Applications of splitting Methods and exponential Integrators to an electro-chemical Heart Cell Model

Sjur Gjerald

Master of Science in Physics and Mathematics
Submission date: January 2007
Supervisor: Brynjulf Owren, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

The aim of the thesis is to investigate splitting methods and exponential integrators for solving a system of stiff ordinary differential equations originating in an electro-chemical heart cell model. The splitting is done by dividing the original system into a small stiff system and a larger non-stiff system.

Assignment given: 23. August 2006
Supervisor: Brynjulf Owren, MATH

Abstract

In this thesis we discuss how a system of ordinary differential equations (ODE) describing electro-chemical processes in a heart cell can be solved by numerical methods. The system is stiff, and explicit numerical solvers are therefore slow. In order to overcome the stiffness, the system is split into a stiff and a non-stiff part. The split system is solved by a Strang splitting method and an exponential integrator, based on a commutator free Lie group method. We outline a theory for estimating the computational cost of a numerical method. The solvers for the split system are compared to implicit solvers for the entire system. The conclusion is that it is possible to take out two components which are responsible for the stiffness of the original system, but that more research needs to be done in order to make efficient methods which take advantage of the fact.

Preface

I would like to thank my supervisor Brynjulf Owren for all hints and encouragements during the work on this report. Furthermore, I wish to thank Joakim Sundnes at the Simula Research Laboratory for providing me with a Matlab implementation of the Winslow heart cell system. I would also like to thank Martin Kaarby for help with the implementation of the ESDIRK solver.

Contents

1 Introduction

Heart disease is, according to the fact section of the World Heart Federation web site (`www.worldheart.org`), the world's number one killer, responsible for one in three deaths. A good understanding of the heart is essential in order to find a cure for heart diseases. The heart is a very complicated organ, consisting of an estimated 10^{10} cells. In order to fully understand how small scale processes affect the global behaviour of the heart, a mathematical model of the heart seems to be a good idea. One such heart model is being developed by researchers at the Simula Research Laboratory in Oslo. A brief introduction to the physiology of the heart and the mathematical heart model developed at the Simula Research Laboratory is given in Section 2. The property of the heart that we are interested in, is the fact that every heart cell is affected by the electrical activity of the heart. The electrical activity in a heart cell can be modelled by a system of ordinary differential equations (ODE). Most heart cell models cannot be evaluated analytically and we need to solve them by numerical methods. Some basic properties of numerical methods and the concept of stiffness are introduced in Section 3. Due to the large number of cells in the heart, it is important to be able to solve the ODE system efficiently. There are many different methods available. Due to possible limitations in computer memory, we will restrict ourselves to one-step solvers. Runge-Kutta and collocation methods are introduced in Section 3.3. In Section 3.4 we investigate several built-in solvers of Matlab. We will use the Matlab solvers for comparison with the numerical methods which we will implement for this thesis. The ODE system which we study in this paper is stiff. One of the aims of this thesis is to show that it is possible to split the ODE system into a stiff and a non-stiff part by removing two ODEs from the original system. A general theory of splitting methods is outlined in Section 3.5 and we decide to use a Strang splitting method. Another approach to solving the split system in this paper is by use of exponential integrators. These concepts will be introduced in section 3.6. Furthermore, we investigate automatic step-size selection for numerical methods in 3.7. In order to compare different solvers, we propose in Section 4 a cost theory for numerical one-step methods. Such a theory is important in order to have an a priori idea of the performance of different classes of solvers, regardless of the way they are implemented. Thereafter, we compare the solvers of this thesis with respect to the number of steps, the number of function evaluations, Jacobian matrix evaluations and linear algebra. Eventually, we conclude that there are good reasons theoretically for splitting the system, but there are challenges regarding implementation which are not yet solved.

2 A mathematical heart model

2.1 Physiology of the heart

Electrical activity in the heart is essential to the functioning of the heart and most grave heart problems are linked to anomalies in the electrical activity [17, p. 1]. This connection between electrical activity and heart problems can be measured by the use of the electrocardiogram (ECG). The ECG is a recording of electrical potential differences on the body surface, whose origins are in the heart. Although the electrical activity of the heart has received much attention, there are a few mechanisms which are not fully understood [17, p. 1]. One example is defibrillation, i.e. the application of an electrical shock to end a state of seemingly random contraction of the heart cells which prevents the heart from pumping blood [17, p. 14]. The electrical activity of the heart as it is globally observed, is the result of billions of small-scale processes in the cells. One way to investigate this connection between small-scale processes and the global behaviour of the heart is by mathematical modelling and computer simulations. This field of study is sometimes referred to as *integrative physiology* [17, p. 2], and

> The field of mathematical modelling in physiology is rapidly gaining popularity, and the potential both for increasing general knowledge and for clinical applications is huge

[17, p. 15]. In order to explain the ECG measurements, it is possible to view the electrical activity originating in the heart as a dipole in a conducting volume. However, in order to make a model which is able to provide more information than is known through the ECG measurements, it is necessary to take a closer look at what constitutes a heart. Because of the amount of cells in the heart, the mathematical heart model in [17] is based on a volume averaged approach called the *bidomain model*. In the bidomain model the heart tissue is divided into two domains, the *intracellular* domain and the *extracellular* domain. These are describing the interior and the exterior of the cells, respectively. Both domains are considered to be continuous and to fill the entire heart muscle. The reason why it is possible to treat the intracellular domain as being continuous, is that each cell interior is connected to that of its neighbours through *gap junctions* which are proteins to be described more closely below. In both domains there is an electrical potential which is averaged over a small volume. Because each small averaged volume generally contains both intracellular and extracellular space, every point of the heart is assumed to be included in both domains. The bidomain model, which is given by a system of partial differential equations (PDE) in [17, pp. 70,71], is as follows

$$\begin{aligned}
\frac{\partial s}{\partial t} &= F(s,v,t) && x \in H, && (1)\\
\nabla\cdot(M_i\nabla v) + \nabla\cdot(M_i\nabla u_e) &= \tfrac{\partial v}{\partial t} + I_{ion}(v,s) && x \in H, && (2)\\
\nabla\cdot(M_i\nabla v) + \nabla\cdot((M_i+M_e)\nabla u_e) &= 0 && x \in H, && (3)\\
\nabla\cdot(M_o\nabla u_o) &= 0 && x \in T, && (4)\\
u_e &= u_o && x \in \partial H, && (5)\\
n\cdot(M_i\nabla v + (M_i+M_e)\nabla u_e) &= n\cdot(M_o\nabla u_o) && x \in \partial H, && (6)\\
n\cdot(M_i\nabla v + M_i\nabla u_e) &= 0 && x \in \partial H, && (7)\\
n\cdot M_o\nabla u_o &= 0 && x \in \partial T. && (8)
\end{aligned}$$

Here equation (1) is a system of ODEs, which describe the electrophysical behaviour a heart cell, equations (2)-(3) describe the electrical signal propagation in the heart tissue. Furthermore, equation (4) describes the signal propagation in the torus, the body surrounding the heart. Equations (5)-(7) are describing the border conditions between the heart and the torus, while equation (8) describes the border condition on the surface of the entire body. The heart muscle and the torus are denoted by H and T, respectively. A schematic drawing of the heart and the torus and their outer borders ∂H and ∂T can be found in Figure 1. In equation (1), s denotes the variables describing the

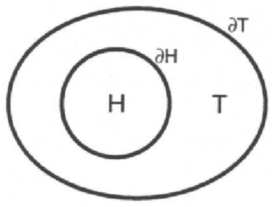


Figure 1: Schematic drawing of the heart (H) and the torus (T), found in [17, p. 24]

state of a cell, v is the electrical potential at the cell and t is time. The other variables are u_o, the potential in the torus, and u_e the extracellular potential. For a description of the other symbols in this PDE system we refer to [17].

Heart cells are so-called *excitable tissue*, that is, they have the ability to respond actively to electrical stimuli (see [17, pp. 24-25]). While at rest, the cells contain another internal ionic concentration than their surroundings. The electrical charge of the ions leads to a potential difference across the cell membrane, the so-called *transmembrane potential.* Under electrical stimuli, the transmembrane potential is changed. If the stimulus is small, the conductive properties of the cell membrane will not change, and the potential will quickly readjust to the resting value. When the transmembrane potential surpasses a certain limit, the conducting properties will change causing a rapid flux of positive ions into the cell. This leads to the *depolarisation* of the cell, which means that the transmembrane potential rise from a negative value of $-70-100mV$ to zero or more. After depolarisation, the cell repolarises to its negative resting state. The whole polarisation cycle is called an *action potential.* Heart cells stay at the depolarised value for a significant period of time, this phase is called the *plateau phase* [17, p. 10]. The action potential found by the Winslow model is calculated for 500 ms in Figure 2.

The action potential is to a large extent explained by the physiology of the membrane delimiting the cell. The cell membrane consists of a double layer of lipids which are characterised by a polar head attracted to water and a nonpolar tail ([17, pp. 36-38]). The tails are pointed towards the interior of the membrane, thus making it insulating, hampering the flow of ions between the interior and exterior of the cell. Embedded in the cell membrane are a number of large proteins called transport proteins, forming channels through the membrane. There are several different kinds of transport proteins. Some proteins are *pumps* which are able to pump ions against the concentration gradient, i.e. against the natural direction of flow, and the electrical field by adding energy stored in

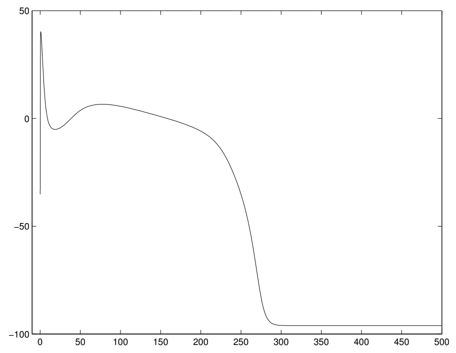


Figure 2: The action potential over 500 ms, given starting value -35 V

an energy-transfer molecule called ATP. The *exchanger* protein opposes the natural flux of ions by using the concentration gradient of another ion. In addition there are passive protein *channels* which allow the ions to flow according to the electric field and their concentration gradients. Most of the channels are selective in the choice of which ions are allowed to pass. Moreover, the channels have the ability to open and close according to changes in electrical field and ionic concentration. As a result, the equilibrium states of the cell and its surroundings are different, thus creating the transmembrane potential. The equilibrium state is reached when the diffusive (chemical) flux of ions is equal and working against the electrically driven flux.

There are several different mathematical models for modelling the action potential. As written above, the action potential depends on the total ionic current across the cell membrane. The first model of ionic currents was developed in 1952, for the ionic current in a squid nerve cell [17, p. 44]. One of the most accurate and complex of the cell models, consisting of 33 differential variables was developed by Winslow et al. in 1999 [18]. We will refer to this system as the *Winslow system*. Due to physiological considerations, the Matlab implementation of the Winslow system, which we make use of in this paper has been reduced to 31 variables. The Winslow system corresponds to equation (1) and also returns the ionic current I_{ion} from (2). In this thesis, we solve the Winslow system decoupled from the bidomain PDE system and thus

$$y' = \begin{bmatrix} \frac{\partial s}{\partial t} \\ \frac{\partial v}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}s}{\mathrm{d}t} \\ \frac{\mathrm{d}v}{\mathrm{d}t} \end{bmatrix} = \begin{bmatrix} F(s,v,t) \\ -I_{ion}(v,s) \end{bmatrix} = f(s,v,t) = f(y), \qquad y = \begin{bmatrix} s \\ v \end{bmatrix}. \tag{9}$$

Starting values $y_0 = y(0)$ for the Winslow system can be found in [18, p. 585]. We note that the Winslow system is autonomous, not depending explicitly on t. The 33 differential equations describing the Winslow system are included in Appendix A.

2.2 Numerical considerations for the bidomain model

The discretisation in space of the bidomain PDEs results in a finite element grid with an ODE system at each node. As a result

> Realistic computations require up to several millions of grid nodes, and each ODE system may consist of 30 or more ODEs. In this context, the memory requirement of the multi-step methods becomes very significant

[17, p. 156]. Due to the problem of potential memory shortage, the focus of study has been on one-step solvers. When an ODE system is coupled with the bidomain PDE system, and the entire system is solved by the operator splitting method in [17], there are at least two properties which a solver has to fulfil. Firstly, the time discretisation of the PDE system forces a limitation to the length of the time steps for solving the ODE system, this length is typically 0.125 ms [17, p. 172]. Secondly, the accuracy of the operator splitting procedure for solving the PDE system is limited to two, and

> It is therefore not very useful to apply ODE solvers with high order of accuracy, because the global accuracy will still be limited by the splitting error

[17, p. 172]. We note that the above mentioned properties for the numerical ODE solver are only required when coupling the ODE system with the bidomain PDE system. They are not necessary when solving the ODE system decoupled from the PDE system.

There exist several cell models of ODEs, each of different complexity and different performance in terms of e.g. stiffness properties. We have chosen the Winslow system because it is stiff and thus computationally demanding.

3 An introduction to numerical methods

3.1 Stability

Stability is a measure of the extent to which the numerical method shows the same perturbation sensitivity as the underlying differential equation (see [3, p. 37]). In order to do stability analysis, we start by reducing the Winslow system to a simpler form. First, we assume that it is possible to linearise the Winslow system to get a system

$$y'(t) = J(t)y(t)$$

where $J(t)$ is the symbol of the *Jacobian matrix* of the Winslow system. The Jacobian matrix of a n-dimensional system of equations for which $y = (y_1, \cdots, y_n)$ and $f(y) = (f_1, \cdots, f_n)$ is

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \ddots & \cdots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} \end{pmatrix}.$$

Assuming that the Jacobian matrix is constant over an interval to be considered, we arrive at the system

$$y' = Jy, \qquad y_0 = y(0). \tag{10}$$

We assume that the Jacobian matrix is diagonalisable, i.e. $X^{-1}JX = \Lambda$, where Λ is a diagonal matrix with the eigenvalues of J at the diagonal, and matrix X has the eigenvectors of J as its column vectors. Given an initial condition of (10) as a linear combination of the eigenvectors of J, all solutions can be expressed as a linear combination of these eigenvectors. Since $Jx = \lambda x$, where λ denotes the eigenvalues and x the eigenvectors, the system (10) is equivalent to the system

$$y' = \lambda y, \qquad y_0 = y(0), \tag{11}$$

called the *Dahlquist test equation.* For a discussion of the validity of reducing the general system (9) to this simple test equations for stability considerations, see [8].

