From PDE to Software: Numerical techniques for the bidomain model

Outline

- The model
- Possible strategies.
- Operator splitting.
- Solving systems of ODEs.
- Solving the PDEs.

The model

$$
\begin{array}{rcll}
\frac{\partial s}{\partial t} & = & F(v,s) & x \in H \\
\chi C_m \frac{\partial v}{\partial t} + \chi I_{\text{ion}}(v,s) & = & \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) & x \in H \\
\nabla \cdot ((M_i + M_e)\nabla u_e) & = & -\nabla \cdot (M_i \nabla v) & x \in H
\end{array}
$$

$$
\begin{array}{rcll}
M_i \nabla v \cdot n^T & = & 0 & x \in \partial H \\
u_e & = & u_o & x \in \partial H \\
M_e \nabla u_e \cdot n^T & = & M_o \nabla u_o \cdot n^T & x \in \partial H
\end{array}
$$

$$
\begin{array}{rcll}
\nabla \cdot (M_o \nabla u_o) & = & 0 & x \in T \\
(M_o \nabla u_o) \cdot n^T & = & 0 & x \in \partial T
\end{array}
$$

Challenges

- Highly complex system of PDEs and ODEs.
- Difficult to construct efficient solution algorithms.
- Simulation software tends to be complex and difficult to maintain.

Coping with the complexity

- Several possible strategies exist for solving the bidomain equations.
- Using explicit schemes for the ODEs and the PDE involving the time derivative.
- Fully coupled approach, i.e. using a fully implicit time discretization to integrate the complete system simultaneously.

Explicit schemes

$$\begin{aligned}
\frac{s^{n+1}-s^n}{\Delta t} &= F(v^n, s^n) && x \in H \\
\chi C_m \frac{v^{n+1}-v^n}{\Delta t} + \chi I_{\text{ion}}(v^n, s^n) &= \nabla \cdot (M_i \nabla v^n) + \nabla \cdot (M_i \nabla u_e^n) && x \in H \\
\nabla \cdot ((M_i + M_e)\nabla u_e^{n+1}) &= -\nabla \cdot (M_i \nabla v^n) && x \in H
\end{aligned}$$

Problem with the explicit approach: severe restrictions on the time step, and the second PDE can not be solved explicitly.

“Solution”: reduction to a monodomain model:

$$\begin{aligned} \frac{s^{n+1}-s^n}{\Delta t} &= F(v^n, s^n) & x \in H \\ \chi C_m \frac{v^{n+1}-v^n}{\Delta t} + \chi I_{\text{ion}}(v^n, s^n) &= \nabla \cdot (M_i \nabla v^n) & x \in H \end{aligned}$$

Fully explicit solution scheme, but the approach still suffers from severe time step restrictions.

Fully implicit solution

$$\begin{aligned}
\frac{s^{n+1}-s^n}{\Delta t} &= F(v^{n+1}, s^{n+1}) \\
\chi C_m \frac{v^{n+1}-v^n}{\Delta t} + \chi I_{\text{ion}}(v^{n+1}, s^{n+1}) &= \nabla\cdot(M_i \nabla v^{n+1}) + \nabla\cdot(M_i \nabla u_e^{n+1}) \\
\nabla\cdot((M_i + M_e)\nabla u_e^{n+1}) &= -\nabla\cdot(M_i \nabla v^{n+1})
\end{aligned}$$

A fully implicit discretization avoids the strict stability restrictions on the time step, but requires solution of large systems of non-linear equations for each time step.

Operator splitting

- An attractive approach for handling the complex equations is to use some form of operator splitting.
- The complexity of the problem may be significantly reduced, without sacrificing too much in accuracy.
- Several approaches exist, but we will focus on a type of methods often referred to as *fractional step methods*.

A simplified problem

To introduce the ideas of operator splitting, we consider an initial value problem on the form

$$\begin{aligned} u'(t) &= (L_1 + L_2)u, \\ u(0) &= u_0, \end{aligned} \tag{1}$$

where L_1 and L_2 are differential operators.

For a given Δt, an approximate solution at $t = \Delta t$ may be found by first solving

$$\begin{aligned} v'(t) &= L_1 v, 0 < t \leq \Delta t \\ v(0) &= u_0, \end{aligned}$$

and then

$$\begin{aligned} w'(t) &= L_2 w, 0 < t < \Delta t \\ w_0 &= v(\Delta t), \end{aligned}$$

and finally set $u(\Delta t) = w(\Delta t)$.

