.00166`.

[18] Winfried Auzinger et al. "Practical splitting methods for the adaptive integration of nonlinear evolution equations. Part I: Construction of optimized schemes and pairs of schemes". In: *BIT Numerical Mathematics* 57.1 (July 2016), pp. 55–74. DOI: `10.1007/s10543-016-0626-9`. URL: `https://doi.org/10.1007/s10543-016-0626-9`.

[19] Gerdes A.M. et al. "Regional differences in myocyte size in normal rat heart". In: *Anatomical Record 215* (1986).

[20] H. Satoh et al. "Surface:volume relationship in cardiac myocytes studied with confocal microscopy and membrane capacitance measurements: species-dependence and developmental effects". In: *Biophysical Journal 70* (1996).

[21] M.S. Spach et al. “Cell size and communication: role in structural and electrical development and remodelling of the heart”. In: *Heart Rhythm 1, 500-515* (2004).

[22] D.D.J. Streetner et al. “Fiber orientation in the canine left ventricle during diastole and systole”. In: *Circulation research 24, 339-347* (1969).

[23] D.M. Bers. “Excitation-Contraction Coupling and Cardiac Contractile Force”. In: *Springer, Dordrecht* (2008).

[24] S.E. Campbell, A.M. Gerdes, and T.D. Smith. “Comparison of regional differences in cardac myocyte dimensions in rats, hamsters, and guinea pigs”. In: *Anatomical Record 19, 53-59* (1987).

[25] Walter F. Boron. *Medical Physiology 2nd edition*. Philadelphia: Saunders. 9–21, 2009.

[26] A. Vander, J. Sherman, and D. Luciano. *human physiology: The Mechanisms of Body Function, seventh edition*. WCB/McGraw-Hill, 1998.

[27] E. Page. “Quantitative ultrastructural analysis in cardiac membrane physiology”. In: *American Journ1al of Physiology-Cell Physiology* 235.5 (1978), pp. C147–C158. DOI: `10.1152/ajpcell.1978.235.5.C147`. URL: `https://doi.org/10.1152/ajpcell.1978.235.5.C147`.

[28] A Katz. *Physiology of the heart Fifth edition*. Lippincott Williams and Wilkins, Philadelphia, 2010.

[29] PhD JEH. *Guyton and hall textbook of medical physiology*. Saunders, 2010.

[30] D.W. Fawcett and N.S. McNutt. “The ultrastructure of the cat myocardium. I. Ventricular papillary muscle”. In: *Journal of Cell Biology 42, 1-45* (1969).

[31] F. Brette and C. Orchard. "T-tubule function in mammalian cardiac myocytes". In: *Circulation research 92, 1182 - 1192* (2003).

[32] van der Velden et al. "Cardiac gap junctions and connexins: their role in atrial fibrillation and potential as therapeutic targets". In: *Cardiovascular Research* 54.2 (May 2002), pp. 270–279. DOI: `10.1016/S0008-6363(01)00557-0`. eprint: `http://oup.prod.sis.lan/cardiovascres/article-pdf/54/2/270/758665/54-2-270.pdf`. URL: `https://doi.org/10.1016/S0008-6363(01)00557-0`.

[33] Reed KE, Westphale EM, and Larson DM et al. "Molecular cloning and functional expression of human connexin37, an endothelial cell gap junction protein". In: *J Clin Invest;91(3):997–1004.* (1993).

[34] Delorme B, Dahl E, and Jarry-Guichard T et al. "Developmental regulation of connexin 40 gene expression in mouse heart correlates with the differentiation of the conduction system". In: *Dev Dyn;204(4):358–371.* (1995).

[35] Verheule S et al. "Characterization of gap junction channels in adult rabbit atrial and ventricular myocardium". In: *Circ Res;80(5):673–681* (1997).

[36] van Kempen MJ, Ten V I, and Wessels A et al. "Differential connexin distribution accommodates cardiac function in different species". In: *Microsc Res Tech;31(5):420–436.* (1995).

[37] Gourdie RG et al. "Immunolabelling patterns of gap junction connexins in the developing and mature rat heart". In: *Anat Embryol (Berl);185(4):363–378* (1992).

[38] A.L. Hodgkin and A.F. Huxley. “A Quantitative Description Of Membrane Current And Its Application To Conduction And Excitation In Nerve”. In: *The Journal of physiology* 117 (Sept. 1952), pp. 500–44. DOI: 10.1016/S0092-8240(05)80004-7.

[39] Dee Unglaub Silverthorn. *Human physiology*. Jones & Bartlett Publishers, 2015.

[40] Frances M Ashcroft. *Ion channels and disease*. Academic press, 1999.