In Section 3.3, we will see that the explicit Runge-Kutta solvers are not suitable for the Winslow system. In order to explain this fact, we now introduce the rational function $R(z)$ which is associated to all Runge-Kutta methods. The $R(z)$ is called the *stability function* of the method. For a Runge-Kutta method that is used on the initial value problem (11), the solution is

$$y_1 = R(z)y_0, \qquad z = h\lambda.$$

The *stability region* or stability domain of a Runge-Kutta method is

$$S = \{z \in \mathbb{C}; |R(z)| \leq 1\}. \tag{12}$$

Whenever $h\lambda \in S$ and $|R(z)| < 1$, the numerical solution y_n of (11) will tend to 0 as $n \to \infty$. If $|R(z)| = 1$, the solution is constant. If a numerical method is stable on the entire left half-plane of the complex plane $\mathbb{C}^-$, i.e $\mathbb{C}^- \subset S$, it is said to be A-stable. Since an A-stable method converges in the same domain as the system (11), an A-stable method is preferable for general numerical integration. A RK-method with stability function of form

$$R(z) = \frac{P(z)}{Q(z)}$$

where $P(z)$ and $Q(z)$ are polynomials, is A-stable only if the degree of $P(z)$ is inferior to that of $Q(z)$. It can be shown that the difference of order must not be greater than two [7, p. 58]. For this reason, explicit Runge-Kutta methods are not A-stable.

For stiff systems, the L-stability of the solver could be important. An L-stable method is A-stable and in addition

$$\lim_{|\Re(z)| \to \infty} R(z) = 0,$$

where $\Re(z)$ is the real part of $z = h\lambda$. The L-stability is desirable because the stability polynomial approaches zero as the real solution of (11) approaches zero. It seems that L-stability is not important for the Winslow system. For instance, the performances of the Matlab solvers ODE23T and ODE23TB are quite similar, and one of the most important difference between the methods is that the former lacks L-stability (see Section 3.4).

3.2 Stiffness

When the step size required by a solver for returning accurate output for some ODE system is governed by stability concerns rather than by accuracy requirements, the system is said to be *stiff*. Typically

> Stiff equations are problems for which explicit methods don't work

[7, p. 2]. A stiff system is also characterised by the property that

> the solution to be computed is slowly varying but that perturbations exist which are rapidly damped

[3, p. 5]. The Winslow system is stiff in both of these respects.

A discussion of how the eigenvalues of the Jacobian matrix of an ODE system indicate the stiffness of the system is made in [3, p. 9f]. It states that a stiff system is characterised by the fact that there exists at least one big negative real eigenvalue for the Jacobian matrix of the system. Thus, for the discussion of the stiffness of the Winslow system, we have found the negative eigenvalues of the Jacobian matrix, and conclude that the system is stiff for large negative eigenvalues. In addition to the presence of a large negative eigenvalue, the system is stiff if there is an eigenvalue of small magnitude and no large positive real eigenvalues. Moreover, if the eigenvalue is a complex number, the imaginary part must not be large unless at the same time the real part is large and negative. The Winslow system is stiff, but does not always satisfy all of these requirements. Since the eigenvalue of largest magnitude for the Winslow system is real and negative, we will focus on the real negative eigenvalues for measuring stiffness.

Other ways of measuring stiffness which do not include the eigenvalues of the Jacobian matrix have been proposed, see e.g. [8], but we will only consider the eigenvalue approach.

The solution of the ODE system (11),

$$y = y_0 e^{\lambda(t-t_0)},$$

will only converge for negative λ as $t \to \infty$. When the eigenvalues of the Jacobian matrix of a system (9) are large and negative, the step size h of an explicit Runge-Kutta method has to be small in order to keep the product $z = h\lambda$ within the stability domain. For implicit Runge-Kutta methods and generally for A-stable methods, there are no restrictions to the range of h. Due to this fact, explicit methods generally require smaller steps than implicit methods for stiff problems.

In Section 3.3, we will explain that implicit methods require the use of non-linear iterations which may be costly. Thus, for moderately stiff systems, explicit methods may still be faster than implicit methods.

Another approach to solving stiff problems is by the use of explicit exponential methods. The drawback of exponential methods is the cost of calculating the exponential of a matrix. For a non-linear ODE system like the Winslow system, the matrix of which we find the exponential is typically a Jacobian matrix of the entire system or an approximation to it (see Section 3.6).

In order to estimate the eigenvalues of the Jacobian matrix of the Winslow system, we will make use of a numerical approximation to its Jacobian matrix. A numerical approximation is typically of the form

$$\frac{\partial f(y_0)}{\partial y_j} = \frac{f(y_0 + he_j) - f(y_0)}{h}.$$

Here y_0 is denoting the initial value at the start of the step, y_j is a component of y_0, and $e_j \in \mathbb{R}^n$ is a vector which is zero everywhere except for the element j where it is one. In this paper we have made use of the built-in Matlab function NUMJAC for numerical calculations of the Jacobian matrix.

3.3 Classical numerical one-step methods

3.3.1 Runge-Kutta methods

The Runge-Kutta methods are one-step methods which we use several times throughout this thesis. A Runge-Kutta method is characterised by the formulas

$$Y_i = y_0 + h\sum_{j=1}^{s} a_{ij} f(t_0 + c_j h, Y_j) \tag{13}$$

$$y_1 = y_0 + h\sum_{j=1}^{s} b_j f(t_0 + c_j h, Y_j) \tag{14}$$

for $i = 1, \cdots, s$. Here Y_i is an intermediate approximation to the solution at $t_0 + c_i h$, while y_0 is the initial value at time t_0 and y_1 is the numerical approximation to the solution at time and t_1. The RK-methods can also be written in a slightly different form by using the expression $k_i = f(t_0 + c_i h, Y_i)$ instead of Y_i, i.e. as

$$\begin{array}{ll} k_i = f\left(y_0 + h\sum_{j=1}^{s} a_{ij} k_j\right) & i = 1, \cdots, s \\ y_1 = y_0 + h\sum_{j=1}^{s} b_j k_j & . \end{array} \tag{15}$$

The coefficients a_{ij},b_j and c_j are scalars specific for each method and summarised in a Butcher tableau, as seen in Table 1.

It is customary to let the coefficients of Table 1 be elements of vectors b, c and a matrix A. If the elements $a_{ij} = 0$ for $i \geq j$, then the method is explicit, otherwise it is implicit. If a Runge-Kutta method has equal non-zero elements on the diagonal of the A-matrix, except for the first step which is explicit, it is said to be a *Singly Diagonally Implicit Runge-Kutta* method with explicit first step (ESDIRK). We will denote the explicit Runge-Kutta method by ERK and the implicit Runge-Kutta methods by IRK.

An embedded method is an auxiliary method with different order than the original method, using the same function evaluations or almost the same function evaluations as

c_1	a_{11}	$\cdots$	a_{1s}
c_2	a_{21}	$\cdots$	a_{2s}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	$\cdots$	a_{ss}
	b_1	$\cdots$	b_s

Table 1: Butcher tableau for a general implicit RK-method

0	0	0	0
$\frac{1}{2}$	1	0	0
1	$\frac{1}{4}$	$\frac{1}{4}$	0
	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{2}{3}$
	$\frac{-1}{6}$	$\frac{-1}{6}$	$\frac{4}{3}$

Table 2: Butcher tableau for the method ODE23

the original method. In this way the embedded method provides an almost free local error estimate. The local error estimate is generally

$$\tilde{r}_{loc} = h \sum_{i=1}^{s} (\hat{b}_i - b_i) k_i,$$

where $\hat{b}_i$ and b_i are the b-coefficients from the embedded and the original method respectively. The order of a method consisting of such an embedded pair is written $m(n)$, where m and n are orders of the original method and the embedded method, respectively.

In this paper we have used an embedded ERK pair of order 2(3) as one of the submethods in a splitting method. The coefficients of the method are written in Table 2. This RK-pair is identical to the pair which is implemented in the explicit Matlab solver ODE23. The stability domain of the ODE23 method is drawn in Figure 3. The smallest domain is for the order two method and the largest one is the order three domain.

The ESDIRK32 method pair, which we have used in the splitting method is given in Table 3. The method is described in [11, p. 497]. The parameter γ in this method is chosen for reasons of stability and depends on whether $y_1 = Y_3$ or $y_1 = Y_4$. We have chosen the order two method for advancing the step and thus $y_1 = Y_4$, $\hat{y}_1 = Y_3$ and $\gamma = 0.4358665215$.

The implementation of IRK methods is more complicated than the implementation of ERK methods. We will now describe the procedures for implementing IRK methods which we have used for this paper. A more thorough survey of the implementation techniques for IRK methods can be found in [7, pp. 118-127]. First, in order to reduce

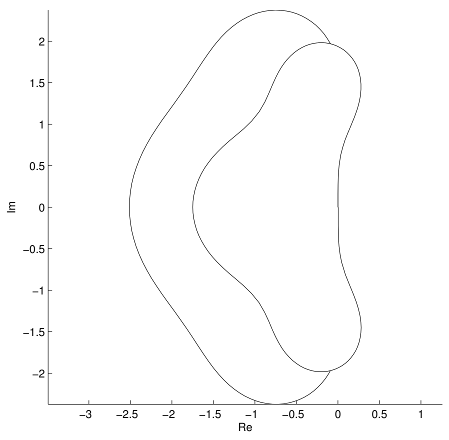


Figure 3: Stability domains of the order two ERK method (smallest domain) and a order three embedded method of ODE23

$$
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
2\gamma & \gamma & \gamma & 0 & 0 \\
1 & \frac{-\gamma^2+g\gamma-1}{4\gamma} & \frac{-2\gamma+1}{4\gamma} & \gamma & \\
1 & \frac{6\gamma-1}{12\gamma} & \frac{-1}{12(2\gamma-1)\gamma} & \frac{-6\gamma^2+6\gamma-1}{3(2\gamma-1)} & \gamma
\end{array}
$$

Table 3: Butcher tableau for the method ESDIRK32

the influence of round-off error, we transform the equations in (13) into

$$u_i = h\sum_{j=1}^{s} a_{ij} f(x_0 + c_j h, y_0 + u_j), \qquad \text{for} \qquad i = 1, \cdots, s \tag{16}$$

where

$$u_i = Y_i - y_0.$$

In order to solve the implicit equations (16), it is necessary to do non-linear iterations, and Newton iterations are recommended [12]. A Newton iteration is

$$u_i^{[j+1]} = u_i^j - \frac{F(u_i^j)}{F'(u_i^j)}$$

where the j denotes the number of iterations for some function $F(u_i)$. The function $F(u_i)$ is generally

$$F(u_i) = u_i - h\sum_{j=1}^{s} a_{ij} f(x_0 + c_j h, y_0 + u_j), \qquad \text{for} \qquad i = 1, \cdots, s.$$

For the implementation of regular Newton iterations, we use the system

$$\begin{aligned} J_{F_i}(u_i^j)\Delta u_i^j &= -F_i(u_i^j) \\ u_i^{j+1} &= u_i^j + \Delta u^j. \end{aligned}$$

Here $J_{F_i}(u_i^j) = F_i'(u_i^j)$ is the Jacobian matrix of $F_i(u_i^j)$. In order to reduce the cost of computing the Jacobian matrix of the system, it is possible to use simplified Newton iterations. A simplified Newton iteration uses the Jacobian matrix

$$J_{F_i}(y_0) = I - ha_{ij}J,$$

where I is the identity matrix, h is the step size, a_{ij} an element for a method, and J is the Jacobian matrix of an ODE system in state y_0, i.e. $J = f'(y_0)$. In other words, simplified Newton iterations reduce the number of calculations of the Jacobian matrix of $f(y)$. How to chose starting values for the Newton iteration is described in [7, p. 120].

3.3.2 Collocation methods

In this paper, we will use a collocation method called RadauIIA on the Winslow system. Collocation methods are methods for which the solution of the system (9) is approximated by a polynomial that interpolates the solution for s different points of the solution. Given distinct real numbers $c_1, \cdots, c_s$, the collocation polynomial $u(t)$ is of degree s and such that

$$\begin{aligned} u(t_0) &= y_0 \\ \dot{u}(t_0 + c_i h) &= f(t_0 + c_i h, u(t_0 + c_i h)) \\ y_1 &= u(t_0 + h) \end{aligned} \tag{17}$$

where $i = 1, \cdots, s$. It has been shown that a collocation method with s interpolation steps is equivalent to a s stage RK-method with coefficients

$$\begin{aligned} a_{ij} &= \textstyle\int_0^{c_i} l_j(\tau)\mathrm{d}\tau \\ b_i &= \textstyle\int_0^1 l_i(\tau)\mathrm{d}\tau \\ l_i(\tau) &= \textstyle\prod_{e\neq i}^{s} \frac{\tau - c_l}{c_i - c_e}, \end{aligned} \tag{18}$$

where $l_i(\tau)$ is called a Lagrange polynomial and $i, j = 1, \cdots, s$ (see e.g. [5, p.27]). After finding a collocation method, it is in other words possible to write it in a Butcher tableau. In order to approximate the integral of a collocation polynomial we can use a quadrature formula

$$\int_0^1 g(t_0 + \tau h)\mathrm{d}\tau \approx \sum_{i=1}^{s} b_i g(t_0 + c_i h).$$

The quadrature formula has order p when it is exact for polynomials of order less than p. If the coefficients of the quadrature formula satisfy

$$\frac{1}{k} = \sum_{j=1}^{s} b_j c_j^{k-1},$$

for $k = 1, \cdots, p$ and $p \geq s$, then the collocation method is of order p. Radau methods are collocation methods of polynomial order $2s-1$. The Radau method has either $c_1 = 0$ or $c_s = 1$ in which case we get a RadauIIA method. The RadauIIA method is described in [7, pp. 72-74] and a Fortran implementation of it called RADAU5 is described in [7, pp. 565-574]. For this paper, we have used a Matlab-version of the RADAU5 code. The Matlab code can be found in [4]. The coefficients of the method are written in Table 4.