- While it may seem that we have integrated the equation a step of length $2\Delta t$, we have only included parts of the differential operator in each step.
- It is natural that the procedure of solving the equation in two steps introduces an error.
- This “splitting error” may be computed by comparing the Taylor expansion of the solution of the original equation and the solution obtained with the two-step approach.

In the Taylor expansion we use that

$$u_t = (L_1 + L_2)u\,.$$

If L_1 and L_2 do not depend explicitly on t we can differentiate both sides of the equation to obtain

$$u_{tt} = (L_1 + L_2)u_t = (L_1 + L_2)^2 u$$

In general we have

$$\partial_t^n u = (L_1 + L_2)^n u$$

The solution of the original equation at time Δt may now be written

$$\begin{aligned} u(\Delta t) &= u(0) + \Delta t(L_1 + L_2)u(0) + \frac{1}{2}\Delta t^2 (L_1 + L_2)^2 u(0) + \ldots \\ &= (I + \Delta t(L_1 + L_2) + \frac{1}{2}\Delta t^2 (L_1 + L_2)^2 + \ldots)u(0) \end{aligned}$$

Similarly, the solution obtained with the splitting method may be written as

$$\begin{aligned}\tilde{u}(\Delta t) &= (I + \Delta t L_2 + \frac{1}{2}\Delta t^2 L_2^2 + \ldots) \\ &\quad (I + \Delta t L_1 + \frac{1}{2}\Delta t^2 L_1^2 + \ldots)u(0) \\ &= (I + \Delta t(L_1 + L_2) + \frac{1}{2}\Delta t^2(L_1^2 + 2L_1L_2 + L_2^2) + \ldots)u(0)\end{aligned}$$

Subtracting $\tilde{u}$ from u, we get

$$\tilde{u}(\Delta t) - u(\Delta t) = \frac{1}{2}\Delta t^2(L_1L_2 - L_2L_1)u(0) + O(\Delta t^3)$$

We see that the splitting introduces an error proportional to Δt^2 for each time step, and after n steps this is expected to accumulate to an $O(\Delta t)$ error.

- The described technique gives a first order approximation to u at any given time step $t_n = n\Delta t$.
- The technique is called Godunov splitting.

A slight modification of the Godunov algorithm is to solve the equation with the three-step algorithm

$$\begin{aligned} v'(t) &= L_1 v, 0 < t \leq \Delta t/2 \\ v(0) &= u_0, \end{aligned}$$

$$\begin{aligned} w'(t) &= L_2 w, 0 < t \leq \Delta t \\ w(0) &= v(\Delta t/2), \end{aligned}$$

$$\begin{aligned} v'(t) &= L_1 v, \Delta t/2 < t \leq \Delta t \\ v(0) &= w(\Delta t), \end{aligned}$$

and finally set $u(\Delta t) = v(\Delta t)$.

- By comparing the Taylor series of the solutions, it is now possible to show that the $O(\Delta t^2)$ terms now cancel, giving an error of $O(\Delta t^3)$ for each step.
- After $n \approx 1/\Delta t$ steps, the accumulated error is $O(\Delta t^2)$.
- This procedure is called Strang splitting.

Op. splitting for the bidomain model

- We consider the PDE containing the time derivative:

$$\chi C_m \frac{\partial v}{\partial t} + \chi I_{\text{ion}}(v,s) \quad = \quad \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) \quad x \in H$$

- In simplified form, this equation may be written as

$$\frac{\partial v}{\partial t} = (L_1 + L_2)v$$

- L_1 is a linear PDE-operator, while L_2 corresponds to the ionic current term $I_{\text{ion}}(s,v)$, and is a non-linear ODE-operator.

Inserting these expressions into the Strang splitting scheme, we get a three-step algorithm:

Step 1. Solve

$$\frac{\partial v}{\partial t} = -\frac{1}{C_m} I_{\text{ion}}(v,s), \quad t_n < t_n \leq t_n + \Delta t/2.$$

Step 2. Use the obtained approximation to $v(t_n + \Delta t/2)$ as initial condition, and solve

$$\chi C_m \frac{\partial v}{\partial t} = \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e), \quad t_n < t \leq t_n + \Delta t\,.$$

Step 3. Finally, use the new value of v as initial condition, and solve

$$\frac{\partial v}{\partial t} = -\frac{1}{C_m} I_{\text{ion}}(v,s), \quad t_n + \Delta t/2 < t \leq t + \Delta t\,.$$

- We see that the equation is solved by alternately solving an ODE and a PDE.
- The ODE may be integrated together with the cell model ODEs, and the PDE (which is now linear) may be solved simultaneously with the other PDEs of the model.
- Free choice of solvers for the two sub-problems (sub-problem solving accuracy must be at least 2 to get overall second order accuracy).

