ELSEVIER

Mathematical Biosciences 172 (2001) 55–72

Mathematical Biosciences

www.elsevier.com/locate/mbs

Efficient solution of ordinary differential equations modeling electrical activity in cardiac cells

Joakim Sundnes *, Glenn Terje Lines, Aslak Tveito

Department of Informatics, University of Oslo, P.O. Box 1080, Blindern, N-0316 Oslo, Norway

Received 4 December 2000; received in revised form 24 May 2001; accepted 22 June 2001

Abstract

The contraction of the heart is preceded and caused by a cellular electro-chemical reaction, causing an electrical field to be generated. Performing realistic computer simulations of this process involves solving a set of partial differential equations, as well as a large number of ordinary differential equations (ODEs) characterizing the reactive behavior of the cardiac tissue. Experiments have shown that the solution of the ODEs contribute significantly to the total work of a simulation, and there is thus a strong need to utilize efficient solution methods for this part of the problem. This paper presents how an efficient implicit Runge–Kutta method may be adapted to solve a complicated cardiac cell model consisting of 31 ODEs, and how this solver may be coupled to a set of PDE solvers to provide complete simulations of the electrical activity. © 2001 Published by Elsevier Science Inc.

Keywords: Bidomain model; Ionic current; Implicit ODE solvers

1. Introduction

The purpose of this paper is to study efficient solution of the bidomain model [1], given by

$$\begin{aligned} &C_\chi \frac{\partial v}{\partial t} + \chi I_{\text{ion}}(v, y) - \nabla \cdot (M_\text{i} \nabla v) = \nabla \cdot (M_\text{i} \nabla u_\text{e}), \quad x \in H, \\ &\nabla \cdot ((M_\text{i} + M_\text{e}) \nabla u_\text{e}) = -\nabla \cdot (M_\text{i} \nabla v), \quad x \in H. \end{aligned} \tag{1}$$

This system models the generation and propagation of an electrical field in the human heart. The main variables v and u_e are the transmembrane potential and the extracellular potential, respectively. Furthermore χ is a geometrical constant, C is the capacitance of the cell membrane, M_i

* Corresponding author. Tel.: +47-22 840 069; fax: +47-22 852 401.
E-mail addresses: sundnes@ifi.uio.no (J. Sundnes), glennli@ifi.uio.no (G.T. Lines), aslak@ifi.uio.no (A. Tveito).

0025-5564/01/$ - see front matter
PII: S0025-5564(01)00069-4

and M_e are conductivity tensors while the computational domain H is the heart muscle. In this paper we will focus on efficient methods for computing the state vector y, present in the term $I_{ion}(v, y)$, denoting flow of ions across the cell membrane. The elements of this vector are variables characterizing the physiological state of the cardiac cells. For each point $x \in H$, the dynamics of the vector y are described by a system of ordinary differential equations (ODEs), given by

$$\frac{dy}{dt} = f(t, y).$$

The right-hand side f is a given vector valued, non-linear function. When Eqs. (1) are discretized in space (see [2,3]), we need to solve one such ODE system for each node in the computational grid. For accurate simulations on realistic geometries, this task tends to be computationally demanding. Several ODE based models exist for the electrical behavior of cardiac cells, of various complexity and accuracy. Rogers and McCulloch [4] have used the FitzHugh–Nagumo model [5] to simulate the electrical activity of the heart. Similar simulations using the more physiologically correct Beeler–Reuter model [6] have been performed by Huiskamp [7].

In [2,3] it is described how Eqs. (1) may be coupled to a corresponding equation describing the tissue surrounding the heart muscle. The resulting equations are solved using a finite element discretization on unstructured 2D and 3D grids. Simulations have been performed using ionic current models of Luo and Rudy [8] and of Winslow et al. [9]. The model due to Winslow et al. is considered to be accurate, but it is also very complex and computationally demanding. The main focus of this paper is on the implementation of efficient solvers for this ODE system and the utilization of these solvers in the complete simulator.

The rest of this paper is organized as follows. Section 2 presents the bidomain model in more detail and describes the coupling of the equations in the heart to the equation describing the surrounding tissue. We also specify the boundary conditions and the values of the various parameters present in the equations. In Section 3 we discuss how the mathematical model may be discretized in time using an operator-splitting technique, and in space using a finite element formulation. Section 4 presents numerical results for this solution method, where a standard explicit Runge–Kutta method has been used for solving the ODE systems. A more efficient method for solving the ODE systems is presented in Section 5. In Section 6 we present numerical experiments with this new solver on single cell simulations, and compare its performance to the solver used initially. The new solver is then coupled to the solvers for the partial differential equations, and numerical results for the complete simulator are presented in Section 7. In Section 8 we summarize the results and outline possible directions of further research.

2. The mathematical model

2.1. The bidomain model

In the bidomain model the cardiac tissue is conceptually divided into the extracellular and the intracellular space, with the corresponding electrical potentials denoted u_e and u_i, respectively. These entities are interpreted as volume-averaged quantities, so that for every point $x \in H$ both

potentials are defined. The difference between the two fields is denoted the transmembrane potential:

$$v = u_{\mathrm{i}} - u_{\mathrm{e}}. \tag{2}$$

The transmembrane potential v and the extracellular potential are determined from (1). The conductivity tensors M_{i} and M_{e} present in the equation are defined in terms of the orientation of the muscle fiber ($\boldsymbol{a}_l(\boldsymbol{x})$):

$$M_{\{\mathrm{i,e}\}} = \left(\sigma_l^{\{\mathrm{i,e}\}} - \sigma_t^{\{\mathrm{i,e}\}}\right)\boldsymbol{a}_l\boldsymbol{a}_l^{\mathrm{T}} + \sigma_t^{\{\mathrm{i,e}\}} I,$$

where σ_l and σ_t are the conductivity along and across the fiber direction, respectively.