[41] Augustus O Grant. “Cardiac ion channels”. In: *Circulation: Arrhythmia and Electrophysiology* 2.2 (2009), pp. 185–194.

[42] David Colquhoun and Alan G Hawkes. “The principles of the stochastic interpretation of ion-channel mechanisms”. In: *Single-channel recording*. Springer, 1995, pp. 397–482.

[43] Frank Lehmann-Horn and Karin Jurkat-Rott. “Voltage-gated ion channels and hereditary disease”. In: *Physiological reviews* 79.4 (1999), pp. 1317–1372.

[44] Dan M Roden et al. “Cardiac ion channels”. In: *Annual review of physiology* 64 (2002), p. 431.

[45] Graham L Collingridge et al. “A nomenclature for ligand-gated ion channels”. In: *Neuropharmacology* 56.1 (2009), pp. 2–5.

[46] Rémi Peyronnet, Jeanne M Nerbonne, and Peter Kohl. “Cardiac mechano-gated ion channels and arrhythmias”. In: *Circulation research* 118.2 (2016), pp. 311–329.

[47] Martin Fink and Denis Noble. “Markov models for ion channels: versatility versus identifiability and speed”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1896 (2009), pp. 2161–2179.

[48] Raimond L Winslow et al. “Mechanisms of altered excitation-contraction coupling in canine tachycardia-induced heart failure, II”. en. In: *Circ. Res.* 84.5 (Mar. 1999), pp. 571–586.

[49] Michael J Ackerman and David E Clapham. “Ion channels—basic science and clinical disease”. In: *New England Journal of Medicine* 336.22 (1997), pp. 1575–1586.

[50] Dario Di Francesco and Denis Noble. “A model of cardiac electrical activity incorporating ionic pumps and concentration changes”. In: *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 307.1133 (1985), pp. 353–398.

[51] Carmeliet E and Vereecke J. *Cardiac cellular electrophysiology 1st edition.* springer, New York, 2002.

[52] Erwin Shibata et al. “Contribution of transient outward current to repolarization in human atrium”. In: *The American journal of physiology* 257 (Jan. 1990), H1773–81. DOI: 10.1152/ajpheart.1989.257.6.H1773.

[53] Henry Gelband et al. “Electrophysiologic Properties of Isolated Preparations of Human Atrial Myocardium”. In: *Circulation research* 30 (Apr. 1972), pp. 293–300. DOI: 10.1161/01.RES.30.3.293.

[54] G.J. Amos et al. “Differences between outward currents of human atrial and subepicardial ventricular myocytes”. In: *The Journal of physiology* 491 (Pt 1) (Mar. 1996), pp. 31–50. DOI: 10.1113/jphysiol.1996.sp021194.

[55] Dobromir Dobrev et al. “Molecular Basis of Downregulation of G-Protein–Coupled Inward Rectifying K+ Current (IK,ACh) in Chronic Human Atrial Fibrillation Decrease in GIRK4 mRNA Correlates With

Reduced IK,ACh and Muscarinic Receptor–Mediated Shortening of Action Potentials". In: *Circulation* 104 (Dec. 2001), pp. 2551–7. DOI: `10.1161/hc4601.099466`.

[56] Yanggan Wang et al. "Differences in transient outward current properties between neonatal and adult human atrial myocytes". In: *Journal of molecular and cellular cardiology* 35 (Oct. 2003), pp. 1083–92. DOI: `10.1016/S0022-2828(03)00200-1`.

[57] D Escande et al. "Age-related changes of action potential plateau shape in isolated human atria! fibers". In: *The American journal of physiology* 249 (Nov. 1985), H843–50. DOI: `10.1152/ajpheart.1985.249.4.H843`.

[58] Arnold M Katz. *Physiology of the Heart.* Lippincott Williams & Wilkins, 2010.

[59] Fan-Yen Lee et al. "Electromechanical properties of Purkinje fiber strands isolated from human ventricular endocardium". In: *The Journal of heart and lung transplantation* 23.6 (2004), pp. 737–744.

[60] Emmanuel Drouin et al. "Electrophysiologic characteristics of cells spanning the left ventricular wall of human heart: evidence for presence of M cells". In: *Journal of the American College of Cardiology* 26.1 (1995), pp. 185–192.

[61] Paul M. Heerdt and George J. Crystal. "Chapter 20 - Cardiovascular Physiology: Cellular and Molecular Regulation". In: *Pharmacology and Physiology for Anesthesia.* Ed. by Hugh C. Hemmings and Talmage D. Egan. Philadelphia: W.B. Saunders, 2013, pp. 351–365. ISBN: 978-1-4377-1679-5. DOI: `https://doi.org/10.1016/B978-1-4377-1679-`

5.00020-X. URL: https://www.sciencedirect.com/science/article/pii/B978143771679500020X.