To simplify the operator splitting formulation for the complete model, we introduce the notation

- v_*^n for the value of v after Step 1, i.e. after the ODE solver has been applied.
- v_*^{n+1} for the value after the second step, i.e. after the PDE system is solved but before the second application of the ODE solver
- $s^{n+1/2} = s(t_n + \Delta t/2)$

The complete model

We assume v_n and s_n are known:

1. Compute $s^{n+1/2}$ and v_*^n by solving the system

$$\left.\begin{array}{ll} \frac{\partial v}{\partial t} = -\frac{1}{C_m} I_{\text{ion}}(v,s), & v(t_n) = v^n \\ \frac{\partial s}{\partial t} = F(v,s), & s(t_n) = s^n \end{array}\right., \; t_n < t < t_{n+1/2}.$$

2. Compute v_*^{n+1} (and new values of u_e and u_o) by solving the coupled PDE system

$$\begin{array}{rcll} \chi C_m \frac{\partial v}{\partial t} & = & \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) & x \in H \\ \nabla \cdot ((M_i + M_e)\nabla u_e) & = & -\nabla \cdot (M_i \nabla v) & x \in H \\ \nabla \cdot (M_o \nabla u_o) & = & 0 & x \in T \end{array}$$

   for $t_n < t \leq t_{n+1}$, with v_*^n as initial condition.

3. Compute s^{n+1} and v^{n+1} by solving the system

$$\left.\begin{array}{ll} \frac{\partial v}{\partial t} = -\frac{1}{C_m} I_{\text{ion}}(v,s), & v(t_{n+1/2}) = v_*^{n+1} \\ \frac{\partial s}{\partial t} = F(v,s), & s(t_{n+1/2}) = s^{n+1/2} \end{array}\right., \; t_{n+1/2} < t \leq t_{n+1}.$$

- In the continuous case all variables, including the cell model state vector s, are defined throughout the heart muscle.
- The solution of the equations is based on discretizing the computational domain in a finite number of points (a grid/mesh), and use a numerical method to determine all variables in each point.
- Applying this type of discretization to the ODEs in step 1 and 3 of the solution algorithm gives one ODE system for each point in the grid.

The computational work for each time step consists of two separate tasks:

1. Solving a large number of non-linear ODE-systems (Step 1 and 3).
2. Discretizing and solving a coupled PDE system (Step 2).

Both tasks contribute significantly to the total computational time, so it is important to choose efficient solvers for both sub-problems.

Solving the ODEs

The task in Step 1 and Step 3 is basically to solve a large number of initial value problems on the form