The state vector y will vary throughout the myocardium just like v and u_{e}, i.e. it will be a function of position. Typically, y is governed by an ODE for each point in space so the system for $y(t,x)$ will be on the form

$$\frac{\partial y}{\partial t}(x) = F(t, y(t,x),\ v(t,x); x) \tag{3}$$

with an initial value

$$y(0,x) = y_0(x).$$

Here F is the vector valued function defining the time derivatives of the state variables.

2.2. The ionic current

There are many possible choices for $I_{\mathrm{ion}}(v,y)$. In our study we have chosen to use the model of Winslow et al. [9] since it is well documented and accurate. The state vector contains 31 variables which include ionic concentrations and the so-called gating variables which characterize the permeability of the cell membrane for different ions. See Section A in the appendix for details.

2.3. Boundary conditions

To have a complete mathematical model, we need boundary conditions for v and u_{e}. Several choices exist [10] but we shall here use the ones suggested by Colli Franzone et al. [11]. We assume that the myocardial domain is embedded in a conductive medium which we will call the extracardiac domain and denote by D. The potential on D is denoted by u_D. This medium might be a torso or just a simple homogeneous and isotropic bath. Due to the cavities of the heart, the domain H will contain holes for physiologically correct geometries. Thus D can also represent one of these cavities depending on what part of the myocardial boundary we are considering. The computation of u_D is discussed in Section 2.4.

At the boundary between D and H the following conditions are used:

$$u_{\mathrm{e}}(x) = u_D(x), \quad x \in \partial H, \tag{4}$$

$$n \cdot M_D \nabla u_D = n \cdot (M_{\mathrm{i}} + M_{\mathrm{e}}) \nabla u_{\mathrm{e}}, \quad x \in \partial H, \tag{5}$$

$$n \cdot (M_{\mathrm{i}} \nabla v) = 0, \quad x \in \partial H. \tag{6}$$

2.4. Electrical potential outside the myocardium

The bidomain model only describes the electrical potentials in the myocardium. In order to simulate ECG measurements we also need to compute the potential distribution in the rest of the torso. The governing equation in this domain is

$$\nabla \cdot (M_D \nabla u_D) = 0, \quad x \in D. \tag{7}$$

The equation is valid for $D = L, R, T$, where L and R are the cavities of the left and the right ventricle and T is the torso excluding the heart. The topology of a horizontal cross-section of the domains is shown in Fig. 1.

At the boundary of the torso a no-flow condition is used:

$$n \cdot (M_T \nabla u_T) = 0, \quad x \in \partial D. \tag{8}$$

The coupling to the heart is obtained through the boundary conditions on the myocardial surface as stated in (4)–(6).

2.5. The parameters

The values of the parameters of the model are as follows:

$$\begin{aligned}
&\sigma_T = 2.39 \text{ mS/cm},\\
&\sigma_L = 6.0 \text{ mS/cm},\\
&\sigma_R = 6.0 \text{ mS/cm},\\
&\sigma_l^{\mathrm{i}} = 3.0 \text{ mS/cm},\\
&\sigma_l^{\mathrm{e}} = 2.0 \text{ mS/cm},\\
&\sigma_t^{\mathrm{i}} = 0.31525 \text{ mS/cm},\\
&\sigma_t^{\mathrm{e}} = 1.3514 \text{ mS/cm},\\
&\chi = 2000 \text{ cm}^{-1},\\
&C = 1 \ \mu\text{F/cm}^2.
\end{aligned} \tag{9}$$

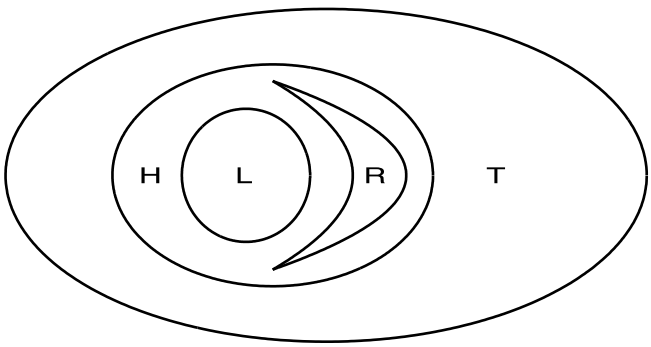

Fig. 1. A schematic horizontal cross-section of the torso and the heart. H represents the myocardium, i.e. the heart muscle. L and R represent the left and right ventricles, respectively. T represents the torso. The four domains are disjoint and their union is denoted Ω.

The conductivity parameters were found in [12], the surface area to volume ratio was found in [13] and the membrane capacitance was taken from Pollard et al. [14].

2.6. Summary of the model

The equations in L, R, T and the second equation in (1) for H all share the same structure, with the biggest difference being that the right-hand side in (1) is non-zero. Due to the continuity conditions of the extracellular and extracardiac potential as stated in (4) and (5) it is possible to replace these four equations with a single equation, cf. [2,15, p. 37]. This simplifies the implementation of the model considerably. We define Ω to be the union of all the domains, $\Omega = T \cup H \cup L \cup R$, and $u(x)$ to coincide with the different potentials on each subdomain:

$$u(x) = \begin{cases} u_T(x) & \text{if } x \in T, \\ u_e(x) & \text{if } x \in H, \\ u_L(x) & \text{if } x \in L, \\ u_R(x) & \text{if } x \in R. \end{cases}$$