[62] C. Abrahams, J.S. Janicki, and K.T. Webet. "Myocardial hypertrophy in Macacafascicularis. Structural remodelling of the collagen matrix". In: *Laboratory Investigation 56, 676-683* (1987).

[63] H Ju and I.M. Dixon. "Exctracellular matrix and cardiovascular diseases". In: *Canadian Journal of Cardiology 12, 1259-1267* (1996).

[64] A.J. Pope et al. "Three dimensional transmural organisation of the perimysial colalgen in the heart". In: *American Journal of Physiology (Heart and Circulatory Physiology) 295, H11243-H1252* (2008).

[65] Douglas P Zipes. "Braunwald's heart disease: a textbook of cardiovascular medicine". In: *BMH Medical Journal-ISSN 2348–392X* 5.2 (2018), pp. 63–63.

[66] Yoram Rudy and Jonathan R Silva. "Computational biology in the study of cardiac ion channels and cell electrophysiology". In: *Quarterly reviews of biophysics* 39.1 (2006), pp. 57–116.

[67] Aslak Tveito and Glenn T Lines. *Computing characterizations of drugs for ion channels and receptors using Markov models*. Springer Nature, 2016.

[68] Martin Fink et al. "Cardiac cell modelling: observations from the heart of the cardiac physiome project". In: *Progress in biophysics and molecular biology* 104.1-3 (2011), pp. 2–21.

[69] David Roxbee Cox and Hilton David Miller. *The theory of stochastic processes*. Methuen, 1965.

[70] C H Luo and Y Rudy. "A dynamic model of the cardiac ventricular action potential. II. Afterdepolarizations, triggered activity, and potentiation." In: *Circulation Research* 74.6 (1994), pp. 1097–1113. DOI: 10.1161/01.RES.74.6.1097. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.RES.74.6.1097. URL: https://www.ahajournals.org/doi/abs/10.1161/01.RES.74.6.1097.

[71] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4 (1952), pp. 500–544. DOI: https://doi.org/10.1113/jphysiol.1952.sp004764. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764. URL: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764.

[72] Bertil Hille. *Ion Channels of Excitable Membranes*. Vol. 18. Sinauer Associates, Jan. 2001, pp. 1–814. ISBN: 0878933212.

[73] Martin Morad et al. *Molecular physiology and pharmacology of cardiac ion channels and transporters*. Vol. 182. Springer Science & Business Media, 2012.

[74] Louis J DeFelice. *Electrical properties of cells: patch clamp for biologists*. Springer Science & Business Media, 1997.

[75] Albert Edward Green and Wolfgang Zerna. *Theoretical elasticity*. Courier Corporation, 1992.

[76] Mario Delmar et al. "Chapter 8 - Molecular Organization and Regulation of the Cardiac Gap Junction Channel Connexin43". In: *Cardiac Electrophysiology (Fourth Edition)*. Ed. by DOUGLAS P. ZIPES and JOSÉ JALIFE. Fourth Edition. W.B. Saunders, 2004, pp. 66–76. ISBN:

978-0-7216-0323-0. DOI: https://doi.org/10.1016/B0-7216-0323-8/50011-7. URL: https://www.sciencedirect.com/science/article/pii/B0721603238500117.

[77] Habo J. Jongsma and Ronald Wilders. "Gap Junctions in Cardiovascular Disease". In: *Circulation Research* 86.12 (2000), pp. 1193–1197. DOI: 10.1161/01.RES.86.12.1193. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.RES.86.12.1193. URL: https://www.ahajournals.org/doi/abs/10.1161/01.RES.86.12.1193.

[78] M J van Kempen et al. "Spatial distribution of connexin43, the major cardiac gap junction protein, in the developing and adult rat heart." In: *Circulation Research* 68.6 (1991), pp. 1638–1651. DOI: 10.1161/01.RES.68.6.1638. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.RES.68.6.1638. URL: https://www.ahajournals.org/doi/abs/10.1161/01.RES.68.6.1638.

[79] Piero Colli Franzone, Luca F Pavarino, and Simone Scacchi. "A numerical study of scalable cardiac electro-mechanical solvers on HPC architectures". In: *Frontiers in physiology* 9 (2018), p. 268.

[80] R.H. Clayton et al. "Models of cardiac tissue electrophysiology: Progress, challenges and open questions". In: *Progress in Biophysics and Molecular Biology* 104.1 (2011). Cardiac Physiome project: Mathematical and Modelling Foundations, pp. 22–48. ISSN: 0079-6107. DOI: https://doi.org/10.1016/j.pbiomolbio.2010.05.008. URL: https://www.sciencedirect.com/science/article/pii/S0079610710000362.