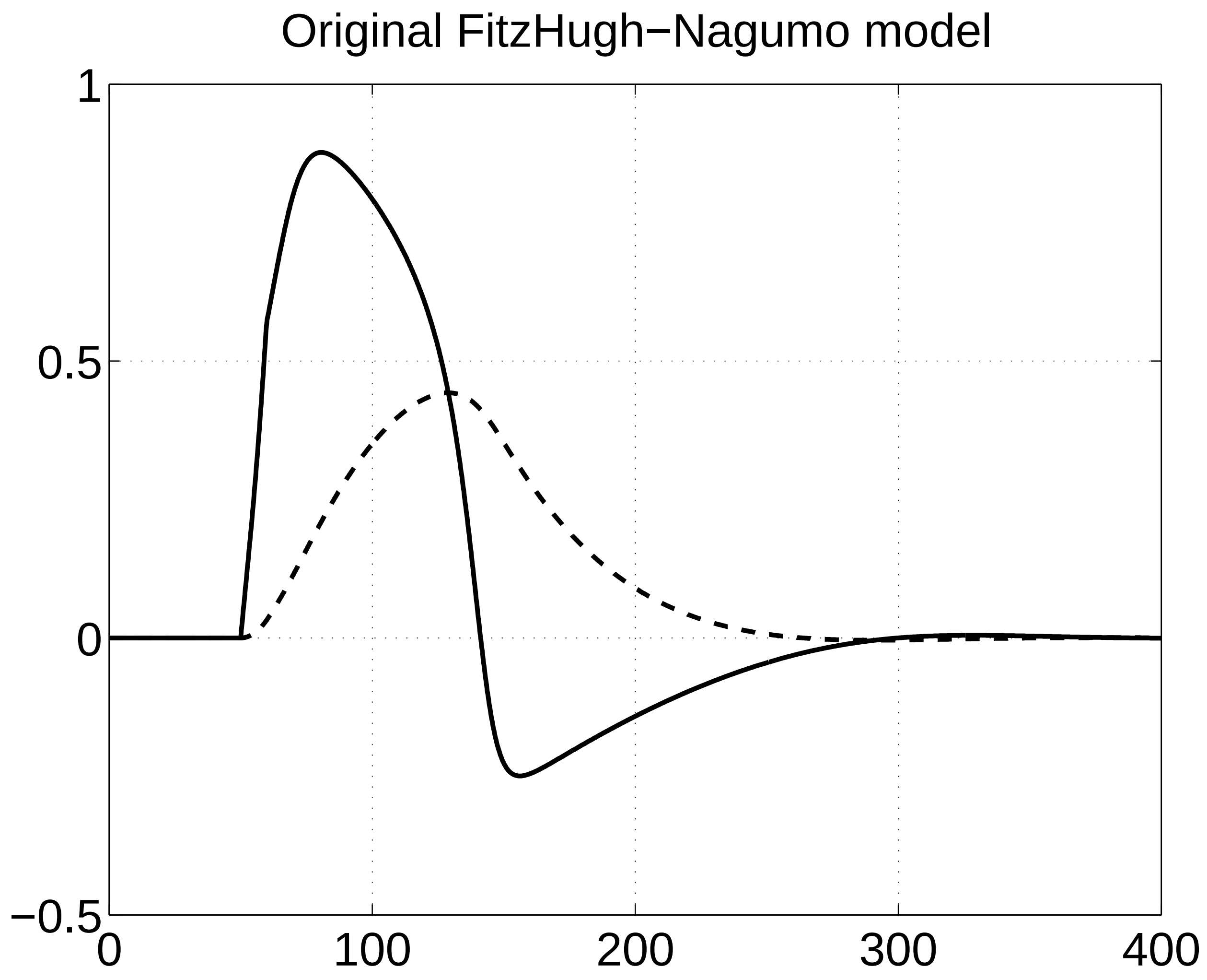
$$\begin{aligned} \frac{d\mathbf{y}}{dt} &= \mathbf{f}(t, \mathbf{y}) \\ y(t_0) &= y_0. \end{aligned}$$

The form and complexity of the ODE system depends on the chosen cell model.