The equation for u becomes

$$\begin{aligned} &\nabla \cdot (M \nabla u) = f, \quad x \in \Omega, \\ &n \cdot (M \nabla u) = 0, \quad x \in \partial\Omega \ (= \partial T), \end{aligned}$$

where

$$f(x, v) = \begin{cases} -\nabla \cdot (M_i \nabla v) & \text{if } x \in H, \\ 0 & \text{otherwise}, \end{cases} \tag{10}$$

and

$$M(x) = \begin{cases} M_T(x) & \text{if } x \in T, \\ M_i(x) + M_e(x) & \text{if } x \in H, \\ M_L(x) & \text{if } x \in L, \\ M_R(x) & \text{if } x \in R. \end{cases} \tag{11}$$

With this simplification the system (1), (6)–(8) with (3) becomes

$$\begin{aligned} H: &\quad \frac{\partial y}{\partial t} = F(t, y, v; x), \\ H: &\quad C\chi \frac{\partial v}{\partial t} + \chi I_{\text{ion}}(v, y) = \nabla \cdot (M_i \nabla (v + u)), \\ \partial H: &\quad n \cdot (M_i \nabla v) = 0, \\ \Omega: &\quad \nabla \cdot (M \nabla u) = f(x, v), \\ \partial\Omega: &\quad n \cdot (M \nabla u) = 0, \end{aligned} \tag{12}$$

with f and M defined as in (10) and (11), respectively.

3. A numerical method

3.1. Time discretization

In order to discretize (12) we introduce the time-step $t_n = n\Delta t = nT/N$, where N denotes the number of time-steps and T is the duration of the simulation. At each time-level we introduce the variables $u^n = u(t_n, x)$, $v^n = v(t_n, x)$ and $y^n = y(t_n, x)$.

The equations in (12) are a coupled system and should ideally be solved simultaneously. We have chosen to solve them sequentially thus applying a standard sequential operator splitting technique. The equations are solved in the following order:

1. Assume that u^n, v^n and y^n are known at time t_n.
2. Compute s^{n+1} by solving

$$\frac{\partial s}{\partial t} = F(t, y, v; x), \quad y(t_n, x) = y^n \quad \text{and} \quad v(t_n, x) = v^n$$

for $t \in (t_n, t_{n+1}]$.

3. Compute v^{n+1} in H by solving

$$C\chi \frac{v^{n+1} - v^n}{\Delta t} + \chi I_{\text{ion}}(v^n, y^{n+1}) = \nabla \cdot (M_{\text{i}} \nabla v^{n+1}) + \nabla \cdot (M_{\text{i}} \nabla u^n).$$

4. Compute u^{n+1} in Ω by solving

$$\nabla \cdot (M \nabla u^{n+1}) = f^{n+1},$$

where f^{n+1} is obtained by inserting v^{n+1} in (10).

In Step 2 the ODEs are integrated forward in time with internal time-steps. Initially we have used a fourth and fifth-order Runga–Kutta method, but it is the purpose of the present paper to refine this part of the method.

In Step 3 we have evaluated the ionic term by using the previous update of the transmembrane potential. In this way the reaction–diffusion equation will become linear with respect to v and thus much easier to solve. The error committed by this approximation will depend upon the size of the time-step taken. Solving with a fully implicit scheme may have permitted larger time-steps to be taken but each step would be very costly.

Numerical experiments indicate that evaluating the ionic current with the previous update of the transmembrane potential does not lead to any instability problems. In fact, this semi-implicit scheme seems to be unconditionally stable.

3.2. Spatial discretization

The PDEs on each time-step are discretized using the finite element method with linear basis functions over the elements. The linear systems are solved using multi-grid. The methods are described in [2,3,16].

4. Numerical considerations

Numerical experiments have been performed in 2D and 3D, see [2]. The boundaries were obtained by tracing slice photographs [17] and are shown for a 2D geometry in Fig. 2(a). In Fig. 2(b) the fiber directions of the heart muscle are indicated.

When solving the bidomain equations numerically three subproblems are solved. These are the ODE equations to find the ionic current in each node, and the two PDEs of the bidomain system. Solving each PDE consists of two parts: assembling and solving the linear system of equations. Thus the whole problem can be divided into five separate tasks.

Table 1 shows the time consumption for a 2D simulation. All execution times reported are obtained from simulations performed on a Pentium III 600 MHz machine. Duration of the simulation was 300 ms and the time-step was 0.125 ms. 15 000 nodes were used in H and 75 000 in Ω.

The workload of each subproblem has been ranked below. The most CPU intensive task is listed first:

1. Solving the ODEs.
2. Solving the linear system of the elliptic equation.
3. Assembly of the linear system of the elliptic equation.

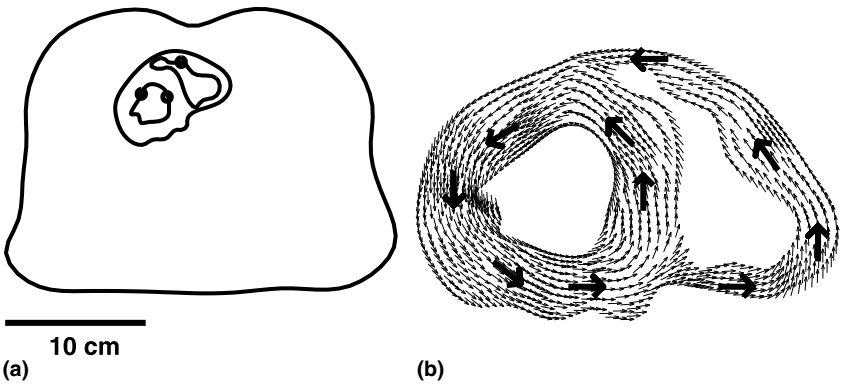

Fig. 2. (a) The boundary of the myocardium and the torso. The dots indicate the activation sites. (b) The fiber directions used in the model. The basis directions are shown with the large arrows while the interpolations used to compute the conductivity tensors are shown with small arrows.