[81] E.J. Vigmond et al. "Solvers for the cardiac bidomain equations". In: *Progress in Biophysics and Molecular Biology* 96.1 (2008). Cardiovascular Physiome, pp. 3–18. ISSN: 0079-6107. DOI: https://doi.org/10.1016/j.pbiomolbio.2007.07.012. URL: https://www.sciencedirect.com/science/article/pii/S0079610707000740.

[82] P. Colli Franzone, L. F. Pavarino, and S. Scacchi. "Effects of mechanical feedback on the stability of cardiac scroll waves: A bidomain electro-mechanical simulation study". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 27.9 (2017), p. 093905. DOI: 10.1063/1.4999465. eprint: https://doi.org/10.1063/1.4999465. URL: https://doi.org/10.1063/1.4999465.

[83] Piero Colli Franzone, Luca F. Pavarino, and Simone Scacchi. "A Numerical Study of Scalable Cardiac Electro-Mechanical Solvers on HPC Architectures". In: *Frontiers in Physiology* 9 (2018). ISSN: 1664-042X. DOI: 10.3389/fphys.2018.00268. URL: https://www.frontiersin.org/articles/10.3389/fphys.2018.00268.

[84] Ramesh M Gulrajani. *Bioelectricity and biomagnetism.* J. Wiley, 1998.

[85] L.Joshua Leon and B.Milan Horáček. "Computer model of excitation and recovery in the anisotropic myocardium: I. Rectangular and cubic arrays of excitable elements". In: *Journal of Electrocardiology* 24.1 (1991), pp. 1–15. ISSN: 0022-0736. DOI: https://doi.org/10.1016/0022-0736(91)90077-Y. URL: https://www.sciencedirect.com/science/article/pii/002207369190077Y.

[86] Vicky Y Wang et al. "Modelling passive diastolic mechanics with quantitative MRI of cardiac structure and function". In: *Medical image analysis* 13.5 (2009), pp. 773–784.

[87] Gerhard A. Holzapfel and Ray W. Ogden. “Constitutive modelling of passive myocardium: a structurally based framework for material characterization”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1902 (2009), pp. 3445–3475. DOI: 10.1098/rsta.2009.0091. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2009.0091. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2009.0091.

[88] L L Demer and F C Yin. “Passive biaxial mechanical properties of isolated canine myocardium”. en. In: *J. Physiol.* 339.1 (June 1983), pp. 615–630.

[89] Richard Bellman. *Introduction to matrix analysis.* SIAM, 1997.

[90] Ming Yang and Larry A Taber. “The possible role of poroelasticity in the apparent viscoelastic behavior of passive cardiac muscle”. In: *Journal of biomechanics* 24.7 (1991), pp. 587–597.

[91] N. P. Smith et al. “Multiscale computational modelling of the heart”. In: *Acta Numerica* 13 (2004), pp. 371–431. DOI: 10.1017/S0962492904000200.

[92] Pras Pathmanathan and Jonathan P Whiteley. “A numerical method for cardiac mechanoelectric simulations”. en. In: *Ann. Biomed. Eng.* 37.5 (May 2009), pp. 860–873.

[93] P.J. Hunter, A.D. McCulloch, and H.E.D.J. ter Keurs. “Modelling the mechanical properties of cardiac muscle”. In: *Progress in Biophysics and Molecular Biology* 69.2 (1998), pp. 289–331. ISSN: 0079-6107. DOI: https://doi.org/10.1016/S0079-6107(98)00013-3.

URL: https://www.sciencedirect.com/science/article/pii/S0079610798000133.

[94] Simone Rossi et al. "Orthotropic active strain models for the numerical simulation of cardiac biomechanics". In: *International Journal for Numerical Methods in Biomedical Engineering* 28.6-7 (2012), pp. 761–788. DOI: https://doi.org/10.1002/cnm.2473. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.2473. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.2473.

[95] Martyn P Nash and Alexander V Panfilov. "Electromechanical model of excitable tissue to study reentrant cardiac arrhythmias". en. In: *Prog. Biophys. Mol. Biol.* 85.2-3 (June 2004), pp. 501–522.

[96] Stanley Rush and Hugh Larsen. "A Practical Algorithm for Solving Dynamic Membrane Equations". In: *IEEE Transactions on Biomedical Engineering* BME-25.4 (1978), pp. 389–392. DOI: 10.1109/TBME.1978.326270.

[97] Ismail Adeniran et al. "Abnormal calcium homeostasis in heart failure with preserved ejection fraction is related to both reduced contractile function and incomplete relaxation: an electromechanically detailed biophysical modeling study". In: *Frontiers in Physiology* 6 (2015). ISSN: 1664-042X. DOI: 10.3389/fphys.2015.00078. URL: https://www.frontiersin.org/article/10.3389/fphys.2015.00078.