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

Table 4: Butcher tableau for the RadauIIA formula with s=3 and order 5

3.4 Matlab ODE-solvers

We will compare built-in Matlab ODE solvers to the solvers implemented for this paper, and we will use one of them (ODE15S) to estimate the exact solution of the Winslow system. The descriptions of the Matlab ODE solvers were found in [16] and [9].

3.4.1 ODE15S

The Matlab ODE-solver ODE15S is a multi-step solver based on an improved Backward Difference Formula (BDF) called the Numerical Differentiation Formula (NDF). For a description of this method we refer to [16]. An introduction to multi-step solvers can be found in [7]. We will see in Section 5 that this method is very efficient for the Winslow system when decoupled from the bidomain model. However, as mentioned in Section 2, due to of memory concerns for the bidomain model, we are not going to consider multi-step solvers. Because of the good properties of the solver, we have used it for calculating an estimate to the exact solution of the Winslow system.

3.4.2 ODE23S

The Matlab ODE-solver ODE23S is based on a modified Rosenbrock formula of second order. The Rosenbrock method is a special case of a diagonally implicit Runge-Kutta method. For an description of how to derive the general Rosenbrock formulas we refer to [7, p. 102-104].

The Rosenbrock method in Matlab is modified, which means that is uses the approximation

$$J = \frac{\partial f(y_0)}{\partial y}(t_0.y_0) + hB + O(h^2).$$

to the Jacobian matrix. According to [16, p. 6] it is necessary to use this approximation in order to be able to estimate the local error of the Rosenbrock method without introducing an extra step.

The modified three stage Rosenbrock method implemented in Matlab is

$$\begin{array}{rcl} f_0 & = & f(t_0, y_0) \\ k_1 & = & W^{-1}(f_0 + hdT) \\ f_1 & = & f(t_0 + 0.5h, y_0 + 0.5hk_1) \\ k_2 & = & W^{-1}(f_1 - k_1) + k_1 \\ y_1 & = & y_0 + hk_2 \\ f_2 & = & f(t_1, y_1) \\ k_3 & = & W^{-1}\left[f_2 - e_{32}(k_2 - f_1) - 2(k_1 - f_0) + hdT\right] \\ error & \approx & \frac{h}{6}(k_1 - 2k_2 + k_3). \end{array} \tag{19}$$

Here $W = I - hdJ$ with $d = 1/(2+\sqrt{2})$ and $J \approx \frac{\partial f}{\partial y}(t_0, y_0)$, $T \approx \frac{\partial f}{\partial t}(t_0, y_0)$ and $e_{32} = 6 + \sqrt{2}$.

The method uses the result y_1 for propagating the solution, and if the step is a success, the f_2 of the current step will be the f_0 of the next one. For this reason we will not need any additional function evaluations. This property is called first-same-as-last (FSAL), which means that the first stage of a step is the same as the last one from the end of the previous step.

In the implementation of this method, the Jacobian matrix is recalculated at every step. This is probably one of the reasons why this method is not among the most efficient Matlab solvers.

3.4.3 ODE23TB

The Matlab ODE-solver ODE23TB is based on a numerical method which can be seen as an implicit Runge-Kutta pair of order 2(3), called TR-BDF2. The method was developed in the context of device simulation and later implemented in the Matlab ODE-suite by Hosea and Shampine [9].

0	0	0	0
γ	d	d	0
1	w	w	d
	w	w	d
	$(1-w)/3$	$(3w+1)/3$	$d/3$

Table 5: Butcher tableau for the TR-BDF2 method, where $\gamma = 2 - \sqrt{2}$, $d = \frac{\gamma}{2}$ and $w = \frac{\sqrt{2}}{4}$.

The method is constructed by first considering the trapezoidal rule

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f(x_{i+1}, y_{i+1})\right). \tag{20}$$

This method is not strongly stable and thus not efficient for very stiff problems. In order to get a method more suitable for stiff problems, it is possible to make a second step taken by the multi-step method BDF2. The two steps have different step-sizes. The way it is implemented in the TR-BDF2 method, we end up with a method with two internal steps and no memory. For this reason it is possible to regard the method as an one-step method which, with an embedded error estimate, turns out to be a Singly Diagonally Implicit Runge-Kutta (SDIRK) pair. The method has been summarised in Table 5.

This method has several nice properties. It is FSAL, and because the elements on the diagonal are similar, the same simplified Newton matrix can be used to evaluate all implicit stages. It is also L-stable.

3.4.4 ODE23T

The ODE-solver ODE23T is based on the trapezoidal rule. This method is called the TRX2 formula and was implemented in Matlab by Hosea and Shampine [9]. The difference between ODE23TB and ODE23T is that another step with the trapezoidal rule replaces BDF2. The Butcher tableau of this method is given in Table 6. One main difference between TRX2 and TR-BDF2 is that the former lacks L-stability [9, p. 25].

3.5 Splitting methods

The basic idea of a splitting method is to divide the vector field of the system one wants to solve into simpler pieces, which are then treated separately. Considering the Winslow

0	0	0	0
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0
1	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

Table 6: Butcher tableau for the TRX2 method

system (9), a corresponding split system of a non-stiff and a stiff vector field is

$$y' = f_{ns}(y) + f_s(y). \tag{21}$$

The exact flows, i.e. the solutions given certain initial values, of the ODE systems

$$y' = f_{ns}(y) \tag{22}$$

and

$$y' = f_s(y) \tag{23}$$

are $\phi_h^{[ns]}$ and $\phi_h^{[s]}$, respectively. If we assume that the exact flows can be calculated explicitly, it is possible to solve the system (23) with the initial value y_0 to obtain a value $y_{\frac{1}{2}}$. From this new value, we solve system (22) to get y_1. A system of this kind can be denoted by the Lie-Trotter splitting formula

$$\Phi_h = \phi_h^{[ns]} \circ \phi_h^{[s]}. \tag{24}$$

It is also possible to start by evaluating the non-stiff part. Then we get the Lie-Trotter splitting formula

$$\Phi_h^* = \phi_h^{[s]} \circ \phi_h^{[ns]},$$

which is called the adjoint method of (24). By Taylor-series expansions it can be shown that for the system (21), the Lie-Trotter splitting formula is a method of order one, i.e.

$$\Phi_h(y_0) = \phi_h(y_0) + \mathcal{O}(h^2),$$

where $\phi_h(y_0)$ is the exact flow of (21). The splitting method which we have implemented for solving the Winslow system is often called Strang splitting. The Strang splitting is of form

$$\Phi_h^{[S]} = \phi_{\frac{1}{2}h}^{[s]} \circ \phi_h^{[ns]} \circ \phi_{\frac{1}{2}h}^{[s]}. \tag{25}$$

The Strang splitting could also be written as

$$\Phi_h^{[S]} = \phi_{\frac{1}{2}h}^{[ns]} \circ \phi_h^{[s]} \circ \phi_{\frac{1}{2}h}^{[ns]}.$$

It has been recommended (see [10]) that when solving a stiff system by the Strang splitting method, the Strang splitting in (25) should be used because it gives the smallest local error for sufficiently large step-sizes. Moreover, we chose this splitting method because we want to reduce the number of function evaluations of the non-stiff system, which we assume is much more expensive to calculate than the stiff system.

It is possible to view the Strang splitting method as being a composition of a Lie-Trotter method and its adjoint method with halved step-sizes. An introduction to composition methods can be found in [5, pp. 39-41]. The general composition method of methods with their adjoint method is

$$\Psi_h = \Phi_{\alpha_s h} \circ \Phi^*_{\beta_s h} \circ \cdots \circ \Phi^*_{\beta_2 h} \circ \Phi_{\alpha_1 h} \circ \Phi^*_{\beta_1 h},$$

and the order conditions for a method of order $p+1$ is

$$\begin{array}{rcl} \beta_1 + \alpha_1 + \beta_2 + \cdots + \beta_s + \alpha_s & = & 1 \\ (-1)^p \beta_1^{p+1} + \alpha_1^{p+1} + (-1)^p \beta_2^{p+1} + \cdots + (-1)^p \beta_s^{p+1} + \alpha_s^{p+1} & = & 0. \end{array} \tag{26}$$

One solution of equation (26) is $\alpha_1 = \beta_1 = \frac{1}{2}$ (also called the consistency requirement) and $p = s = 1$. This means that two consistent one-step methods of order 1 can be composed into a second-order method. In other words, since Strang splitting is a composition of the Lie-Trotter method and its adjoint with halved step sizes, the Strang splitting method is of second order.

A general splitting method for a vector field split into two parts, is a method on the form

$$\Psi_h = \phi^{[s]}_{b_m h} \circ \phi^{[ns]}_{a_m h} \circ \phi^{[s]}_{b_{m-1} h} \circ \cdots \circ \phi^{[ns]}_{a_2 h} \circ \phi^{[s]}_{b_2 h} \circ \phi^{[ns]}_{a_1 h}.$$

The splitting of a system is not limited to two vector fields, but in our case we are only splitting the system according to stiffness, and two fields is a natural choice (i.e. stiff/non-stiff). There exist several splitting methods, some of which there are references to in [5, p. 43]. In this thesis we will only consider Strang splitting for several reasons. For one reason, the discretisation error in the operator splitting algorithm for the PDE system is of order two [17, p. 172]. Thus, it possibly suffices to use a numerical method for the ODE system with order two. Another reason is that we seek to minimise the number of function evaluations, and a splitting method of higher order than two is likely to add function evaluations.

It is possible to make splitting methods where one flow is computed exactly (see e.g. [5, p. 44]). For the Winslow system written on form (27) in Section 3.6, it is possible to solve the first linear part exactly over one step as $y_1 = e^L y_0$. This could be an efficient approach to solving the system since the most stiff part is supposed to be in the linear part. We will leave this idea for future work.

3.6 Exponential integrators

3.6.1 Background

Exponential integrators are numerical methods which involve an exponential function of the Jacobian or an approximation to it [13, p. 4]. One advantage of the exponential methods is that they usually have good stability properties, which make them suitable for solving stiff problems. Here we are going to follow [13] and apply exponential integrators to the Winslow system written on form

$$y'(t) = f(y(t)) = Ly(t) + N(y(t)), \qquad y(t_0) = y_0. \tag{27}$$

Here L is a constant matrix which is supposed to carry the stiffness of the system. Since the Winslow system does not have an independent linear part, we will transform equation (9) onto the form in equation (27) by setting

$$N(y) = f(y) - Ly,$$

where $y = y(t)$. The L is chosen such that Ly becomes the stiff part of the system, while hopefully making $N(y)$ non-stiff. The problem is that is not evident that the function $N(y)$ is in fact non-stiff. We will see that for the Winslow system, $N(y)$ is still quite stiff. In fact there are large positive and negative values in both the real part and the imaginary part of the eigenvalues. The matrix L depends on y. Thus we have to recalculate L several times during an integration, and freezing it during each step of the numerical solver. Different ways of constructing L are discussed in Section 5.

The general form of a one-step exponential linear methods is

$$\begin{array}{rcl} Y_i &=& \sum_{j=1}^{s} a_{ij}(hL)hN(Y_j) + u_i(hL)y_0 \\ y_1 &=& \sum_{i=1} b_i(hL)hN(Y_i) + v(hL)y_0, \end{array} \tag{28}$$

where h is the step-size [13, p. 5]. The parameters a_{ij},b_{ij}, u_i and v are the coefficients of the method and are functions of the exponential and related functions. On matrix form the expression (28) is

$$\begin{array}{rcl} Y &=& A(hL)hN(Y) + U(hL)y_0 \\ y_1 &=& B(hL)hN(Y) + v(hL)y_0, \end{array} \tag{29}$$

where

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_s \end{bmatrix} \qquad N(Y) = \begin{bmatrix} N(Y_1) \\ N(Y_2) \\ \vdots \\ N(Y_s) \end{bmatrix}.$$

As in [13, p. 6], the coefficient matrices can be represented in a Butcher style table

$$M(hL) = \left[\begin{array}{c|c} A(hL) & U(hL) \\ \hline B(hL) & v(hL) \end{array}\right].$$

There are several classes of methods of exponential integrators. In the next sections we will introduce the exponential integrators which we will use in this paper. Other methods can be found in the article [13].

3.6.2 Exponential integrators of order one

Several exponential integrators of order one are presented in [13, pp. 2-4]. In this thesis, we will use two of these methods. We start by linearising an ODE system (9). This gives

$$y'(t) = f(y_0) + f'(y_0)(y - y_0),$$

where $f'(y)$ is the Jacobian matrix of $f(y)$. The exact solution of this problem is

$$y_1 = y_0 + h\phi_1(hf'(y_0))f(y_0), \tag{30}$$

where ϕ_1 is a function defined as

$$\phi_1(z) = \frac{e^z - 1}{z}.$$

The ϕ_k-functions play an important part in the construction of exponential integrators. The method defined by (30) is of order two and is called the *exponential Euler method.* Following [13] we approximate the Jacobian matrix in equation (30) by L. The exponential Euler method with approximated Jacobian matrix is commonly called the *Exponential Time Differencing (ETD) Euler* method and denoted by

$$y_1 = y_0 + h\phi_1(hL)(Ly_0 + N(y_0)) = e^{hL}y_0 + h\phi_1(hL)(N(y_0)).$$

The method is also known as the *Nørsett-Euler* method. This method requires the computation of one matrix exponential and one ϕ_1-function and is of order one. We have used this method as a step-size corrector. An alternative method of order one is the *Lawson-Euler* method defined by

$$y_1 = e^{hL}y_0 + e^{hL}hN(y_0).$$

We also tried to use this method as a step-size corrector.