Table 1
Execution times in seconds for the simulations with the anatomical 2D geometry

Task	CPU time
Solving the ODEs	124 432
Assembly of linear system, parabolic equation	3712
Solving the linear system, parabolic equation	1170
Assembly of linear system, elliptic equation	253
Solving the linear system, elliptic equation	14 819

The number of nodes in the myocardium is 15 607, and the number of nodes in the entire torso (including the myocardium) is 73 313. Simulation was performed in 300 ms in steps of 0.125 ms, giving a total of 2400 time-steps.

4. Assembly of the linear system of the parabolic equation.
5. Solving the linear system of the parabolic equation.

We see that for this problem size, solving the ODEs is by far the most CPU intensive part of the problem. Because we use efficient multi-grid algorithms to solve the linear systems, the workload of each task is proportional to the number of unknowns. This means that the solution of the ODEs will be the dominant task for all problem sizes, and it is thus the obvious place to look for potential efficiency improvements.

5. Efficient ODE-solvers

As mentioned in the previous section, the most CPU intensive part of the heart simulations is the solution of the ODE systems having the form

$$\frac{\partial y}{\partial t} = F(t, y, v; x).$$

We have also seen that because of the coupling between the ODEs and the PDEs, the ODEs are integrated in small intervals, governed by the time-step of the PDE solvers. We thus want an ODE solver capable of solving efficiently general initial value problems of the form

$$y' = f(t, y), \quad t \in (T_n, T_{n+1}], \quad y(T_n) = y_n,$$

where $T_{n+1} - T_n$ is very small. Because of stability problems the explicit Runge–Kutta methods used in the original simulator typically required about 20–30 time-steps to integrate one such interval $[T_n, T_{n+1}]$. While it is hoped that using more stable methods will significantly reduce this number, it is likely that at least some of the intervals will still require a few internal steps, since some state variables change very rapidly.

Several approaches were tested in order to improve the efficiency of solving the ODE systems. Realizing that most of the individual equations in the ODE system are in fact linear, we tested several methods based on dividing the equations into two groups, and solving the respective linear and non-linear equation-systems sequentially. Combining this approach with various implicit schemes significantly improved the stability when compared to the explicit Runge–Kutta methods used originally. However, experiments revealed that the accuracy of the sequential method was insufficient for our purposes. A detailed description of this solution procedure, as well as the results from the experiments, is found in [18].

5.1. Implicit Runge–Kutta methods

Considering the poor stability properties observed with explicit solvers and the large errors introduced by the sequential approach, our focus turned to fully implicit methods. Analysis presented in [19,20] indicates that a special class of methods known as SDIRK-methods, or singly diagonally implicit Runge–Kutta methods, are well suited for our type of problem. This was confirmed by our initial tests, where methods of this type were compared to fully implicit Runge–Kutta methods and traditional multi-step methods for stiff problems. Based on an overall consideration of computational speed, accuracy and memory requirements when coupled to the PDE

solver, we chose to implement a 4-stage pair of embedded methods of order 3(2), originally presented in [21]. Despite the relatively low order of accuracy, the error in the solution of the ODE systems is small compared to the error arising from the discretization of the PDEs. A general implementation of this method is found in the ODE library Godess (see [20]). As described above our applications require frequent stopping and restarting of the ODE-solver to exchange data with the PDE-solver. This makes it difficult to achieve the desired efficiency by using Godess directly, and calls for a specialized implementation of the SDIRK-method.

The main strength of the SDIRK method we have implemented, referred to as ESDIRK23A (see [20]), is that the third-order main method and the embedded second-order error estimating method are both stiffly accurate (see [19]). Thus no order reduction will occur when applied to stiff problems. This could otherwise cause problems in the adaptive time-stepping algorithm, which is based on the known analytical order of both methods. The Butcher tableau defining the method ESDIRK23A is shown in Table 2, with the actual values of the coefficients specified in Appendix B.

Applying this implicit scheme to a non-linear system like the present cell model involves solving systems of non-linear algebraic equations for each time-step. For fully implicit Runge–Kutta methods, with Butcher tableaus as defined in Table 3, we generally have all coefficients $a_{ij} \neq 0$. These methods require solving a system of $s \cdot n$ non-linear algebraic equations for each time-step, where s is the number of stages in the method and n is the number of differential equations in the system. For the ESDIRK23A-method, where we have $a_{ij} = 0$ for $j > i$, the stages may be treated sequentially. Following the conventional notation for Runge–Kutta methods, we denote by $k_1, \ldots, k_4$ the internal stage derivatives of one time-step. Integrating the equations one step, from $t = t_n$ to $t = t_{n+1}$, involves the following operations:

1. Compute the first stage derivative (explicitly) from

$$k_1 = f(t_n, y_n).$$

2. Find k_2, k_3 and k_4 by solving the following non-linear algebraic equations:

$$k_2 = f(t_n + \Delta t c_2, y_n + \Delta t(a_{21}k_1 + \gamma k_2)), \tag{13}$$

$$k_3 = f(t_n + \Delta t c_3, y_n + \Delta t(\hat{b}_1 k_1 + \hat{b}_2 k_2 + \gamma k_3)), \tag{14}$$

$$k_4 = f(t_n + \Delta t c_4, yn + \Delta t(b_1 k_1 + b_2 k_2 + b_3 k_3 + \gamma k_4)). \tag{15}$$

3. Compute the solution y_{n+1} and the error estimate e_{n+1} from

$$y_{n+1} = y_n + \sum_{i=1}^{4} b_i k_i$$

Table 2
Butcher tableau for the Runge–Kutta pair ESDIRK23A

0	0			
c_2	a_{21}	γ		
c_3	$\hat{b}_1$	$\hat{b}_2$	γ	
c_4	b_1	b_2	b_3	γ
	$\hat{b}_1$	$\hat{b}_2$	γ	
	b_1	b_2	b_3	γ