[98] Megan E. Marsh, Saeed Torabi Ziaratgahi, and Raymond J. Spiteri. "The Secrets to the Success of the Rush–Larsen Method and its Generalizations". In: *IEEE Transactions on Biomedical Engineering* 59.9 (2012), pp. 2506–2515. DOI: 10.1109/TBME.2012.2205575.

[99] J. Crank and P. Nicolson. "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 43.1 (1947), pp. 50–67. DOI: 10.1017/S0305004100023197.

[100] C F Curtiss and J O Hirschfelder. "Integration of stiff equations". en. In: *Proc. Natl. Acad. Sci. U. S. A.* 38.3 (Mar. 1952), pp. 235–243.

[101] Matthias Heil, Andrew L Hazel, and Jonathan Boyle. "Solvers for large-displacement fluid–structure interaction problems: segregated versus monolithic approaches". In: *Computational Mechanics* 43.1 (2008), pp. 91–101.

[102] Richard FitzHugh. "Impulses and Physiological States in Theoretical Models of Nerve Membrane". In: *Biophysical Journal* 1.6 (1961), pp. 445–466. ISSN: 0006-3495. DOI: https://doi.org/10.1016/S0006-3495(61)86902-6. URL: https://www.sciencedirect.com/science/article/pii/S0006349561869026.

[103] Thomas O'Hara et al. "Simulation of the Undiseased Human Cardiac Ventricular Action Potential: Model Formulation and Experimental Validation". In: *PLOS Computational Biology* 7.5 (May 2011), pp. 1–29. DOI: 10.1371/journal.pcbi.1002061. URL: https://doi.org/10.1371/journal.pcbi.1002061.

[104] J.P. Whiteley. "An Efficient Numerical Technique for the Solution of the Monodomain and Bidomain Equations". In: *IEEE Transactions on Biomedical Engineering* 53.11 (2006), pp. 2139–2147. DOI: 10.1109/TBME.2006.879425.

[105] James A Southern et al. “Solving the coupled system improves computational efficiency of the bidomain equations”. In: *IEEE Transactions on Biomedical Engineering* 56.10 (2009), pp. 2404–2412.

[106] Sergio Blanes and Fernando Casas. “On the necessity of negative coefficients for operator splitting schemes of order higher than two”. In: *Applied Numerical Mathematics* 54.1 (2005), pp. 23–37. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/j.apnum.2004.10.005`. URL: `https://www.sciencedirect.com/science/article/pii/S0168927404002259`.

[107] Shev MacNamara and Gilbert Strang. “Operator Splitting”. In: *Splitting Methods in Communication, Imaging, Science, and Engineering*. Cham: Springer International Publishing, 2016. Chap. 3, pp. 95–114. ISBN: 978-3-319-41589-5. DOI: `10.1007/978-3-319-41589-5_3`. URL: `https://doi.org/10.1007/978-3-319-41589-5_3`.

[108] Steven Niederer et al. “Simulating Human Cardiac Electrophysiology on Clinical Time-Scales”. In: *Frontiers in Physiology* 2 (2011). ISSN: 1664-042X. DOI: `10.3389/fphys.2011.00014`. URL: `https://www.frontiersin.org/article/10.3389/fphys.2011.00014`.

[109] Shankarjee Krishnamoorthi, Mainak Sarkar, and William S. Klug. “Numerical quadrature and operator splitting in finite element methods for cardiac electrophysiology”. In: *International Journal for Numerical Methods in Biomedical Engineering* 29.11 (2013), pp. 1243–1266. DOI: `https://doi.org/10.1002/cnm.2573`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.2573`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.2573`.

[110] Pras Pathmanathan et al. “Computational modelling of cardiac electrophysiology: Explanation of the variability of results from different numerical solvers”. In: *International Journal for Numerical Methods in Biomedical Engineering* 28 (Aug. 2012), pp. 890–903. DOI: `10.1002/cnm.2467`.

[111] Ismail Adeniran et al. “Increased Vulnerability of Human Ventricle to Re-entrant Excitation in hERG-linked Variant 1 Short QT Syndrome”. In: *PLOS Computational Biology* 7.12 (Dec. 2011), pp. 1–16. DOI: `10.1371/journal.pcbi.1002313`. URL: `https://doi.org/10.1371/journal.pcbi.1002313`.

[112] William B. Gragg and Hans J. Stetter. “Generalized Multistep Predictor-Corrector Methods”. In: *J. ACM* 11.2 (Apr. 1964), pp. 188–209. ISSN: 0004-5411. DOI: `10.1145/321217.321223`. URL: `https://doi.org/10.1145/321217.321223`.