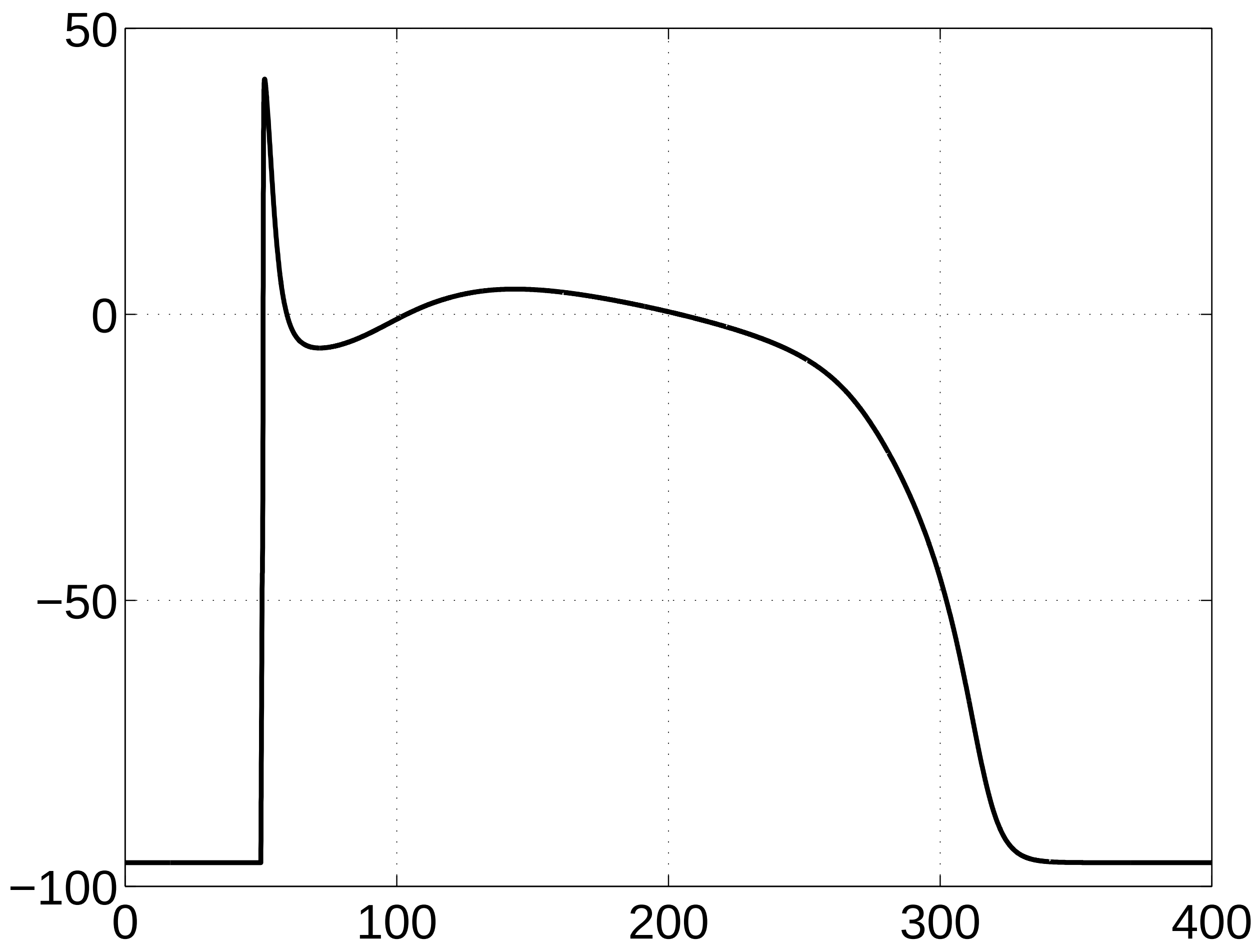
- Simpler cell models: Hodgkin-Huxley (4 ODEs), Beeler-Reuter (8 ODEs), Fitzhugh-Nagumo (2 ODEs).
- More realistic models: Luo-Rudy (13 ODEs), Winslow (31 ODEs) ++

The more realistic cardiac cell models typically have much faster dynamics than the simpler models, adding further to the difficulty of solving these equations.

The Fitzhugh-Nagumo model



The Winslow model



Solvers for ODE systems

- A huge variety of methods have been developed for solving systems of ODEs.
- Simple, well known methods are the forward and backward Euler methods.
- A (large) family of widely used methods are the so-called Runge-Kutta methods.

Runge-Kutta methods

Definition:
Let $b_i, a_{ij}, c_i \ (i, j = 1, \ldots, s)$ be real numbers.
The method

$$\begin{aligned} \mathbf{k}_i &= \mathbf{f}\left(t_0 + c_i \Delta t, \mathbf{y}_n + \Delta t \sum_{j=1}^{s} a_{ij} \mathbf{k}_j\right) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \Delta t \sum_{i=1}^{s} b_i \mathbf{k}_i \end{aligned}$$

is called an s-stage Runge-Kutta method.