Table 3
Butcher tableau for a general s-stage implicit Runge–Kutta method, using an embedded method for error estimation

c_1	a_{11}	a_{12}	$\cdots$	a_{1s}
c_2	a_{21}	a_{22}	$\cdots$	a_{2s}
c_3	$\vdots$	$\vdots$		$\vdots$
c_4	a_{s1}	a_{s2}	$\cdots$	a_{s4}
	$\hat{b}_1$	$\hat{b}_2$	$\cdots$	$\hat{b}_s$
	b_1	b_2	$\cdots$	b_s

and

$$e_{n+1} = \sum_{i=1}^{4} (b_i - \hat{b}_i) k_i.$$

The standard procedure for solving the systems of Eqs. (13)–(15) is by Newton iterations. This involves, after computing a starting value k_i^0, iterations of the form

$$(I - \Delta t \gamma J)\Delta k_i^m = -k^m + f\left(t_n + \Delta t c_i, y_n + \Delta t \sum_{j=1}^{i-1} a_{ij} k_i^m + \gamma k_i^m\right), \tag{16}$$

$$k_i^{m+1} = k_i^m + \Delta k_i^m \tag{17}$$

for $i = 2, 3$ and 4 until the termination criterion is fulfilled. The matrix J is the Jacobian of the right-hand side function f. For standard Newton iterations, this matrix is recomputed for every iteration, and it is given by

$$J = \frac{\partial f}{\partial y}(t_n + \Delta t c_i, y_n + r\Delta t), \tag{18}$$

where r is a collection of terms from Eq. (16). For our application, recomputing the Jacobian for each iteration is too costly, and it is instead approximated by

$$J \approx \frac{\partial f}{\partial y}(t_n, y_n). \tag{19}$$

We see that if Δt is chosen sufficiently small, (19) is a good approximation to (18). In the implementation of the solver this is taken into account by reducing the time-step if the Newton iterations converge slowly, indicating that the Jacobian is not adequately approximated.

When the approximation (19) is used, the Jacobian is the same for all the stages within one time-step. Since Δt and γ are also constant, the matrix of the linear system, $(I - \Delta t \gamma J)$, is the same for all stages. This allows for significant efficiency improvements, because the LU-factorization of this matrix may be reused for all the stages within one time-step. Further improvements are obtained by reusing the factorized matrix for several consecutive time-steps, only recomputing and factorizing the matrix when the time-step is changed significantly or the convergence of the Newton iterations becomes too slow. Initial experiments confirmed that it was indeed possible to keep the Jacobian for several steps while still obtaining good convergence of the Newton iterations, and this proved to be crucial to obtain the desired efficiency. If the Jacobian was recomputed for every time-step, the SDIRK method offered only a slight performance improvement

compared to explicit Runge–Kutta solvers. The criteria used for deciding when the Jacobian is to be recomputed, as well as other implementational details, are described in [18].

As mentioned above, using sequential solution methods to solve the ODE system results in most of the individual (scalar) equations becoming linear. In fact we get 25 linear and only six non-linear equations. This is of course also the case for the algebraic equations (13)–(15), and this may be considered when choosing the solution procedure. We experimented with standard Gauss–Seidel iterations for this system, allowing the use of an explicit formula for updating the linear equations. For the non-linear equations we still need to use Newton's method, but since we are now considering scalar equations there is no need for computing a Jacobian matrix and solving linear systems for each iteration. Solving the non-linear system then involves, after computing a starting value for all variables, iterations of the form:

1. update the linear equations from an explicit formula, using the most recently obtained value for all other variables;
2. solve the non-linear equations one by one with Newton's method.

These iterations are repeated until the termination criterion is fulfilled. In Section 6 the performance of this method is compared to standard simplified Newton iterations.

6. Numerical results for single cell experiments

This section describes the results of numerical experiments with different solvers and different choices of parameters, applied to the present ODE-system, as defined in Appendix A. Although in these experiments the ODE-solvers are not coupled to PDEs, the restrictions on the time-step induced by the PDE solvers are included in the simulations. This means that the maximum time-step for every solver is 0.125 ms. The effect of this restriction is of course most significant for the SDIRK-method, because the good stability properties of this method would otherwise allow it to proceed with much larger steps. For explicit methods, the time-step restrictions imposed by stability requirements are so strict that the restriction from the PDEs hardly affects the performance.

Table 4 shows the performance of the ESDIRK23A solver compared to an explicit Runge–Kutta method. These results are from a fairly standard version, capable of solving general ODE systems of the form $y' = f(t, y)$. Simplified Newton iterations are used to solve the non-linear equations. The recorded CPU times include only the work required to integrate the equations. We do not include the time required to store the solution to a file, since this is not relevant for our final application. The other entries in the table include the pre-specified local tolerance *tol*, the total

Table 4
Numerical results for the method ESDIRK23A compared to an explicit Runge–Kutta method

Method	tol	e_{global}	Steps	CPU
Explicit	1.0×10^{-4}	0.0462	42 662	6.49
SDIRK	1.0×10^{-4}	0.0457	2688	1.00
SDIRK	1.0×10^{-3}	0.4457	2533	0.86

The CPU times listed are in seconds.

Table 5
Results from the ESDIRK23A method using Gauss–Seidel iterations to solve the non-linear equation, compared to a version using simplified Newton iterations

Method	tol	e_{global}	Steps	Iterations	CPU
Newton	1.0×10^{-4}	0.0457	2688	12 108	1.00
GS	1.0×10^{-4}	0.0421	2681	20 708	2.05

number of time-steps used and the L^∞-norm of the global error in V, e_{global}. This error is given by the expression

$$e_{\text{global}} = \max(V - V_{\text{ref}}),$$

where V_{ref} is a reference solution computed with a fifth-order Runge–Kutta scheme and approximately 160 000 time-steps. The difference is evaluated at every global time-step.