[113] Richard Wesley Hamming. “Stable predictor-corrector methods for ordinary differential equations”. In: *Journal of the ACM (JACM)* 6.1 (1959), pp. 37–47.

[114] William Edmund Milne and WE Milne. *Numerical solution of differential equations*. Vol. 64. Wiley New York, 1953.

[115] Erwin Fehlberg. *Classical fifth-, sixth-, seventh-, and eighth-order Runge-Kutta formulas with stepsize control*. National Aeronautics and Space Administration, 1968.

[116] O. Koch, Ch Neuhauser, and Mechthild Thalhammer. “Embedded exponential operator splitting methods for the time integration of nonlinear evolution equations”. In: *Applied Numerical Mathematics* 63 (Jan. 2013), pp. 14–24. DOI: `10.1016/j.apnum.2012.09.002`.

[117] Roland Bulirsch, Josef Stoer, and J Stoer. *Introduction to numerical analysis*. Springer, 1991.

[118] Andrew L. Hazel. “Spatial and Temporal Adaptivity in Numerical Studies of Instabilities, with Applications to Fluid Flows”. In: *Computational Modelling of Bifurcations and Instabilities in Fluid Dynamics*. Cham: Springer International Publishing, 2019, pp. 75–115. ISBN: 978-3-319-91494-7. DOI: `10.1007/978-3-319-91494-7_3`. URL: `https://doi.org/10.1007/978-3-319-91494-7_3`.

[119] David M. Harrild and Craig S. Henriquez. “A finite volume model of cardiac propagation”. In: *Annals of Biomedical Engineering* 25.2 (Mar. 1997), pp. 315–334. DOI: `10.1007/bf02648046`. URL: `https://doi.org/10.1007/bf02648046`.

[120] Paul Brocklehurst et al. “A 2D electromechanical model of human atrial tissue using the discrete element method”. en. In: *Biomed Res. Int.* 2015 (Oct. 2015), p. 854953.

[121] Matthias Heil and A Hazel. *Oomph-lib documentation*. `http://oomph-lib.maths.man.ac.uk/doc/html/index.html`. Accessed: 2020-04-15. 2017.

[122] Matthias Heil and Andrew L. Hazel. “oomph-lib – An Object-Oriented Multi-Physics Finite-Element Library”. In: *Fluid-Structure Interaction*. Ed. by Hans-Joachim Bungartz and Michael Schäfer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 19–49. ISBN: 978-3-540-34596-1.

[123] Matthias Heil, Andrew Hazel, and Amine Massit. *Linking VMTK with oomph-lib*. `http://oomph-lib.maths.man.ac.uk/doc/meshes/mesh_from_vmtk/html/index.html`. Accessed: 2020-05-4. 2017.

[124] S Manini and L Antiga. *VMTK website*. http://www.vmtk.org/. Accessed: 2020-06-24. 2020.

[125] Matthias Heil and Andrew Hazel. *Oomph-lib The data structure*. http://oomph-lib.maths.man.ac.uk/doc/the_data_structure/html/index.html. Accessed: 2020-06-12. 2017.

[126] Thomas Scrase. *user_src*. Version 1.0.0. Apr. 2023. URL: https://github.com/thomas-scrase/user_src.

[127] Marina Strocchi et al. "A publicly available virtual cohort of four-chamber heart meshes for cardiac electro-mechanics simulations". In: *PLOS ONE* 15.6 (June 2020), pp. 1–26. DOI: 10.1371/journal.pone.0235145. URL: https://doi.org/10.1371/journal.pone.0235145.

[128] Jason D Bayer et al. "A novel rule-based algorithm for assigning myocardial fiber orientation to computational heart models". In: *Annals of biomedical engineering* 40.10 (2012), pp. 2243–2254.

[129] D F Scollan et al. "Reconstruction of cardiac ventricular geometry and fiber orientation using magnetic resonance imaging". en. In: *Ann. Biomed. Eng.* 28.8 (Aug. 2000), pp. 934–944.

[130] K H W J Ten Tusscher and A V Panfilov. "Cell model for efficient simulation of wave propagation in human ventricular tissue under normal and pathological conditions". In: *Physics in Medicine and Biology* 51.23 (Nov. 2006), pp. 6141–6156. DOI: 10.1088/0031-9155/51/23/014. URL: https://doi.org/10.1088/0031-9155/51/23/014.

[131] K. H. W. J. ten Tusscher et al. "A model for human ventricular tissue". In: *American Journal of Physiology-Heart and Circulatory Physiology* 286.4 (2004). PMID: 14656705, H1573–H1589. DOI: 10.