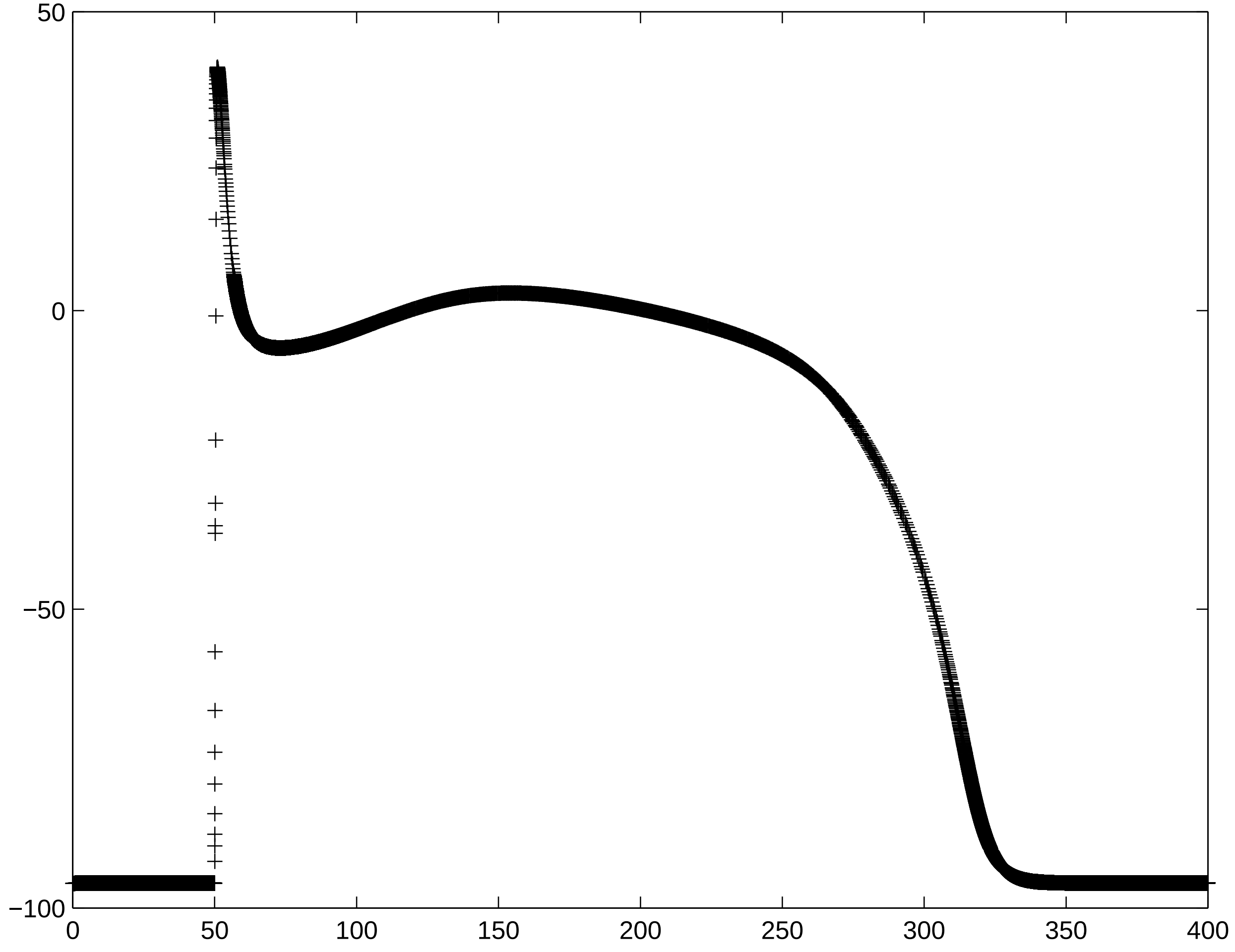
Explicit methods

If $a_{ij} = 0$ for $i \geq j$ we have an explicit (ERK) method, that can be expressed by

$$\begin{aligned}
k_1 &= f(t_0, y_0) \\
k_2 &= f(t_0 + c_2\Delta t, y_0 + \Delta t a_{21} k_1) \\
k_3 &= f(t_0 + c_3\Delta t, y_0 + \Delta t(a_{31} k_1 + a_{32} k_2)) \\
& \dots \\
k_s &= f(t_0 + c_s\Delta t, y0 + \Delta t(a_{s1} k_1 + \cdots + a_{s,s-1} k_{s-1})) \\
y_1 &= y_0 + \Delta t(b_1 k_1 + \cdots + b_s k_s)
\end{aligned}$$

- Explicit RK methods are the classical, most well-known RK methods.
- Efficient methods with good accuracy.
- Stability problems when applied to the stiff cell model equations.

Integrating the ODE-system in the Winslow cell model from $t = 0$ to $t = 400$. A good explicit RK-solver requires about 60000 timesteps.