3.6.3 Commutator Free methods

One class of exponential integrators is the Lie group methods. The theory behind the Lie group methods tends to be rather abstract and we refer to e.g. [5, pp. 110-128] and references therein for an introduction to the subject. Lie Group methods include methods such as the Runge-Kutta Munthe-Kaas (RKMK) methods which are not suitable for stiff problems [13, p. 21]. In [2] a class of Lie group methods called Commutator Free (CF) methods were developed. CF-methods are suitable for solving the Winslow system because they are applicable to problems where L represents the stiff part and they require relatively few exponential function evaluations. The general Commutator Free Lie group method is

$$\begin{aligned}
Y_i &= \mathrm{Exp}\left(h\textstyle\sum_{j=1}^{s}\alpha_{ij}^{K}(L,N_j)\right)\cdots\mathrm{Exp}\left(h\textstyle\sum_{j=1}^{s}\alpha_{ij}^{1}(L,N_j)\right)\cdot y_0\\
N_i &= N(Y_i)\\
y_1 &= \mathrm{Exp}\left(h\textstyle\sum_{j=1}^{s}\beta_{j}^{K}(L,N_j)\right)\cdots\mathrm{Exp}\left(h\textstyle\sum_{j=1}^{s}\beta_{j}^{1}(L,N_j)\right)\cdot y_0
\end{aligned}$$

where $i \in \{1, 2, \cdots, s\}$, K counts the number of Exp-evaluations at each stage and α_{ij}^k and β_j^k are parameters of the method. The parameters are determined by an order theory which is outlined generally and explicitly for orders up to order four in [15]. A CF-method is explicit if $\alpha_{ij}^k = 0$ for $i \leq j$, else it is implicit. An explicit CF-method of order four is implemented in the Expint package (see [1]) for Matlab. The method is

$$\begin{array}{cccc|c} 0 & 0 & 0 & 0 & I \\ \frac{1}{2}\phi_{1,2} & 0 & 0 & 0 & e^{\frac{1}{2}hL} \\ 0 & \frac{1}{2}\phi_{1,2} & 0 & 0 & e^{\frac{1}{2}hL} \\ \frac{1}{2}(e^{\frac{1}{2}hL} - I)\phi_{1,2} & 0 & \phi_{1,2} & 0 & e^{\frac{1}{2}hL} \\ \hline \frac{1}{2}\phi_1 - \frac{1}{3}\phi_{1,2} & \frac{1}{3}\phi_1 & \frac{1}{3}\phi_1 & -\frac{1}{6}\phi_1 + \frac{1}{3}\phi_{1,2} & e^{\frac{1}{2}hL} \end{array},$$

where $\phi_{1,2} = \phi_1(c_2 hL)$.

3.6.4 Implementation issues

In order to construct an exponential integrator, it is necessary to compute the matrix exponential and functions thereof, notably the ϕ_k-functions. Different procedures for finding the matrix exponential are discussed in [14]. For this thesis, we used the matrix exponential function of Matlab and the ϕ_k-functions of the Expint package [1].

It is useful to note that the stiff system is a vector field with just a few non zero components. The non-vanishing dimensions depend on several variables. This means that the Jacobian matrix J only has a few non zero rows and a number of non-zero columns which is inferior to the size of the system. It is possible to take advantage of this fact by expanding the matrix exponential in the series

$$e^J = \sum_{k=0}^{\infty} \frac{J^k}{k!}.$$

Perturbing the rows of the Jacobian to have the non-zero rows as the first rows of the matrix, we find that the Jacobian matrix can be written as

$$J = \left[\begin{array}{c|c} A & B \\ \hline 0 & 0 \end{array}\right],$$

where A is a square matrix and B is a rectangular matrix. The matrix exponential is thus

$$e^J = I + \left[\begin{array}{c|c} \sum_{k=1}^{\infty} \frac{A^k}{k!} & \sum_{k=0}^{\infty} \frac{A^k}{(k+1)!} B \\ \hline 0 & 0 \end{array}\right] = I + \left[\begin{array}{c|c} e^A - I_A & A^{-1}(e^A - I_A)B \\ \hline 0 & 0 \end{array}\right],$$

where I is the identity matrix of same dimension as J and I_A is the identity matrix of same dimension as A. The expression $A^{-1}(e^A - I_A)$ is the ϕ_1-function of A. In this way it should be possible to reduce the cost of the calculation of the matrix exponential. We have not taken advantage of this fact in the calculations of this paper, because for now we are only interested in some of the aspects of the theoretical computation time.

3.6.5 The Expint Matlab package

The Expint Matlab package is described in [1]. The package consists of constant step exponential solvers for problems similar to equation (27). We have made a variable step-size solver by using two solvers from the Expint package with different order.

The solvers of the Expint package are set up with the same user interface. The ϕ_k-functions are made by using diagonal Padé approximants, and they are defined by an integral representation

$$\phi_k(z) = \frac{1}{(k-1)!}\int_0^1 e^{z(1-x)}x^{k-1}\mathrm{d}x,$$

for $k = 1, 2, \cdots$. The ϕ_k-functions are calculated once per integration. Since, for the variable time step solver which we have developed for use in this paper, the linear term of the solution changes for each step, the ϕ_k-functions are calculated once per time step. We have sought to find two methods within the Expint framework which can be used to estimate the local error in a step-size solver. We wanted two methods which share the same ϕ_k-functions and exponential functions, because these are the most expensive parts of the solver. This claim is based on the asymptotic cost theory (see Section 4). A method pair which satisfies these requirements is the commutator free Lie group method of order four (CFREE4) and the order one Nørsett-Euler method.

The computational costs of the solvers of the Expint package are listed in [1, p. 22]. Among the one-step exponential methods of order four, the commutator free exponential method and a four order RKMK method are the theoretically most efficient. As noted above, the RKMK scheme is not suitable for stiff problems, and thus we have opted for the commutator free method. The reason for choosing the Nørsett-Euler rather than the Lawson-Euler method is that the latter does not seem to be suitable for our problem, giving small steps even at modest accuracy requirements. We see from Table 7 that the commutator free method requires four ϕ_k-evaluations and nine matrix-vector multiplications, while the Nørsett-Euler method needs two ϕ_k-functions and two matrix-vector multiplications. The four stages of the commutator free method require four function evaluations of the non stiff system, in addition there is one function evaluation for the Nørsett-Euler method. The function evaluation of the Nørsett-Euler method coincides with the first function evaluation of the commutator free method. Moreover, there are less than 31 function evaluations of the stiff part of the system when calculating the Jacobian matrix. The calculation complexity of the ϕ_k-functions would be reduced significantly if we employ what we know about the sparsity of the Jacobian matrix which we mentioned in Section 3.6.4.

3.7 Step-size selection

The Winslow system has several transient phases in its different components and the stiffness of the system varies with the state of the system. Consequently, the step-size required by a numerical solver depends on the state of the system, and it is useful to vary the step-size accordingly.

Name	Non stiff p	Stiff p	Stages s	Output r	ϕ_k-evals	mat-vecs
Lawson-Euler	1	1	1	1	1	1
Nørsett-Euler	1	1	1	1	2	2
Cfree4	4	2	4	1	4	9

Table 7: Key properties of the exponential methods of the Expint package used in this paper [1]

First we will introduce the concept of local error. The exact solution of the system (9) integrated from t_0 to t_1 is

$$y(t_1) = \int_{t_0}^{t_1} f(y(t))\mathrm{d}t.$$

The solution of a general numerical one step method $\Phi_h(y)$ applied to a system (9) is

$$y_1 = \Phi_h(y_0). \tag{31}$$

The error added to the exact solution from each time step of the numerical method is called the the *local error* and is given by

$$r_{loc} = |y_1 - y(t_1)|,$$

where the starting value for the numerical solution is exact. For practical purposes we use two numerical methods of different order and with the same initial value for the local error approximation.

For Runge-Kutta methods it is often possible to find embedded pairs, i.e. methods of different order but with the same internal steps (see Section 3.3).

In the case of splitting methods, it is more difficult to construct a step-size corrector. One possibility is to use a Lie-Trotter/Strang-pair, since asymptotically for $h \to 0$, the Strang splitting method has higher order than the Lie-Trotter splitting. Unfortunately, for stiff ODE systems, the order of the Strang splitting method and the order of the Lie-Trotter splitting method may not differ for some step-sizes (see [10]). The step sizes for which the two methods have same order, are determined by some stiffness parameter. Since the Strang splitting method is not guaranteed to have higher order than the Lie-Trotter splitting, it could be problematic to make a step size corrector based on such a pair. Generally, it is difficult to find a method of different order than the splitting method that does not require many additional function evaluations or at least additional linear algebra calculations. We have implemented a Strang splitting where each sub-method is a Runge-Kutta pair. Then, the minimal step-size of the sub-methods is chosen as step-size for the entire Strang method. One problem with this approach is a large number of rejected steps. In addition, we seem to lose some information about the global behaviour of the solution, which may lead to large errors.

For exponential methods, we have used the methods already implemented in the Expint package, and found a pair of order 4(1), where the order one step-size control does not require additional ϕ-function evaluations. In order to get a better step-size control, we could use a pair of order 4(2) instead, but we have chosen the 4(1) pair because it is simple to implement.

There is also a technique called Richardson extrapolation for which the methods of different order are found by using the same method with different step-sizes for the error estimation (see e.g. [6, p. 164f]). This technique, however, requires a lot of function evaluations, and as we seek to minimise the number of function evaluations, we have not considered Richardson extrapolation.

The size of the next step can be estimated by

$$h_{new} = h\left(\frac{\mathrm{TOL}}{r_{loc}}\right)^{(1/(p+1))},$$

where p is the order of the local error estimate of the method, h is the previous step-size, TOL is the internal tolerance which we want for the local error and r_{loc} is the approximation to the local error. There are techniques for optimising the step-size control, e.g. how to chose an appropriate first step and how to minimise the number of rejected steps. In this paper the step-size control has been used to indicate optimal step-size, ignoring step rejections and the cost of the estimation. More on step-size selection can be found in [7, pp. 123-127].

4 Computational cost theory

The computational cost of a numerical method depends on factors such as the design of the computer, the numerical method, the methods of linear algebra, the implementation of the algorithms and properties of the equation system. In this paper, we are looking at the theoretical cost of a numerical ODE solver in terms of linear algebra computations and function evaluations, and not its actual speed. One reason for this is that speed is hardware and programming language dependent, and a good implementation may require special programming skills beyond the scope of theoretical numerical mathematics.

This outline is based on the assumption that the ODE system is stiff and non-linear. Furthermore, we assume that it is possible to reduce the stiff nature of the system by removing some components from the original system. We then get a stiff system with few non-zero components and a non-stiff system, which added together gives the original system. The expression "few non-zero components in the stiff part" means that their number does not depend on the system size of the original system. The non-stiff part is still proportional to the original system size.

4.1 The step-size

The step-size of a numerical method is usually determined by the accuracy which we require for the solution. When solving stiff systems by explicit solvers, the step-size is determined by stability concerns. We will outline a theory which is valid for all the numerical solvers of this paper.

4.1.1 Implicit solver

The length of the steps required by an implicit solver depends on the order of the method, the required accuracy and the problem on which we are using the solver. Inspired by the order theory for numerical methods (see [5, p. 47f]), this relation can be expressed as

$$\alpha_i(y)h_i^{p_i+1} \leq \text{TOL}.$$

Here p_i is the order of the method, $\alpha_i(y)$ is a function of the state of the system, and TOL is the local error tolerance per step. The tolerance is set according to how accurate we want the solution to be. The function $\alpha_i(y)$ depends on the ODE system which we are dealing with. The relation has to be satisfied for all y. In order to ensure this for constant step solvers, the step-size h_i must satisfy the inequality

$$h_i \leq h_{ic}$$

where h_{ic} is the critical or maximum allowed step-size. The computational cost of each step is assumed constant throughout the integration interval and depends on the implicit method that we use. For most implicit solvers, it is not necessary to calculate a new Jacobian matrix at each step. In this case, the cost of the step will depend on whether or not a new Jacobian matrix is calculated. For the stiff part of the split system, we will denote the constant cost per step by c_i. For a general stiff system, the cost per step of the implicit solver is $c_i(n)$ for a system of size n.

4.1.2 Explicit solver

The explicit solver must also satisfy an accuracy requirement similar to the one for the implicit solver. The outline of this requirement is identical to the outline above, but the values may be different, and we denote the accuracy criterion by

$$\alpha_e(y)h_e^{p_e+1} \leq \text{TOL}.$$

The requirement on the step-size of a constant step solver is consequently

$$h_e \leq h_{ec}.$$

In addition there is a stability requirement which has to be taken into account in case the underlying problem is stiff. The concept of stability was introduced in Section 3.1. Based on this theory, the stability requirement is

$$|\lambda h_e| \leq \tilde{d}. \tag{32}$$

Here λ is a stiffness parameter, which in applications could be the most negative real eigenvalue, but which is any good parameter measuring the stiffness. For a discussion of different ways of determining the stiffness parameter of an ODE system, see e.g. [8]. The parameter $\tilde{d}$ is a constant determined by the numerical method. The exact solution of (11) (for a linear system) after n time steps of length h_e is

$$y_n = e^{\lambda h_e n}y_0 = (e^{\lambda h_e})^n y_0.$$

In order to assure convergence of the method, the condition $|e^{\lambda h_e}| < 1$ must be fulfilled. When $|e^{\lambda h_e}| = 1$, the solution is constant. A large λh_e leads to fast convergence. Thus λh_e in (32) should be as large as possible, i.e. close to $\tilde{d}$. This is a problem for a method with variable step-size, because the step-size corrector is then likely to reject steps frequently. In order to avoid this, we multiply $\tilde{d}$ by a safety factor $\gamma < 1$ to get a parameter $d = \gamma\tilde{d}$. In the notation of this section and by application of the definition of stiffness in Section 3.2, the problem is stiff if

$$\alpha_e(y)h_e^{p_e+1} \leq \text{TOL} \iff |\lambda h_e| > d.$$

Each step of the explicit solver is associated with a cost function $c_e(n)$, depending on the size of the system n.