1152/ajpheart.00794.2003. eprint: https://doi.org/10.1152/ajpheart.00794.2003. URL: https://doi.org/10.1152/ajpheart.00794.2003.

[132] Yoram Rudy and Jonathan R. Silva. “Computational biology in the study of cardiac ion channels and cell electrophysiology”. In: *Quarterly Reviews of Biophysics* 39.1 (2006), pp. 57–116. DOI: 10.1017/S0033583506004227.

[133] Jakub Tomek et al. “Development, calibration, and validation of a novel human ventricular myocyte model in health, disease, and drug block”. In: *eLife* 8 (Dec. 2019). Ed. by José D Faraldo-Gómez et al., e48890. ISSN: 2050-084X. DOI: 10.7554/eLife.48890. URL: https://doi.org/10.7554/eLife.48890.

[134] Bernard Victorri et al. “Numerical integration in the reconstruction of cardiac action potentials using Hodgkin-Huxley-type models”. In: *Computers and Biomedical Research* 18.1 (1985), pp. 10–23. ISSN: 0010-4809. DOI: https://doi.org/10.1016/0010-4809(85)90003-5. URL: https://www.sciencedirect.com/science/article/pii/0010480985900035.

[135] Zhilin Qu and A. Garfinkel. “An advanced algorithm for solving partial differential equation in cardiac conduction”. In: *IEEE Transactions on Biomedical Engineering* 46.9 (1999), pp. 1166–1168. DOI: 10.1109/10.784149.

[136] Joakim Sundnes et al. “A Second-Order Algorithm for Solving Dynamic Cell Membrane Equations”. In: *IEEE Transactions on Biomedical Engineering* 56.10 (2009), pp. 2546–2548. DOI: 10.1109/TBME.2009.2014739.

[137] Mauro Perego and Alessandro Veneziani. "An efficient generalization of the rush-larsen method for solving electro-physiology membrane equations". In: *Electronic transactions on numerical analysis ETNA* 35 (Dec. 2009), pp. 234–256.

[138] V M Garcia et al. "An adaptive step size GPU ODE solver for simulating the electric cardiac activity". In: *2011 Computing in Cardiology.* 2011, pp. 233–236.

[139] V.M. Garcia-Molla et al. "Adaptive step ODE algorithms for the 3D simulation of electric heart activity with graphics processing units". In: *Computers in Biology and Medicine* 44 (2014), pp. 15–26. ISSN: 0010-4825. DOI: `https://doi.org/10.1016/j.compbiomed.2013.10.023`. URL: `https://www.sciencedirect.com/science/article/pii/S0010482513003107`.

[140] QIN SHENG. "Global error estimates for exponential splitting". In: *IMA Journal of Numerical Analysis* 14.1 (Jan. 1994), pp. 27–56. ISSN: 0272-4979. DOI: `10.1093/imanum/14.1.27`. eprint: `https://academic.oup.com/imajna/article-pdf/14/1/27/1998333/14-1-27.pdf`. URL: `https://doi.org/10.1093/imanum/14.1.27`.

[141] Gilbert Strang. "On the Construction and Comparison of Difference Schemes". In: *SIAM Journal on Numerical Analysis* 5.3 (1968), pp. 506–517. ISSN: 00361429. URL: `http://www.jstor.org/stable/2949700`.

[142] Konstantinos A. Mountris and Esther Pueyo. "A dual adaptive explicit time integration algorithm for efficiently solving the cardiac monodomain equation". In: *International Journal for Numerical Methods in Biomedical Engineering* 37.7 (2021), e3461. DOI: `https://doi.`

org/10.1002/cnm.3461. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.3461. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.3461.

[143] Andrew Sornborger. “Higher-order operator splitting methods for deterministic parabolic equations”. In: *Int. J. Comput. Math.* 84 (June 2007), pp. 887–893. DOI: 10.1080/00207160701458294.

[144] Jessica Cervi and Raymond J Spiteri. “High-order operator splitting for the bidomain and monodomain models”. en. In: *SIAM J. Sci. Comput.* 40.2 (Jan. 2018), A769–A786.

[145] Yuanfang Xie et al. “How does β-adrenergic signalling affect the transitions from ventricular tachycardia to ventricular fibrillation?” In: *Europace* 16.3 (2014), pp. 452–457.

[146] Hong Zhang et al. “Mechanisms of the acute ischemia-induced arrhythmogenesis–a simulation study.” In: *Mathematical biosciences* 203 1 (2006), pp. 1–18.

[147] E. Heidenreich et al. “Vulnerability to reentry in a 3D regionally ischemic ventricular slab preparation: A simulation study”. In: *2007 Computers in Cardiology*. 2007, pp. 321–324. DOI: 10.1109/CIC.2007.4745486.