A semi-implicit method

A class of implicit methods is commonly referred to as semi-implicit methods. An example such a method is the following 4-step scheme:

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(t_n, \mathbf{y}_n) \\
\mathbf{k}_2 &= \mathbf{f}\left(t_n + \Delta t c_2, \mathbf{y}_n + \Delta t \left(a_{21}\mathbf{k}_1 + \gamma \mathbf{k}_2\right)\right) \\
\mathbf{k}_3 &= \mathbf{f}\left(t_n + \Delta t c_3, \mathbf{y}_n + \Delta t \left(\hat{b}_1\mathbf{k}_1 + \hat{b}_2\mathbf{k}_2 + \gamma \mathbf{k}_3\right)\right) \\
\mathbf{k}_4 &= \mathbf{f}\left(t_n + \Delta t c_4, \mathbf{y}_n + \Delta t \left(b_1\mathbf{k}_1 + b_2\mathbf{k}_2 + b_3\mathbf{k}_3 + \gamma \mathbf{k}_4\right)\right) \\
\mathbf{y}_{n+1} &= \mathbf{y}_0 + \sum_{i=1}^{4} b_i \mathbf{k}_i \\
\mathbf{e}_{n+1} &= \sum_{i=1}^{4} (b_i - \hat{b_i})\mathbf{k}_i .
\end{aligned}$$

Main differences between explicit methods and the semi-implicit method.

- A system of non-linear equations must be solved to determine each k_j. For an ODE system of n equations, each k_j is computed by solving a system of n algebraic equations.
- The stability is dramatically improved.

Application to the cell model ODEs

- Numerical experiments have confirmed that implicit methods are much more well-suited for solving the cell-model ODE systems than explicit methods.
- The efficiency difference between different implicit methods (semi- and full) is much smaller, and depends on the accuracy requirements.
- Because of the coupling to the PDEs, the accuracy demands for our application are generally low.

The semi-implicit method

As an example, consider the 4-step semi-implicit method introduced above. Application of this method to the cell model ODEs involves the following tasks:

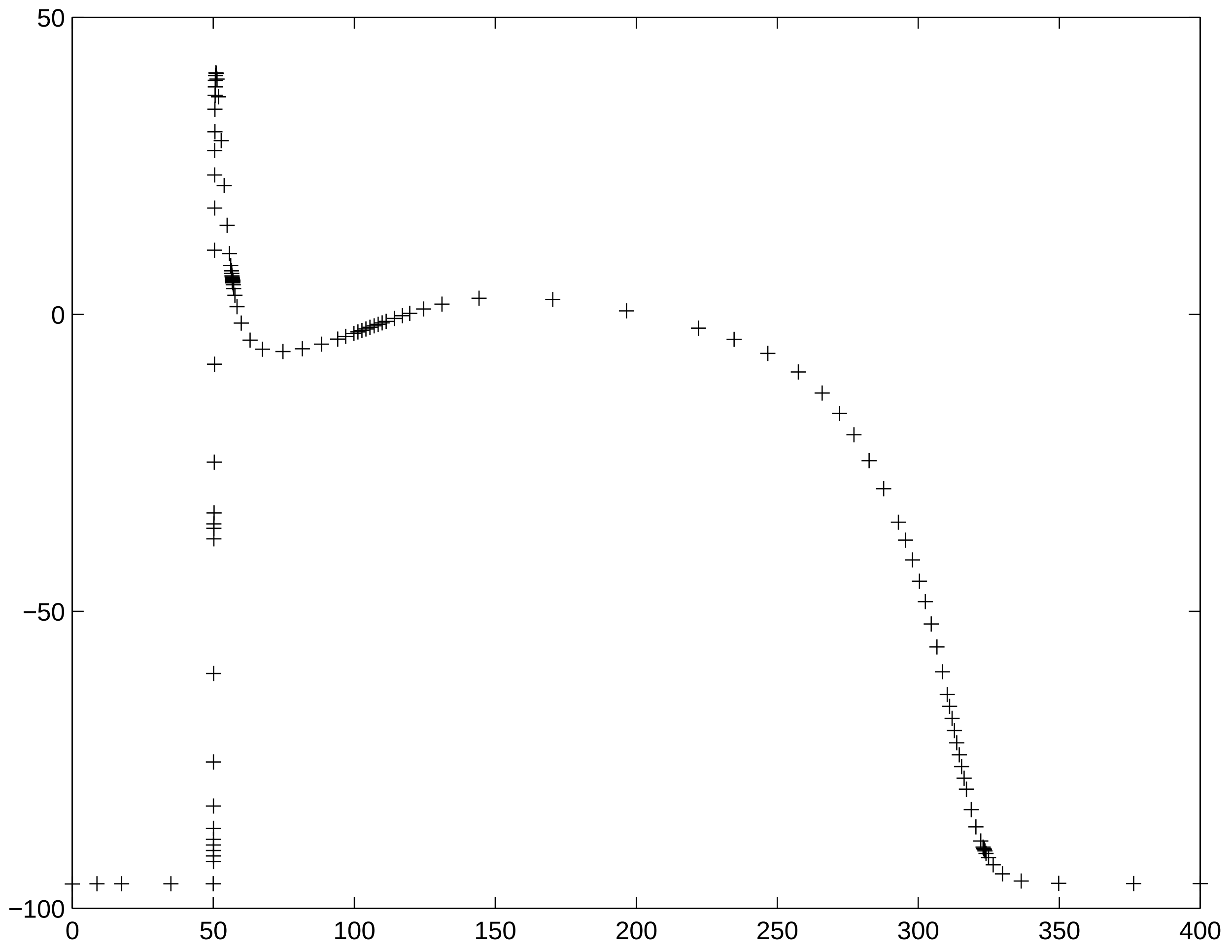
1. Compute the first stage derivative (explicitly) from