4.1.3 Exponential integrator

The requirement on the step-size for an exponential integrator is similar to the expression proposed for the implicit integrator

$$\alpha_{exp}(y)h_{exp}^{p_{exp}+1} \leq \text{TOL}.$$

The cost per step depends on the system size n and is denoted by $c_{exp}(n)$.

4.1.4 Splitting method

The splitting method solves different parts of the original problem separately, by using different solvers. In addition to the error committed by the solvers of each part, the global method adds an error. Subjected to the same argumentation as for the implicit method, the error can be defined as

$$\alpha_s(y)h_s^{p_s+1} \leq \text{TOL}.$$

The computational cost per step depends on n and is denoted by $c_s(n)$.

4.1.5 Estimating α

In order to make the above theory practically useful, it is necessary to find a way to estimate α. The $\alpha(y)$ depends on both the method and the problem at hand. One idea for estimating α for a wide range of methods including Runge-Kutta methods, partitioned methods, splitting methods and commutator free Lie group methods is by use of B-series. The B-series approach is proposed for finding the order conditions of a method, but by studying first non-vanishing term of the series it should be possible to get an expression for the error of the method. For an introduction to B-series, see e.g. [5, p. 52f.]. A procedure for finding the error term by B-series is not straight forward, and involves unknowns which would have to be estimated. Especially for methods of high order, the B-series approach tends to be intricate and technical. Also because the B-series has to be calculated for all values of y during integration, this method could be very complicated for a non-linear system, e.g. the Winslow system. For the time being, we will leave the α as an unknown function.

4.2 Computational cost per step

The computational cost per step depends on several factors. We chose to regard the cost as a function of the problem size and the right hand side of the ODE system (9). In the following part, it is necessary to bear in mind that the function f and thus the computational cost of it may depend on the state of the system, although here we will assume that the cost is constant. In addition there are several constant factors. There are two main sources of computational time consumption. These are the evaluation time of the function and the linear algebra. For the computational cost, we will use $\mathcal{O}$-notation, i.e.

$$\mathcal{O}(n^q) = Cn^q,$$

where C is some constant, n is the system size and q is some integer.

For the Winslow system, the function evaluations are expensive and therefore we seek to find a method which minimises the number of function evaluations. In order to get an impression of the cost of the Winslow system, we used the built-in Matlab command PROFILE. We noted that for the multi-step solver ODE15S, the Winslow function evaluations takes about 45% of the total time, for Runge-Kutta-like solvers ODE23S, ODE23T and ODE23TB the proportion of times spent at calculating the Winslow system was approximately 70%, 40% and 50% respectively. It is important to keep in mind that these proportions change slightly for each run of the solver. They might also change for other implementations of the Winslow system.

4.2.1 Cost of implicit method

The most expensive part of the linear algebra of an implicit method is the inversion of the Jacobian matrix. Non-iterative schemes typically require $\mathcal{O}(n^3)$ binary operations for doing LU-factorisations. In addition, the calculation of the Jacobian matrix requires n function evaluations. Generally, it is difficult to anticipate whether it is the linear algebra or the function evaluations which are most costly.

An expression for the cost per step of an implicit method is

$$c_i = \mathcal{O}(n)c_f + \mathcal{O}(n^3).$$

Here c_f is the cost associated with the evaluation of the function. The use of $\mathcal{O}$-notation is not absolutely correct, but because of the n-dependency of c_f, we have chosen to let the two terms of the addition be independent of each other.

It is important to note that the Jacobian matrix does not need to be updated at each step. When the previous Jacobian matrix is kept, we can also keep the LU-factorisation of the previous step, and the cost is reduced to

$$\text{cost}_i = \mathcal{O}(1)c_f + \mathcal{O}(n^2).$$

In the case that we have a split system where the stiff part is small, i.e. the number of non-zero elements of the function does not depend on n, the expression can be simplified as

$$c_i = \mathcal{O}(1)c_f + c_m.$$

Here c_f is still the cost of a function evaluation, and $c_m = \mathcal{O}(1)$ is the cost of the linear algebra. We will make use of the last expression in the expression for the cost of the splitting method.

4.2.2 Cost of explicit method

The explicit method is characterised by a linear algebra cost of $\mathcal{O}(n)$ from scalar-vector multiplications and vector additions. It uses only $\mathcal{O}(1)$ function evaluations. The cost of the explicit method is consequently

$$c_e = \mathcal{O}(1)c_f + \mathcal{O}(n).$$

4.2.3 Cost of exponential integrators

Exponential integrators require the computation of the matrix exponential and functions thereof. Matrix exponentials generally cost $\mathcal{O}(n^3)$, where the notation hides a constant considerably larger than that for the LU-decomposition of an implicit method (see [14, pp. 5 and 16]). Thus, the total cost of the exponential integrator is

$$c_{exp} = \mathcal{O}(1)c_f + \mathcal{O}(n^3).$$

In Section 3.6, we saw that the matrix exponential for the stiff part of the Winslow system, may not depend on n. In this case, the multiplications of equations (29) are the asymptotically most expensive part of the exponential integrator. Now, the cost per step of the exponential integrator is

$$c_{exp} = \mathcal{O}(1)c_f + \mathcal{O}(n^2).$$

We also notice that the cost of evaluating the Winslow system c_f is really the cost of evaluating $N(y)$ of equation (27). Although the cost of evaluating $N(y)$ is higher than that of evaluating f, we assume that the difference in computation complexity is negligible.

4.2.4 Cost of splitting method

The computational cost per time step is the sum

$$c_s(n) = \sum_{k=1}^{sp} \gamma_k c_k$$

where *sp* is the number of partitions of the original vector field, γ_k is the number of calls to each sub-method and c_k is the cost per step of each sub-method. Thus, the Strang splitting using an implicit and an explicit method has the cost function

$$\begin{aligned} c_s &= 2c_i(n_s) + c_e(n_{ns}) \approx 2c_i + c_e(n_{ns}) = \\ &\quad 2\mathcal{O}(1)c_{f_s} + 2\mathcal{O}(1) + \mathcal{O}(1)c_{f_{ns}} + \mathcal{O}(n_{ns}) \approx \mathcal{O}(1)c_f + \mathcal{O}(n). \end{aligned}$$

Here we assume that the cost of evaluating the stiff system is negligible in comparison with the non-stiff system, and that the non-stiff system is approximately as expensive as the original system. We note that the $\mathcal{O}$-functions above change after the $\approx$-sign.

4.3 Total computational cost

The total cost of the method is

$$c_{tot} = \sum_{i=1}^{N} c_{method}^{[i]},$$

where N is the total number of steps, $c_{method}^{[i]}$ is the cost of step number i for some method. For a method with constant step-size and constant cost per step, the total cost of an integration is

$$c_{tot} = N c_{method}.$$

For the Winslow system, the constant step-size approach is slow, e.g. because the region of very negative eigenvalues which determines a critical step-size is small. Therefore, the step-size of a constant step solver is much shorter than necessary for most of the system. In the case of variable step-size, the cost of the step-size control has to be taken into account. This cost depends on the number of rejected steps and the cost of the method which we use to estimate the local error. For Runge-Kutta methods, there are embedded methods which require only $\mathcal{O}(n)$ additional linear algebra computations. As seen in Section 3.7, the step-size selector of splitting methods, could be another splitting method. An exponential method could use another exponential method as step-size corrector, possibly without calculating new exponentials.

In table 8 we have summed up the cost per step of the different solvers of this paper. Since the solvers are not directly comparable, it is not possible a priori to recommend one, but both the Strang splitting and the exponential method we study in this paper could be less expensive per step than an ESDIRK method. It is important to notice that the function evaluations needed to find the Jacobian matrix are not included in the table, but hidden in the calculation of the Jacobian matrix.

5 Results

In this section, we will show the results which we obtained when splitting the Winslow system and solving the split system by a Strang splitting method and an exponential integrator. We will measure the quality of a solver by investigating the cost of obtaining a small global error. The *global error* is given by

$$r_{global} = |y_n - y(t_n)|,$$

where y_n and $y(t_n)$ are the solution at time $t = t_n$ for the numerical and exact solutions, respectively. The methods implemented for this thesis were rather slow in actual computation time, therefore we have restricted ourselves to an integration interval of 10 ms.

	CFREE4	ESDIRK32	Strang split ODE23/ESDIRK32
Jacobian matrix of			
- entire system		yes	
- stiff part	yes		yes
Function evaluations			
- of entire system		4	
- of stiff part			$2 \cdot 3$
- of non stiff part			4
- of $N(y)$	4		
ϕ-evaluations	4		
LU-factorisation			
- system size		full	not full

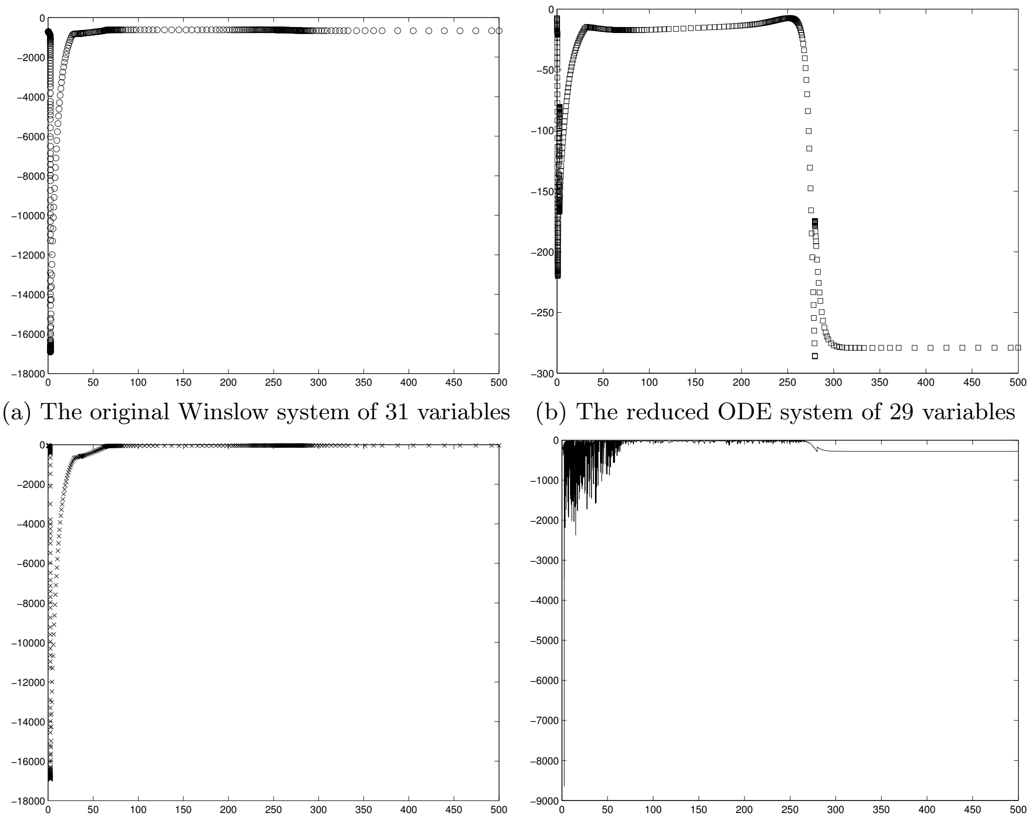
Table 8: Some key cost parameters for the numerical methods of this paper

5.1 Splitting a stiff ODE system into a stiff and a non-stiff part

We have seen that the stiffness of a system like (9) is indicated by large negative eigenvalues of the Jacobian matrix. As mentioned in Section 3.1, explicit Runge-Kutta methods tend to be unstable if the absolute value of the product of the most negative eigenvalue and the step-size is greater than a certain stiffness parameter which depends on the numerical method. Therefore, stiff systems force the step-size to be small. A study of the eigenvalues of the numerically approximated Jacobian matrix of the ODE-system for different values of y, revealed that there was one large negative eigenvalue value, which may be responsible for the stiffness of the system. The y-values at which we calculated the Jacobian matrix were determined by the solution of the Winslow system by Matlab solver ODE15S at tolerance 10^{-10}. The extreme eigenvalue was associated with an eigenvector which was dominated by the variable P_{C_1} and P_{O_1} (see Appendix A). Variable P_{O_1} depends strongly on P_{C_1}, and both variables depend on variable $\left[Ca^{2+}\right]_{ss}$. Consequently, by removing the corresponding differential equations, i.e. 12 and 30, from the system, we hoped that the resulting system would have smaller negative eigenvalues. Computing the eigenvalues of the Jacobian matrix of the reduced system, we noticed that the most extreme eigenvalues had disappeared. Constructing a new ODE system of the differential equations for P_{C_1} and $\left[Ca^{2+}\right]_{ss}$, we observed that the eigenvalues of this system are as extreme as the eigenvalues of the original system. In this way it was possible to split the original system into a stiff and a non-stiff system. It is important to notice that the stiffness change for different states of the system, and thus for some states, the "non-stiff" system may be more stiff than the "stiff" system. As we proposed in Section 3.6 for the exponential integrators, we have transformed the split system into

$$y'(t) = f(y(t)) = Ly(t) + N(y(t)), \qquad y(t_0) = y_0.$$

We have also investigated the eigenvalues of the Jacobian matrix of $N(y(t))$.