Table 5 shows the performance of the ESDIRK23A method where the non-linear equations are solved with Gauss–Seidel iterations, compared to the standard version with simplified Newton iterations. We see that the total number of iterations required is nearly doubled when replacing the simplified Newton iterations with Gauss–Seidel. We also see that the execution time is more than doubled when using Gauss–Seidel iterations, indicating that one such iteration is more costly than a simplified Newton iteration.

6.1. Further optimizations

The results obtained with a general implementation of the SDIRK method, displayed in the previous section, indicate that the SDIRK method offers substantial improvements over explicit methods regarding CPU time. For our applications, where computational efficiency is crucial, we wanted to investigate even further optimizations of the method. A simple profiling of the program reveals that the dominating operations are as follows:

1. Evaluation of the right-hand side function f, done once for each Newton iteration.
2. The forward–backward substitution used to solve the linear system in each Newton iteration.
3. Recomputing the Jacobian matrix.

We have already put some effort into reducing the number of Newton iterations required by the solver, and thereby the number of evaluations of the right-hand side function f. Initial experiments revealed that there is also a limited potential for optimizing the function itself. This indicates that the main candidates for further optimization are items two and three on the list. Focusing first on the recomputing of the Jacobian matrix, a closer examination of this function reveals that it consists of 32 calls to the right-hand side function f. This is because we compute the derivatives by differences, from the simple formula

$$J_{ij} = \frac{f_i(t, y + hI_j) - f_i(t, y)}{h},$$

where $I_j(i) = 0$ for $i \neq j$ and $I_j(j) = 1$. We see that we need one evaluation of $f(t, y)$ and 31 evaluations of $f(t, y + hI_j)$, one for each variation of I_j. A more thorough examination of the Jacobian matrix reveals that it is sparse, with 128 out of the total 961 entries being non-zero. Since most of the couplings between variables are linear, it is also fairly easy to derive analytical

Table 6
Computation time for the ESDIRK23A solver for different levels of optimization

Level of optimization	Iterations	CPU
None	12 108	1.00
Jacobian only	12 108	0.72
Full	12 108	0.66

expressions for most of the entries in the matrix. Considering these facts, we designed a specialized function for evaluating the Jacobian. Working only on the non-zero entries, and computing most of the entries analytically from simple expressions, we managed to reduce the work for one Jacobian evaluation by approximately 80%.

Turning to item two on the list, an investigation of the structure of the LU-factorized matrix revealed a large potential for optimization. Even though the LU-factorization alters the sparsity pattern of the Jacobian matrix, the matrix is still sparse, with less than 25% of the elements being non-zero. By designing a new function for the forward–backward substitution, working only on the non-zero elements, the work for this process was reduced by approximately 50%.

Table 6 shows the total CPU time of the optimized code compared to the general code. We see that the reduction is significant, at least for the optimization of the Jacobian evaluation. Since computational efficiency is crucial for our applications we believe that these gains outweigh the loss of generality introduced by the optimized functions.

The results displayed in this section indeed show that the SDIRK method is far more efficient than explicit methods for solving the present ODE system. Significant improvements were obtained with a fairly general implementation, and by optimizing the solver even further the total reduction in execution time was close to 90%.

7. Numerical results for the global simulator

The promising results presented in the previous section all related to simulations of the behavior of a single cardiac cell. It still remains to be verified whether similar improvements may be achieved when the ODE solver is coupled to the global heart simulator. The coupling of the implicit Runge–Kutta solver to the PDE solvers is somewhat more complicated than for the explicit solver, mostly because of considerations regarding recomputing and factorizing the Jacobian matrix. For single cell experiments we were able to reuse the factorized Jacobian for several consecutive time-steps, and this proved to be crucial to achieve good efficiency. Implementing this strategy for the global simulator would require storing the Jacobian for each nodal point in the computational grid. The number of non-zero elements in the factorized Jacobian is 216, and the large number of nodes obviously prohibits the use of this strategy. The most obvious solution to this problem is to recompute the Jacobian for each node for every 'global' time-step. This approach requires a large number of Jacobian evaluations, but the total number of Newton iterations required is likely to be relatively low. In Table 7 CPU times for this approach are compared to results from explicit Runge–Kutta methods. The results presented are for a simulation from $t = 0$ to $t = 300$ ms, with a time-step of 0.125 ms and 15 607 nodes in the myocardium.

Table 7
Numerical results for the approach where we recompute the Jacobian for each time-step, compared to an explicit Runge–Kutta method

	CPU total	CPU forward
Explicit	144 417	124 432
SDIRK	83 134	64 076

The execution times listed are in seconds.

It is likely that the approach described above involves a large number of unnecessary evaluations and factorizations of the Jacobian matrix. Particularly in the beginning stage of a simulation there will be a large number of nodes that does not change at all from one time-step to another. Recomputing the Jacobian for each global step then seems like a waste. To avoid this we propose to reuse the factorized Jacobian matrix from the 'previous' nodal point. Theoretically this approach may seem rather dubious, as we actually base our Newton iterations on the Jacobian of a system that is possibly in an entirely different state. In practice however, the spacing between the nodes in the grid is very small, and the state of a node is thus likely to be very similar to the neighboring nodes. Numerical results for this approach are shown in Table 8, where we see that this method is significantly more efficient than recomputing the Jacobian for each nodal point. Although we did experience an increase in the number of rejected steps and required Newton iterations, the reduced number of Jacobian evaluations seems to outweigh these costs by a large margin.

Table 9 shows the time consumption for the different parts of the global simulator, when the SDIRK method is used for solving the ODEs. We see that solving the ODEs is still the most time consuming task, but it is not nearly as dominating as it was when explicit ODE solvers were used.