[148] Emmanuel Drouin et al. “Electrophysiologic characteristics of cells spanning the left ventricular wall of human heart: Evidence for presence of M cells”. In: *Journal of the American College of Cardiology* 26.1 (1995), pp. 185–192. ISSN: 0735-1097. DOI: https://doi.org/10.1016/0735-1097(95)00167-X. URL: https://www.sciencedirect.com/science/article/pii/073510979500167X.

[149] Henggui Zhang et al. “Repolarisation and vulnerability to re-entry in the human heart with short QT syndrome arising from KCNQ1 mutation—A simulation study”. In: *Progress in Biophysics and Molecular Biology* 96.1 (2008). Cardiovascular Physiome, pp. 112–131. ISSN: 0079-6107. DOI: https://doi.org/10.1016/j.pbiomolbio.2007.07.020. URL: https://www.sciencedirect.com/science/article/pii/S0079610707000661.

[150] Daniel L. Weiss et al. “Modelling of short QT syndrome in a heterogeneous model of the human ventricular wall”. In: *EP Europace* 7.s2 (Jan. 2005), S105–S117. ISSN: 1099-5129. DOI: 10.1016/j.eupc.2005.04.008. eprint: https://academic.oup.com/europace/article-pdf/7/s2/S105/28936103/s105.pdf. URL: https://doi.org/10.1016/j.eupc.2005.04.008.

[151] Kazutaka Gima and Yoram Rudy. “Ionic Current Basis of Electrocardiographic Waveforms”. In: *Circulation Research* 90.8 (2002), pp. 889–896. DOI: 10.1161/01.RES.0000016960.61087.86. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.RES.0000016960.61087.86. URL: https://www.ahajournals.org/doi/abs/10.1161/01.RES.0000016960.61087.86.

[152] Henggui Zhang and Jules C. Hancox. “In silico study of action potential and QT interval shortening due to loss of inactivation of the cardiac rapid delayed rectifier potassium current”. In: *Biochemical and Biophysical Research Communications* 322.2 (2004). Calcium Signaling and Disease, pp. 693–699. ISSN: 0006-291X. DOI: https://doi.org/10.1016/j.bbrc.2004.07.176. URL: https://www.sciencedirect.com/science/article/pii/S0006291X04016201.

[153] Cunjin Luo et al. “Effects of quinidine on short QT syndrome variant 2 in the human ventricle: A modelling and simulation study”. In: *2017 Computing in Cardiology (CinC)*. 2017, pp. 1–4. DOI: 10.22489/CinC.2017.310-147.

[154] Peter Taggart et al. “Inhomogeneous Transmural Conduction During Early Ischaemia in Patients with Coronary Artery Disease”. In: *Journal of Molecular and Cellular Cardiology* 32.4 (2000), pp. 621–630. ISSN: 0022-2828. DOI: https://doi.org/10.1006/jmcc.2000.1105. URL: https://www.sciencedirect.com/science/article/pii/S0022282800911052.

[155] B Neta and T Fukushima. “Obrechkoff versus super-implicit methods for the solution of first- and second-order initial value problems”. In: *Computers & Mathematics with Applications* 45.1 (2003), pp. 383–390. ISSN: 0898-1221. DOI: https://doi.org/10.1016/S0898-1221(03)80024-X. URL: https://www.sciencedirect.com/science/article/pii/S089812210380024X.

[156] Alfio Quarteroni et al. “Integrated Heart—Coupling multiscale and multiphysics models for the simulation of the cardiac function”. In: *Computer Methods in Applied Mechanics and Engineering* 314 (2017). Special Issue on Biological Systems Dedicated to William S. Klug, pp. 345–407. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2016.05.031. URL: https://www.sciencedirect.com/science/article/pii/S0045782516304662.

[157] Nico Westerhof, Jan-Willem Lankhaar, and Berend E Westerhof. “The arterial Windkessel”. en. In: *Med. Biol. Eng. Comput.* 47.2 (Feb. 2009), pp. 131–141.

[158] Yasser Aboelkassem and Zdravko Virag. “A hybrid Windkessel-Womersley model for blood flow in arteries”. In: *Journal of Theoretical Biology* 462 (2019), pp. 499–513. ISSN: 0022-5193. DOI: `https://doi.org/10.1016/j.jtbi.2018.12.005`. URL: `https://www.sciencedirect.com/science/article/pii/S0022519318305952`.

[159] A Cappello, G Gnudi, and C Lamberti. “Identification of the three-element windkessel model incorporating a pressure-dependent compliance”. en. In: *Ann. Biomed. Eng.* 23.2 (Mar. 1995), pp. 164–177.