(a) The original Winslow system of 31 variables

(b) The reduced ODE system of 29 variables

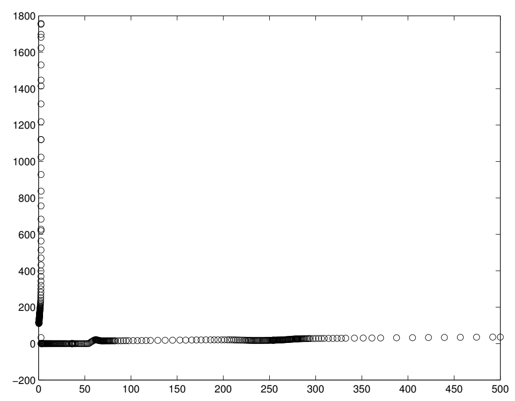
(c) The small stiff ODE system of two variables

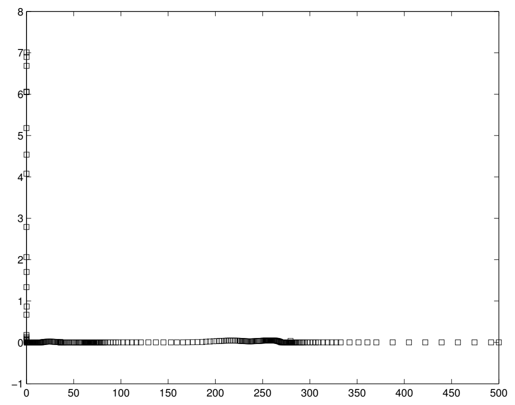
(d) The ODE system $N(y)$ for the exponential integrator

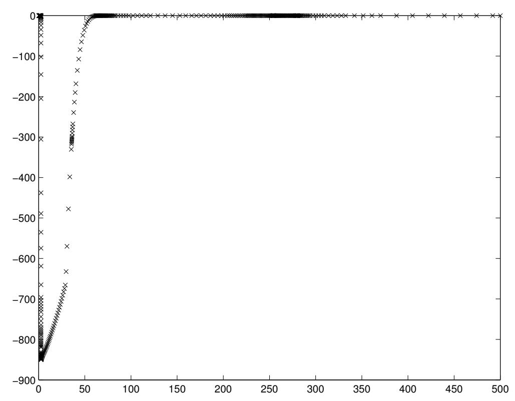
Figure 4: Distribution of the most extreme negative eigenvalue of the original and the partitioned systems for time from 0 to 500 ms

	Min. eigenvalue	Max. eigenvalue
Original system	-1.6916×10^4	1.7578×10^3
Modified system	-286.1646	7.0087
Resulting stiff system	-1.6916×10^4	-7.8256×10^{-9}
The system $N(y)$	-8.6429×10^3	9.7284×10^4

Table 9: The extreme real part of the eigenvalues of the Jacobian matrix for the Winslow system


(a) The original Winslow system of 31 variables


(b) The non-stiff part of the ODE system


(c) The stiff part of the ODE system

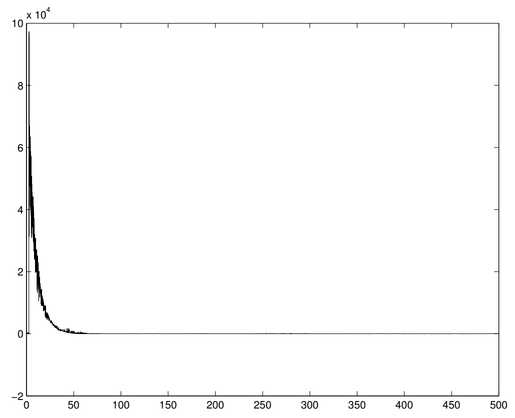
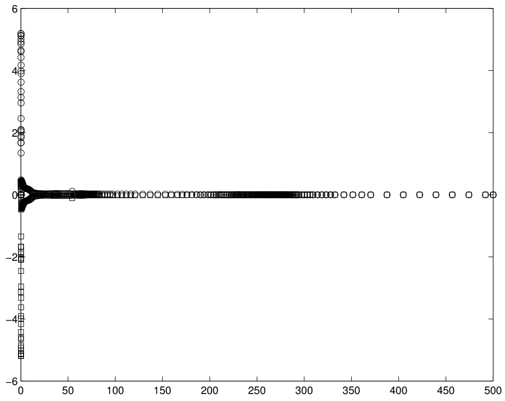

(d) The ODE system $N(y)$ for the exponential integrator

Figure 5: Distribution of the most extreme positive eigenvalue of the original and the partitioned systems for time 0 to 500 ms. This shows that the large initial eigenvalues are lost in the partitioning


(a) The original Winslow system of 31 variables

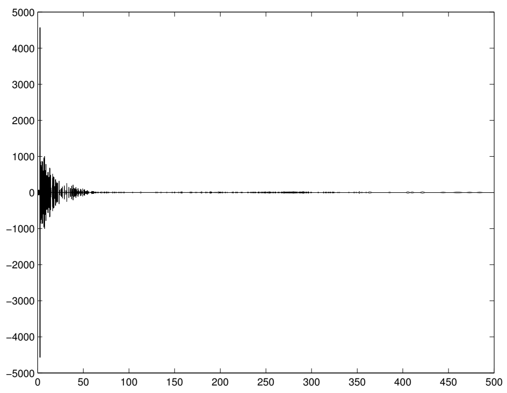

(b) The ODE system $N(y)$ for the exponential integrator

Figure 6: Distribution of maximum and minimum imaginary parts of the eigenvalues. The non-stiff part of the partitioned system shares the pattern of the original system, albeit with smaller magnitude. For the stiff system the imaginary parts are always zero. The eigenvalues of the ODE system $N(y)$ has very large imaginary parts

In Figure 4 we see how the largest negative eigenvalues of the original system, the reduced system and the system of the variables taken out of the system are distributed in a time span from 0 to 500 ms. We see that initially the modified, "non-stiff", system has considerably smaller negative eigenvalues than the original system. We then observe that the original system goes through phases where the eigenvalues are not that extreme. The stiff system goes towards a phase where the most extreme negative eigenvalue approaches zero. This indicates that the supposedly stiff system is only stiff initially. The non-stiff system on the other hand experiences two regions of moderately large negative eigenvalues. This means that the two variables which we have split out of the system are responsible for the most negative eigenvalues, globally speaking, but that the negative eigenvalues of some regions are dominated by other variables. The assumption that the $N(y)$-function of equation (27) is non-stiff, is to a certain degree refuted by the fact that the eigenvalues of its Jacobian matrix are large and negative. Still, the negative eigenvalues are smaller than for the original system.

The fact that the Winslow system can be split into a stiff and a non-stiff part by filtering out some of the components of the system, suggests using a splitting method. There is however also the possibility of using different solvers for the stiff and the non-stiff states of the original system. When to change between the solver for the system in the stiff state and the solver for the non-stiff state can be determined a priory by a analysis similar to the one above. Alternatively, it could be determined by implementation of an automatic stiffness detector (see e.g. [7, p. 21]). In this paper we will only investigate the splitting approach.

Investigating the maximum eigenvalues, we find that initially the original system has large positive eigenvalues. Apparently, the large positive eigenvalues are lost when partitioning the system, as can be seen from Figure 5. The stiff system does not have any positive eigenvalues, and Figure 5c only displays the eigenvalues closest to zero. For the $N(y)$-system there are even larger positive eigenvalues than for the original system.

Regarding the imaginary part of the eigenvalues, it can be seen from Figure 6 that there is only initially a non-zero imaginary component. The same pattern which can be seen for the original system is also seen for the non-stiff partition. The stiff partition has no non-zero imaginary component of its eigenvalues. As was the case for the original system, there are also large positive and negative imaginary parts of the Jacobian matrix of the $N(y)$-system.

5.2 Splitting ESDIRK/RK

The motivation for using a splitting method is that we hope that the step-size of the splitting method is considerably larger than for an explicit system while the computation time per step is less than for an implicit method. We saw in Section 4 that theoretically, this might be the case.

In Section 3.7, we were discouraged from using a step-size corrector based on the Strang splitting method and the Lie-Trotter splitting method for the Winslow system. Instead, we proposed to have a step-size controller in each sub-method of the Strang splitting method, hoping that this would give sufficiently good step-sizes for the entire system. The initial motivation for choosing this step-size corrector was that we wanted to investigate the step-size of the stiff and the non-stiff part of the Winslow system.

First, we applied a step-size corrector to the non-stiff part of the splitting method in order to see if the step-sizes of an explicit solver for the non-stiff part of the Winslow system increase with respect to the original system. Then, by applying a step-size corrector to the stiff system as well, we see from Table 10 that the step length is determined by the explicit system for most parts of the calculation interval, except from the part where the stiff system has most negative real eigenvalues. Since the implicit method is supposed to be A-stable, it is not evident from the point of view of stiffness why it should limit the step-size more than the explicit solver in the stiff phase. One explanation is that the stiff system is in a transient phase, as seen in Figure 8, and that the small step-sizes are due to accuracy requirements.

In Figure 7, we see that in the phase where the negative eigenvalues of the Jacobian matrix of the Winslow system are small, the splitting method generally does not accept larger step-sizes than the explicit ODE23 solver, although it does in some regions. In the stiff region of the original system, however, the step-size of the Strang splitting method increases considerably with respect to the step-size of the entire system solved by an explicit RK-method. A problem with the comparison is that we have not taken into account the global error of the numerical solutions. In order to measure the quality of a solver we are interested in the step-size required in order to get a sufficiently small global error. Nevertheless, the step-sizes of the ODE23 solver in the stiff region are so small that the Strang splitting method is very likely to accept larger steps than the explicit method even when global error is taken into account.

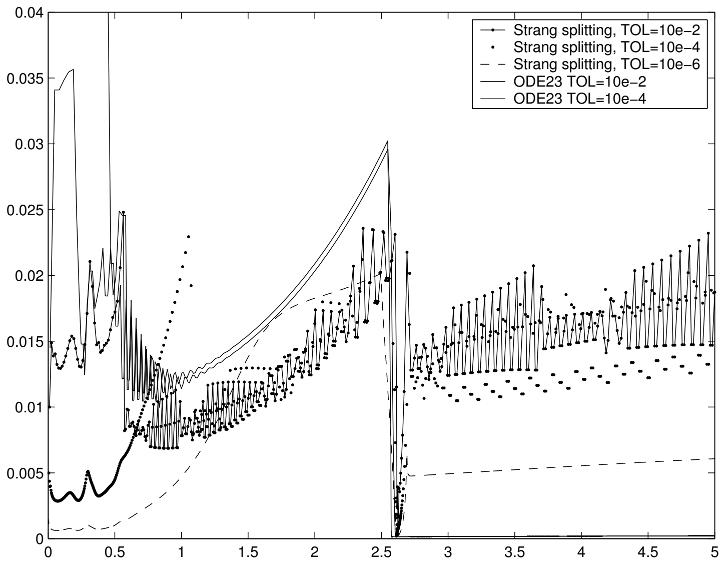


Figure 7: The step-size obtained by a simple step-size selector from time 0 to 5 ms. The splitting method accepts larger steps than the explicit solver in the stiff region.

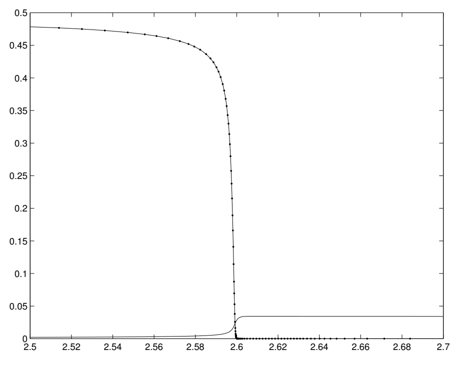


Figure 8: A transient phase in the stiff part of the system may explain the need for small step-sizes for the implicit solver in the splitting method

A problem with the splitting method for which the step-size selectors are put in each sub method, is the large global error. Compared to a reference solution obtained with Matlab solver ODE15s with internal error 10^{-12} at time 10.0107, we find that the norm of the difference of each component is 0.0840. Compared to the total norm of the reference solution 159.4921, the relative error is $5.2692 \cdot 10^{-4}$. For an internal error of 10^{-4} this error is perhaps not very high, but the number hides the fact that some components of small magnitude may have an considerable error. The qualitative difference between the reference solution and the splitting method for the above case is plotted in Figure 9a. In comparison, a splitting method with constant step-size 10^{-4} is qualitatively very accurate, although it has an error after 3.0001 seconds of $3.8959 \cdot 10^{-4}$. For the explicit method with step-size control and internal error 10^{-6}, the error is qualitatively nice and the final error is $1.4407 \cdot 10^{-8}$.

In order to minimise the error of the sub-methods of the Strang splitting, we exchange the ESDIRK method and the explicit method in the Strang splitting scheme by Matlab solver ODE15S. We see that the local error gets smaller, but still it is considerable. By comparing the result of a Strang splitting method using only ODE15S and a calculation by ODE15S on the original system, we observe in Figure 9b a large local error which coincides with the transient phase of the stiff system as mentioned above. In other words, a large local error at this point seems to be unavoidable for the Strang splitting. More work needs to be done in order to find a step-size corrector which is able to minimise the negative effects of this peak in the local error for the rest of the integration. As mentioned above, a run of the Strang splitting method with small constant step-size gives a more accurate global result.