Table 8
CPU times in seconds for the method where the Jacobian is reused from the previous nodal point, compared to the original explicit method

	CPU total	CPU forward
Explicit	144 417	124 432
SDIRK with Jacobian reuse	37 309	17 470

Table 9
Execution times in seconds for simulations with the anatomical 2D geometry, comparing the CPU times of the explicit ODE solver to the SDIRK solver

Task	Explicit	SDIRK
Solving the ODEs	124 432	17 470
Assembly of linear system, parabolic equation	3712	3647
Solving the linear system, parabolic equation	1170	1163
Assembly of linear system, elliptic equation	253	236
Solving the linear system, elliptic equation	14 819	14 712

The number of nodes in the myocardium is 15 607, and the number of nodes in the entire torso (including the myocardium) is 73 313. Simulation was performed in 300 ms in steps of 0.125 ms, giving a total of 2400 time-steps.

8. Conclusion

We have seen that the use of an SDIRK method to solve the ODEs was highly efficient in terms of reducing the total execution time of the simulator. Avoiding the stability problems of the explicit methods, the CPU time required for solving the ODEs was reduced by approximately 85%. The CPU time reduction for the total simulator was about 75%. When explicit ODE solvers were used the solution of the ODEs dominated the work of a simulation, contributing more than 85% to the total execution time. With the SDIRK method the contribution from the ODEs was reduced to less than 50%. One interesting conclusion from the present work is that it was relatively straightforward to couple the implicit ODE solver to the global simulator. Using the Jacobian from the previous nodal point for our Newton iterations did not cause any problems, even for the fairly coarse grids that were used in the present work. We observed a small increase in the total number of Newton iterations, but the added work was not significant. For simulations on finer grids, with shorter distance between the nodes, the increase in required iterations is likely to be even smaller. Possible future work in this area will probably focus on the coupling between the ODEs and the PDEs in the global problem, rather than on the ODE solver itself. Although we are able to produce good results today it is not obvious that the strategy we are using is optimal. Improvements in this area may allow a better utilization of the ODE solvers, and consequently lead to even further reductions in the CPU time for the global simulator.

Appendix A. The ODE system

This section lists the differential equations present in the cell model. The model consists of 33 ODEs, but two of the equations have very little effect on the system and may thus be neglected. The remaining 31 equations are given below. A complete description of the model, including specification of coefficient functions and constants, is given in [9].

The transmembrane potential

$$\frac{dV}{dt} = -(I_{Na} + I_{Ca} + I_{Ca,K} + I_{Kr} + I_{Ks} + I_{to} + I_{K1} + I_{Kp} + I_{NaCa} + I_{NaK} + I_{p(Ca)} + I_{Ca,b} + I_{Na,b}).$$

The ionic currents are in general non-linear functions of V.

K^+- and Na^--gates

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m m,$$
$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h h,$$
$$\frac{dj}{dt} = \alpha_j(1-j) - \beta_j j,$$
$$\frac{dX_{Kr}}{dt} = \frac{(X_{Kr}^{\infty} - X_{Kr})}{\tau_{X_{Kr}}},$$

$$\frac{dX_{Ks}}{dt} = \frac{(X_{Ks}^{\infty} - X_{Ks})}{\tau_{X_{Ks}}},$$
$$\frac{dX_{to}}{dt} = \alpha_{X_{to}}(1 - X_{to}) - \beta_{X_{to}} X_{to},$$
$$\frac{dY_{to}}{dt} = \alpha_{Y_{to}}(1 - Y_{to}) - \beta_{Y_{to}} Y_{to}.$$

All coefficients are functions of the transmembrane potential only.

Intracellular ion concentrations

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

The ionic currents and the coefficient functions β_i, β_{ss} and β_{NSR} are non-linear functions, making all these equations non-linear.

RyR-channel

$$\frac{dP_{C_1}}{dt} = -k_a^+[Ca^{2+}]_{ss}^n P_{C_1} + k_a^- P_{O_1},$$
$$\frac{dP_{O_1}}{dt} = k_a^+[Ca^{2+}]_{ss}^n P_{C_1} - k_a^- P_{O_1} - k_b^+[Ca^{2+}]_{ss}^m P_{O_1} + k_b^- P_{O_2} - k_c^+ P_{O_1} + k_c^- P_{C_2},$$
$$\frac{dP_{O_2}}{dt} = k_b^+[Ca^{2+}]_{ss}^m P_{O_1} - k_b^- P_{O_2},$$
$$\frac{dP_{C_2}}{dt} = k_c^+ P_{O_1} - k_c^- P_{C_2},$$

where all the k-coefficients are constants.

L-type Ca^{2+}-channel

$$\frac{dC_0}{dt} = \beta C_1 + \omega C_{Ca0} - (4\alpha + \gamma)C_0,$$
$$\frac{dC_1}{dt} = 4\alpha C_0 + 2\beta C_2 + \frac{\omega}{b}C_{Ca1} - (\beta + 3\alpha + \gamma a)C_1,$$
$$\frac{dC_2}{dt} = 3\alpha C_1 + 3\beta C_3 + \frac{\omega}{b^2}C_{Ca2} - (2\beta + 2\alpha + \gamma a^2)C_2,$$