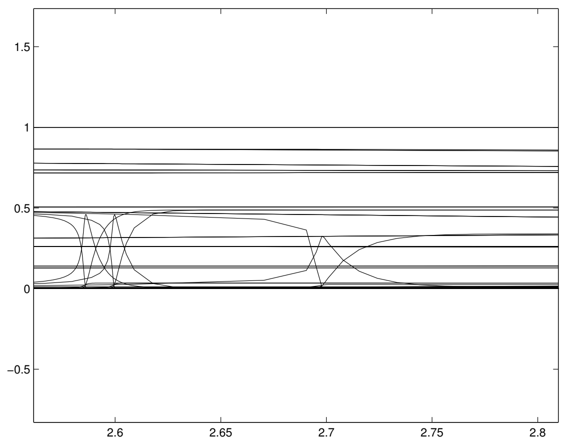
The asymptotic cost theory presented in Section 4 gives the same asymptotic calculation time per step for the explicit solver and the splitting method (except for the constants hidden in the $\mathcal{O}$-notation). Assuming that the step-size selection of the splitting method is not considerably more costly than for the explicit Runge-Kutta solver, the splitting method may have advantages over the explicit method. As yet however, the implementation of the splitting method is slow in terms of actual computation time, and we have not yet found a good step-size corrector for it. Implementation issues must be studied further before we can conclude that there is in fact a fast splitting method.

5.3 Exponential integration

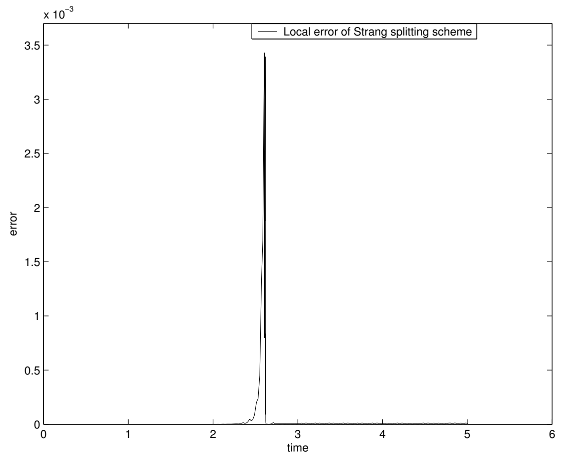
We have solved the split system by use of the Expint package for Matlab. We have used two different approaches for making a system of form (27). First we let $L = J$, where $J = J(y_0)$ is the Jacobian matrix of the Winslow system at the start of each step, and $N(y) = f(y) - Ly$. The second approach was to construct the matrix

$$L^* = \begin{pmatrix} \frac{f_{s1}(y)}{y_{s1}} & 0 \\ 0 & \frac{f_{s2}(y)}{y_{s2}} \end{pmatrix},$$

where $f_{s1}(y)$ and $f_{s2}(y)$ are the two non-zero elements of the stiff system, and y_{s1} and y_{s2} are the corresponding state variables. We then let $N_1^* = f_{ns}$ and $N_2^*(y) = f(y) - L^*y$.



(a) The qualitative difference between the splitting method (right peak) and a reference solution (left peak)



(b) The local error between a Strang splitting scheme using ODE15S and a run of the ODE15S integrator on the original system

Figure 9: The splitting method with a simple step-size corrector is not able to reproduce the results of a reference solution

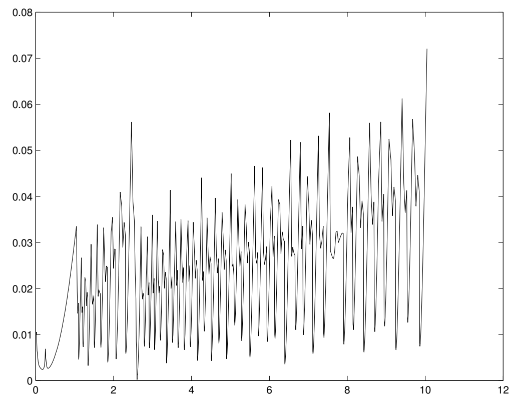
Time	Step-size	Step-size change	Method
0.0594	0.0606	0.0198	ERK
0.1002	0.0442	0.0049	ERK
0.3788	0.0476	0.0203	ERK
0.4696	0.0335	0.0109	ERK
0.4922	0.0249	0.0073	ERK
0.5097	0.0189	0.0023	ERK
0.5263	0.0170	0.0018	ERK
2.6050	0.0323	0.0199	ESDIRK
2.6174	0.0289	0.0263	ESDIRK
2.6201	0.0204	0.0134	ESDIRK
2.6270	0.0148	0.0133	ESDIRK
2.6285	0.0092	0.0067	ESDIRK
2.6310	0.0075	0.0011	ESDIRK
2.6310	0.0063	0.0048	ESDIRK
2.6326	0.0025	0.0010	ESDIRK
2.7328	0.0204	0.0033	ERK

Table 10: Survey of step-size changes larger than 10^{-6}.

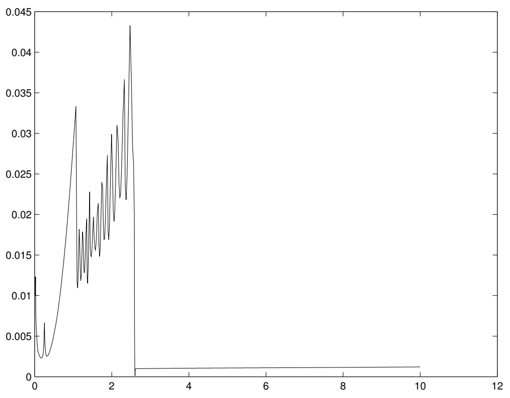
For N_1^*, the solver breaks down in the region of large negative eigenvalues. For $N_2^*(y)$, the solution is also unstable. We have therefore settled for the first approach. The global error is 0.0069 when calculated with respect to the same reference solution as we used in Section 5.2. But by taking a qualitative look at the output, it is clear that the solution by Expint is considerably closer to the reference solution. It is not possible to see differences as we saw in Figure 9a.

We implemented a step-size corrector for the fourth order commutator free method of the Expint package, CFREE4, by use of the ESDIRK23 method. The resulting step-size versus time in Figure 10a shows a step-size pattern which is fluctuating in the stiff region. For a step-size corrector based on the Lawson-Euler exponential solver, the step-size in the stiff region is not fluctuating, but small as can be seen in 10b. By the use of the Nørsett-Euler method in the step-size corrector, we obtained, for a tolerance of 10^{-2}, a more fluctuating pattern of step-sizes comparable to those obtained by the ESDIRK23 based step-size control. For more accurate integration, the step-size was not fluctuating and the step-sizes were larger than for the step-size corrector based on the Lawson-Euler method. The step-sizes are plotted in Figure 10c. The global error of the CFREE4/Nørsett-Euler scheme is compared to the reference solution and plotted in Figure 11a. We observe that the global error does not only depend on time, but also on the state of the system. The observation of large global error in a region is supported by the large local error in the same region in Figure 11b.

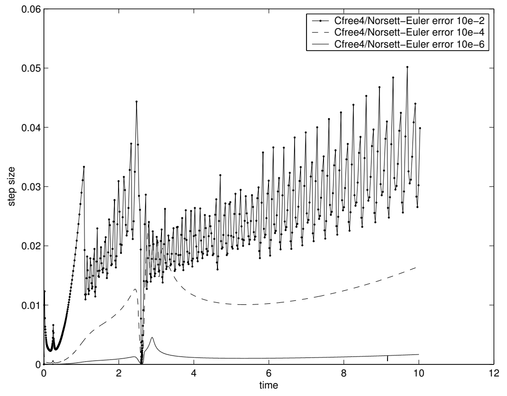
In order to answer the question whether the $N(y)$ system is stiff or not, we saw in Section 5.1 that the Jacobian matrix of the ODE system $N(y)$ has large negative real



(a) Step-sizes for CFREE4 with ESDIRK23 as step-size corrector, tolerance 10^{-2}

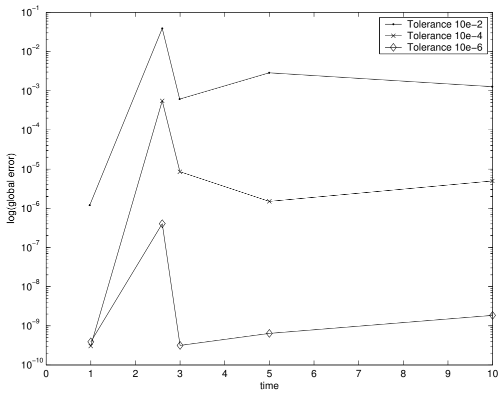


(b) Step-sizes for CFREE4 with Lawson-Euler as step-size corrector, tolerance 10^{-2}

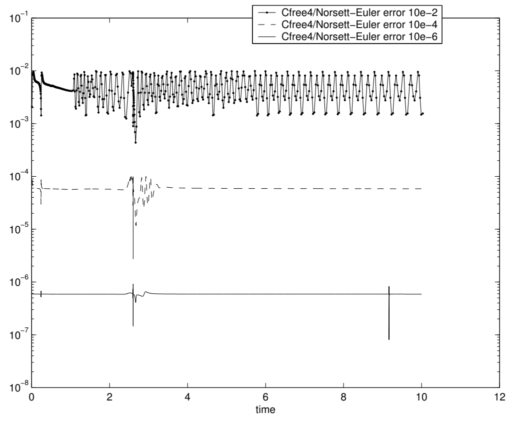


(c) Step-sizes for CFREE4 with Nørsett-Euler as step-size corrector, tolerances 10^{-2}, 10^{-4} and 10^{-6}

Figure 10: Step-size for numerical solution of the Winslow system by exponential integrator CFREE4 from the Expint package



(a) Global error of the Cfree4/Nørsett-Euler scheme



(b) Local error of the Cfree4/Nørsett-Euler scheme

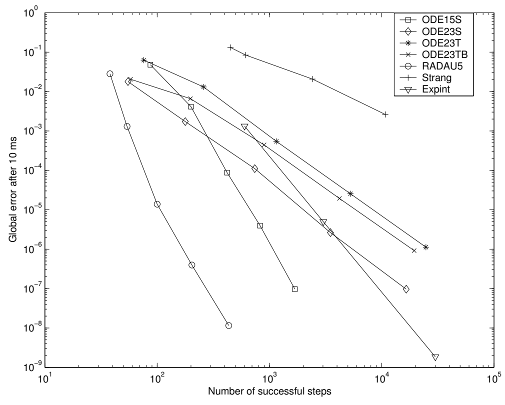
Figure 11: Estimations of global and local error for numerical solution of the Winslow system by exponential integrator CFREE4 from the Expint package

eigenvalues. Although the eigenvalues are not as large as the negative eigenvalues of the original system, they are much larger than the negative eigenvalues of the non-stiff system. It is important to bear in mind that large negative eigenvalues do not necessarily imply stiffness if for instance the system is in a transient phase. In the Expint solvers, the $N(y)$ is solved by an explicit method, and in the next section we see in Figure 12a that the number of steps taken by the Expint solver is comparable to some of the implicit solvers of Matlab. In other words, it is possible that the $N(y)$ system is not very stiff.

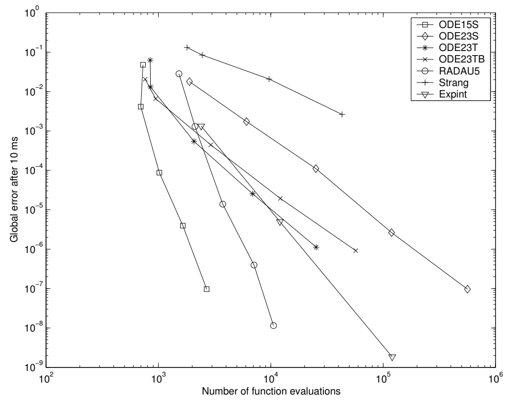
5.4 Comparison of solvers

We will measure the quality of a solver by the cost of obtaining a small global error. We calculated the global error for the built-in Matlab solvers of Section 3.4 for internal tolerances 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8} and 10^{-10} with respect to a reference solution obtained by ODE15S with relative and absolute internal tolerance 10^{-12}. Then, we did the same procedure for the RADAU5 method, the Strang splitting method and the exponential integrator. The results are plotted in Figure 12. We see that the ODE15S method uses few function evaluations, but since ODE15S is a multi-step solver we will not investigate this solver further. The step-sizes of the RADAU5 method are generally large. Since we imposed a step-size restriction on the solver because of the coupling of the ODE and the PDE in the bidomain heart model, we are not able to take advantage of the large time steps of the RADAU5 solver. Note should be taken however that it is possible to interpolate the solution of the RADAU5 solver in order to get the solution at the time that we are interested in. Especially for high accuracy, the RADAU5 method is efficient in terms of function evaluations compared to the one-step implicit solvers of Matlab.

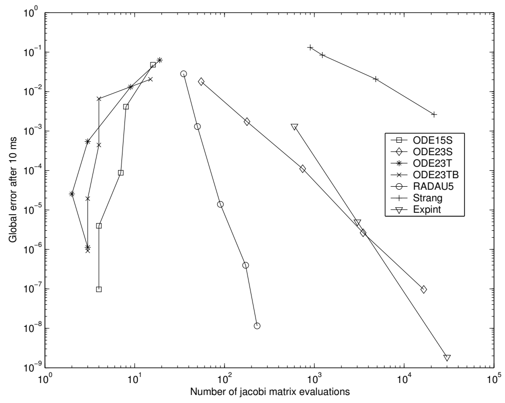
Function evaluations of the Winslow system are costly, as seen in Section 4.2, and we therefore want to reduce their number. The method based on the Expint-solvers generally performs better than the Strang splitting procedure. It is, however, important to note that the most expensive part of the exponential integrator, namely the calculation of the matrix exponential, is not included in this comparison. The number of time steps for the Strang splitting and the Expint method were calculated by the step-size controllers discussed in Sections 5.2 and 5.3. In order to make an estimation of the number of function evaluations, LU factorisations and Jacobian matrix evaluations, we multiplied the number of steps by a suitable number from Table 8. In other words, we do not take into account the cost of the step-size selector. For the Strang splitting method, the number of function evaluations is taken to be the number of evaluations of the non-stiff part of the Winslow system. For the Expint-solver, we count the number of $N(y)$-evaluations. We see that the Strang splitting method performs considerably worse than the other solvers.



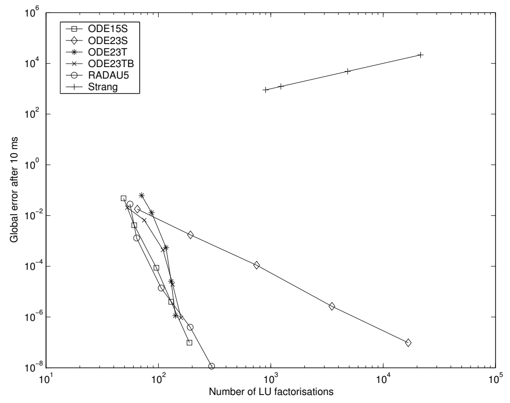
(a) Global error after 10 ms as a function of number of time steps



(b) Global error after 10 ms as a function of number of function evaluations



(c) Global error after 10 ms as a function of number of Jacobian matrix evaluations



(d) Global error after 10 ms as a function of number of LU evaluations

Figure 12: Cost efficiency of different numerical methods applied to the Winslow system

6 Conclusion

In theory, a stiff system split into a stiff and a non-stiff part and which is solved by a numerical splitting method is supposed to accept longer steps than an explicit method while having a smaller cost per step than an implicit method. We showed that it is possible to split the Winslow system into a stiff and a non-stiff system by removing two variables. In order to find the step-size of the splitting method, we constructed different step-size selectors. We have so far been unable to construct an efficient step-size selector which is cheap and which gives the system a small global error. Nevertheless, we were able to show that the non-stiff part of the system could be solved by an explicit solver with larger step-size than that required for the entire system. A step-size selector based on Runge-Kutta pairs for the sub-integrators of a Strang splitting method did not appear to be an optimal way for determining the step-size.

In order to solve the Winslow system by an exponential integrator, we had to calculate the Jacobian matrix of the stiff part of the system and make a new non-stiff part which depends on the Jacobian matrix of the stiff part. The new non-stiff part of the system appeared to have large negative eigenvalues, still we were able to solve the system by use of solvers from the Expint package for Matlab. The step-size of the exponential integrator is larger than for the explicit solver, still much shorter than for the best implicit solvers. The global error of the exponential method with step-size corrector does not only depend on time, but also on the state which the system has reached. More work has to be done, probably on the step size corrector, in order to get a more accurate solution for all states of the system.

Although theoretically the splitting of the Winslow system and the solving of the system by a splitting method or an exponential method might be more efficient than using an implicit solver on the entire system, more work has to be done in order to make good methods for the split system. The most promising of the splitting methods has been the exponential integrator. We believe future work should focus on modifying this solver in order to take advantage of the special properties of the Winslow system such as the sparsity of the Jacobian matrix. The problem of such a specific solver is that it may not work for other cell models than the Winslow model.

An alternative approach to solving the Winslow system could be to use different solvers for different states of the system. Since there are several implicit solvers which solve the Winslow system well, and the Winslow system is not very stiff after an initial phase, this could be a good idea for future work.

References

[1] Håvard Berland, Bård Skaflestad, and Will Wright. EXPINT - A MATLAB package for exponential integrators. PREPRINT NUMERICS NO. 4/2005 available at `http://www.math.ntnu.no/num/expint/expintmanual.pdf` with MATLAB package available at `http://www.math.ntnu.no/num/expint/expint-1.1.tar.gz`.

[2] Elena Celledoni, Arne Marthinsen, and Brynjulf Owren. Commutator-free Lie group methods. *Future Gener. Comput. Syst.*, 19(3):341–352, 2003.

[3] K. Dekker and J.G. Verwer. *Stability of Runge-Kutta methods for stiff nonlinear differential equations.* North-Holland, 1984.

[4] Ch. Engstler. RADAU5: Matlab implementation of the Radau IIA method of order5 by Ch. Engstler after the Fortran Code RADAU5 of Hairer/Wanner. Matlab code available at `http://na.uni-tuebingen.de/pub/codes/radau5.tar.gz`.

[5] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration - Structure Preseving Algorithms for Ordinary Differential Equations.* Springer, 2002.

[6] Ernst Hairer, Syvert Paul Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I.* Springer, 2000.

[7] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II.* Springer, 2002.

[8] Desmond J. Higham and Lloyd N. Trefethen. Stiffness of ODEs. *BIT Numerical Mathematics*, 33(2):285–303, 1993.

[9] M.E. Hosea and L.F. Shampine. Analysis and implementation of TR-BDF2. *Applied Numerical Mathematics*, 20:21–37, 1996.

[10] Roman Kozlov, Anne Kværnø, and Brynjulf Owren. The behaviour of the local error in splitting methods applied to stiff problems. *J. Comput. Phys.*, 195(2):576–593, 2004.

[11] Anne Kværnø. Singly Diagonally Implicit Runge-Kutta Methods with an Explicit First Stage. *BIT Numerical Mathematics*, 44(3):489–502, August 2004.

[12] Werner Liniger and Ralph A. Willoughby. Efficient Integration Methods for Stiff Systems of Ordinary Differential Equations. *SIAM Journal on Numerical Analysis*, 7(1):47–66, March 1970.

[13] Borislav V. Minchev and Will M. Wright. A review of exponential integrators for first order semi-linear problems. PREPRINT NUMERICS NO. 2/2005, available at `http://www.math.ntnu.no/preprint/numerics/2005/N2-2005.ps`.

[14] Cleve Moler and Charles Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM Review*, 45(1):3–49, 2003.

[15] Brynjulf Owren. Order conditions for commutator-free Lie group methods. PREPRINT NUMERICS NO. 7/2005 available at `http://www.math.ntnu.no/preprint/numerics/2005/N7-2005.ps`.

[16] Lawrence F. Shampine and Mark W. Reichelt. The Matlab ode suite. *SIAM Journal of Scientific Computing*, 18(1):1–22, 1997.

[17] Joakim Sundnes, Glenn Terje Lines, Xing Cai, Bjørn Fredrik Nielsen, Kent-Andre Mardal, and Aslak Tveito. *Computing the Electrical Activity in the Heart*. Springer, 2006.

[18] Raimond L. Winslow, Jeremy Rice, Saleet Jafri, Eduardo Marban, and Brian O'Rourke. Mechanisms of Altered Excitation-Contraction Coupling in Canine Tachycardia-Induced Heart Failure, II, Model Studies. *Circulation Research*, 84:571–586, 1999.

A The Winslow ODE system

A.1 The transmembrane potential

$$\frac{\mathrm{d}V}{\mathrm{d}t} = -(I_{Na} + I_{Ca} + I_{Ca,K} + I_{Kr} + I_{Ks} + I_{to} + I_{K1} + I_{Kp} + I_{NaCa} + I_{NaK} + I_{p(Ca)} + I_{Ca,b} + I_{Na,b})$$

A.2 K^+- and Na^+- gate variables

$$\begin{aligned}
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(1-m) - \beta_m m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(1-h) - \beta_h h \\
\frac{\mathrm{d}j}{\mathrm{d}t} &= \alpha_j(1-j) - \beta_j j \\
\frac{\mathrm{d}X_{Kr}}{\mathrm{d}t} &= \frac{X_{Kr}^{\infty} - X_{Kr}}{\tau_{X_{Kr}}} \\
\frac{\mathrm{d}X_{Ks}}{\mathrm{d}t} &= \frac{X_{Ks}^{\infty} - X_{Ks}}{\tau_{X_{Ks}}} \\
\frac{\mathrm{d}X_{to}}{\mathrm{d}t} &= \alpha_{X_{to}}(1-X_{to}) - \beta_{X_{to}} X_{to} \\
\frac{\mathrm{d}Y_{to}}{\mathrm{d}t} &= \alpha_{Y_{to}}(1-Y_{to}) - \beta_{Y_{to}} Y_{to}
\end{aligned}$$

Here all coefficients only depend on the transmembrane potential V.

A.3 RyR-channel

$$\begin{aligned}
\frac{\mathrm{d}P_{C_1}}{\mathrm{d}t} &= -k_a^+ \left[Ca^{2+}\right]_{ss}^4 P_{C_1} + k_a^- P_{O_1} \\
\frac{\mathrm{d}P_{O_1}}{\mathrm{d}t} &= -k_a^+ \left[Ca^{2+}\right]_{ss}^4 P_{C_1} - k_a^- P_{O_1} - k_b^+ \left[Ca^{2+}\right]_{ss}^3 P_{O_1} + k_b^- P_{O_2} - k_c^+ P_{O_1} + k_c^- P_{C_2} \\
\frac{\mathrm{d}P_{O_2}}{\mathrm{d}t} &= -k_b^+ \left[Ca^{2+}\right]_{ss}^3 - k_b^- P_{O_2} \\
\frac{\mathrm{d}P_{C_2}}{\mathrm{d}t} &= k_c^+ P_{O_1} - k_c^- P_{C_2}
\end{aligned}$$

All the k-coefficients are constant. The stiffness of the system is to a great extent attributed to the state variable P_{C_1}.

A.4 L-type Ca^{2+}-channel

$$\begin{aligned}
\frac{\mathrm{d}C_0}{\mathrm{dt}} &= \beta C_1 + \omega C_{Ca0} - (4\alpha + \gamma)C_0 \\
\frac{\mathrm{d}C_1}{\mathrm{dt}} &= 4\alpha C_0 + 2\beta C_2 + \tfrac{\omega}{b}C_{Ca1} - (\beta + 3\alpha + \gamma a)C_1 \\
\frac{\mathrm{d}C_2}{\mathrm{dt}} &= 3\alpha C_1 + 3\beta C_3 + \tfrac{\omega}{b^2}C_{Ca2} - (2\beta + 2\alpha + \gamma a^2)C_2 \\
\frac{\mathrm{d}C_3}{\mathrm{dt}} &= 2\alpha C_2 + 4\beta C_4 + \tfrac{\omega}{b^3}C_{Ca3} - (3\beta + \alpha + \gamma a^3)C_3 \\
\frac{\mathrm{d}C_4}{\mathrm{dt}} &= \alpha C_3 + gO + \tfrac{\omega}{b^4}C_{Ca4} - (4\beta + f + \gamma \alpha^4)C_4 \\
\frac{\mathrm{d}O}{\mathrm{dt}} &= fC_4 - gO \\
\frac{\mathrm{d}C_{Ca0}}{\mathrm{dt}} &= \beta' C_{Ca1} + \gamma C_0 - (4\alpha' + \omega)C_{Ca0} \\
\frac{\mathrm{d}C_{Ca1}}{\mathrm{dt}} &= a\alpha' C_{Ca0} + 2\beta' C_{Ca2} + \gamma a C_1 - (\beta' + 3\alpha' + \tfrac{\omega}{b})C_{Ca1} \\
\frac{\mathrm{d}C_{Ca2}}{\mathrm{dt}} &= 3\alpha' C_{Ca1} + 3\beta' C_{Ca3} + \gamma a^2 C_2 - (2\beta' + 2\alpha' + \tfrac{\omega}{b^2})C_{Ca2} \\
\frac{\mathrm{d}C_{Ca3}}{\mathrm{dt}} &= 2\alpha' C_{Ca2} + 4\beta' C_{Ca4} + \gamma a^3 C_3 - (3\beta' + \alpha' + \tfrac{\omega}{b^3})C_{Ca3} \\
\frac{\mathrm{d}C_{Ca4}}{\mathrm{dt}} &= \alpha' C_{Ca3} + \gamma a^4 C_4 - (4\beta' + f' + \tfrac{\omega}{b^4})C_{Ca4} \\
\frac{\mathrm{d}y}{\mathrm{dt}} &= \frac{y_\infty - y}{\tau_y}
\end{aligned}$$

A.5 Intracellular Ca^{2+} fluxes (slow buffers)

$$\begin{aligned}
\frac{\mathrm{d}[HTRPNCa]}{\mathrm{dt}} &= k^+_{htrpn}\left[Ca^{2+}\right]_i([HTRPN]_{tot} - [HTRPNCa]) - k^-_{htrpn}[HTRPNCa] \\
\frac{\mathrm{d}[LTRPNCa]}{\mathrm{dt}} &= k^+_{ltrpn}\left[Ca^{2+}\right]_i([LTRPN]_{tot} - [LTRPNCa]) - k^-_{ltrpn}[LTRPNCa]
\end{aligned}$$

A.6 Intracellular ionic concentrations

$$\begin{aligned}
\frac{\mathrm{d}[K^+]}{\mathrm{dt}} &= -(I_{Kr} + I_{Ks} + I_{to} + I_{K1} + I_{Kp} + I_{Ca,K} - 2I_{NaK})\frac{A_{cap}C_{sc}}{V_{myo}F} \\
\frac{\mathrm{d}\left[Ca^{2+}\right]_i}{\mathrm{dt}} &= \beta_i\left(J_{xfer} - J_{up} - J_{trpn} - (I_{Ca,b} - 2I_{NaCa} + I_{p(Ca)})\frac{A_{cap}C_{sc}}{2V_{myo}F}\right) \\
\frac{\mathrm{d}\left[Ca^{2+}\right]_{ss}}{\mathrm{dt}} &= \beta_{ss}(J_{rel}\frac{V_{JSR}}{V_{ss}} - J_{xfer}\frac{V_{myo}}{V_{ss}} - I_{Ca}\frac{A_{cap}C_{sc}}{2V_{myo}F}) \\
\frac{\mathrm{d}\left[Ca^{2+}\right]_{JSR}}{\mathrm{dt}} &= \beta_{JSR}(J_{tr} - J_{rel}) \\
\frac{\mathrm{d}\left[Ca^{2+}\right]_{NSR}}{\mathrm{dt}} &= J_{up}\frac{V_{myo}}{V_{NSR}} - J_{tr}\frac{V_{JSR}}{V_{NSR}}
\end{aligned}$$

The stiffness of the system is to a large extent due to $\left[Ca^{2+}\right]_{ss}$. The parameters of the equation system can be found e.g in [18].