$$\frac{dC_3}{dt} = 2\alpha C_2 + 4\beta C_4 + \frac{\omega}{b^3}C_{Ca3} - (3\beta + \alpha + \gamma a^3)C_3,$$
$$\frac{dC_4}{dt} = \alpha C_3 + gO + \frac{\omega}{b^4}C_{Ca4} - (4\beta + f + \gamma a^4)C_4,$$
$$\frac{dO}{dt} = fC_4 - gO,$$
$$\frac{dC_{CaO}}{dt} = \beta' C_{Ca1} + \gamma C_0 - (4\alpha' + \omega)C_{Ca0},$$
$$\frac{dC_{Ca1}}{dt} = a\alpha' C_{Ca0} + 2\beta' C_{Ca2} + \gamma a C_1 - \left(\beta' + 3\alpha'\frac{\omega}{b}\right)C_{Ca1},$$
$$\frac{dC_{Ca2}}{dt} = 3\alpha' C_{Ca1} + 3\beta' C_{Ca3} + \gamma a^2 C_2 - \left(2\beta' + 2\alpha'\frac{\omega}{b^2}\right)C_{Ca2},$$
$$\frac{dC_{Ca3}}{dt} = 2\alpha' C_{Ca2} + 4\beta' C_{Ca4} + \gamma a^3 C_3 - \left(3\beta' + \alpha' + \frac{\omega}{b^3}\right)C_{Ca3},$$
$$\frac{dC_{Ca4}}{dt} = \alpha' C_{Ca3} + \gamma a^4 C_4 - \left(4\beta' + f' + \frac{\omega}{b^4}\right)C_{Ca4},$$
$$\frac{dy}{dt} = \frac{y_\infty - y}{\tau_y}.$$

The coefficients α, β, α' and β' are functions of the transmembrane potential V. The parameter γ, describing the transition from state normal to state Ca, is a function of the myoplasm subspace Ca^{2+} concentration $[Ca^{2+}]_{ss}$. The other coefficients are constants.

Intracellular Ca^{2+} fluxes

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

where all the k-coefficients are constants.

Appendix B. Coefficients for the Runge–Kutta method ESDIRK23A

The values of the coefficients in Table 2 are as follows:

$$c_2 = 0.87173304301691799884,$$
$$c_3 = 1.0,$$
$$c_4 = 1.0,$$
$$\gamma = 0.43586652150845899942,$$
$$a_{21} = 0.43586652150845899942,$$
$$\hat{b}_1 = 0.49056338842178057060,$$
$$\hat{b}_2 = 0.073570090069760429950,$$
$$b_1 = 0.30880996997674652335,$$
$$b_2 = 1.4905633884217805707,$$
$$b_3 = -1.2352398799069860932.$$

References

[1] D.B. Geselowitz, W.T. Miller, A bidomain model for anisotropic cardiac muscle, Ann. Biomed. Eng. 11 (1983) 191.
[2] G.T. Lines, Modeling the electrical activity of the heart – a bidomain model of the ventricles embedded in a torso, Doctoral thesis, No. 13, Department of Informatics, University of Oslo, 1999. URL: http://www.ifi.uio.no/~glennli/thesis.pdf.
[3] G.T. Lines, P. Grøttum, A. Tveito, Modeling the electrical activity of the heart – a bidomain model of the ventricles embedded in a torso, Research Report, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2000. URL:http://www.ifi.uio.no/~glennli/heart.pdf.
[4] J.M. Rogers, A.D. McCulloch, A collocation-galerkin FEM of cardiac action potential propagation, IEEE Trans. Biomed. Eng. 41 (1994) 743.
[5] R.A. FitzHugh, Impulses and physiological states in theoretical models of nerve membrane, Biophys. J. 1 (1961) 445.
[6] G.W. Beeler, H. Reuter, Reconstruction of the action potential of ventricular myocardial fibres, J. Physiol. 268 (1977) 177.
[7] G. Huiskamp, Simulation of depolarization in a membrane-equation-based model of the anisotropic ventricle, Trans. Biomed. Eng. 45 (1998) 847.
[8] C.H. Luo, Y. Rudy, A dynamic model of the cardiac ventricular action potential, Circ. Res. 74 (1994) 1071.
[9] R.L. Winslow, J. Rice, S. Jafri, E. Marban, B. O'Rourke, Mechanisms of altered excitation–contraction coupling in Canine tachycardia-induced heart failure, II. Model studies, Circ. Res. 84 (1999) 571.
[10] C.S. Henriquez, Simulating the electrical behavior of cardiac tissue using the bidomain model, Crit. Rev. Biomed. Eng. 21 (1993) 1.
[11] P. Colli Franzone, L. Guerri, S. Rovida, Wavefront propagation in an activation model of the anisotropic cardiac tissue: asymptotic analysis and numerical simulations, J. Math. Biol. 28 (1990) 121.
[12] R.N. Klepfer, C.R. Johnson, R.S. Macleod, The effects on inhomogeneities and anisotropics on electrocardiographic fields: a 3D FE study, IEEE Trans. Biomed. Eng. 44 (1997) 706.
[13] C.S. Henriquez, A.L. Muzikant, K. Smoak, Anisotropi, fibre curvature and bath loading effects on activation in thin and thick cardiac tissue preparations, J. Cardiovascular Electrophysiol. 7 (1996) 424.
[14] A.E. Pollard, N. Hooke, C.S. Henriquez, Cardiac propagation simulation, Crit. Rev. Biomed. Eng. 20 (1992) 171.
[15] W. Hackbusch, Elliptic Differential Equations, Springer, Berlin, 1992.
[16] H.P. Langtangen, Computational Partial Differential Equations Numerical Methods and Diffpack Programming, Springer, Berlin, 1999.
[17] The Visible Human Project, see http://www.nlm.nih.gov/research/visible/visible_human.html.
[18] J. Sundnes, G.T. Lines, A. Tveito, An investigation of different solvers for stiff ODE systems, Research Report, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2000. http://www.ifi.uio.no/~sundnes/report.pdf.
[19] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II. Stiff and Differential Algebra Equations, Springer, Berlin, 1991.
[20] H. Olsson, Runge–Kutta solution of initial value problems, doctoral thesis, Department of Computer Science, Lund Institute of Technology, Lund University, 1999.
[21] A. Kværnø, More, and to be hoped, better DIRK methods for the solution of stiff ODEs, Mathematical Sciences Division, Norwegian Institute of Technology, Trondheim, 1992 (unpublished).