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

2. For $j = 2, 3, 4$ use k_{j-1} as start vector and compute k_j by solving a non-linear equation with Newtons method.
3. Compute y_{n+1} from

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

- The amount of work for each time step is considerably larger than for explicit methods, because we need to solve non-linear algebraic equations.
- For stiff problems like the cell model ODEs, the extra work per time step is easily outweighed by the enhanced stability properties, allowing the equations to be integrated with much larger time steps.
- The required number of time steps to integrate the Winslow ODE system with an implicit method is approximately 150. (Compared to about 60000 for the explicit method).


50
0
−50
−100
0
50
100
150
200
250
300
350
400

Discretization of the PDE system

$$\begin{aligned}
\chi C_m \frac{\partial v}{\partial t} &= \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) && x \in H \\
\nabla \cdot ((M_i + M_e) \nabla u_e) &= -\nabla \cdot (M_i \nabla v) && x \in H \\
\nabla \cdot (M_o \nabla u_o) &= 0 && x \in T
\end{aligned}$$

Outline of PDE discretization

- To obtain the overall second order accuracy of the Strang splitting, the PDE system must be solved with second order accuracy. This is achieved with the Crank-Nicolson scheme.
- To simplify the spatial discretization of the problem, the second and third equation of the system are combined to one equation.
- A standard finite element method is used for the spatial discretization of the resulting equations.

Time discretization

A widely used scheme for time discretization of PDEs is the Crank-Nicolson scheme

$$\frac{\partial v}{\partial t} = G(v),$$

an approximate solution is found by

$$\frac{v^{n+1} - v^n}{\Delta t} = 1/2G(v^{n+1}) + 1/2G(v^n)\,.$$

Crank-Nicolson for coupled problems

For a more complex problem on the form

$$\frac{\partial u}{\partial t} = G(v) + H(u),$$

we may apply a combination of Crank-Nicolson and themidpoint rule

$$\frac{u^{n+1} - u^n}{\Delta t} = 1/2G(v^{n+1}) + 1/2G(v^n) + H(u^{n+1/2})\,.$$

This also gives a second order accurate scheme.

Application to the bidomain PDEs

$$\begin{aligned}
&\nabla \cdot M_i \nabla (1/2 v_*^{n+1} + 1/2 v_*^n) \\
&\qquad + \nabla \cdot M_i \nabla u_e^{n+1/2} = \frac{v_*^{n+1} - v_*^n}{\Delta t} \quad x \in H, && (2) \\
&\nabla \cdot M_i \nabla (1/2 v_*^{n+1} + 1/2 v_*^n) \\
&\qquad + \nabla \cdot (M_i + M_e) \nabla u_e^{n+1/2} = 0 \qquad x \in H, && (3) \\
&\qquad \nabla \cdot M_T \nabla u_T^{n+1/2} = 0 \qquad x \in T, && (4)
\end{aligned}$$

Notes on space discretization

- To simplify the space discretization of the problem, the continuity conditions on the heart surface are used to reduce the three equations to two PDEs.
- The result is one equation which is defined only in the heart muscle, and one which is defined throughout the heart and the surrounding body.
- A standard finite element technique is used to discretize the resulting equations.

Notes on implementation

- The main goal of the operator splitting technique was to handle the complexity of the problem.
- Although the solution algorithm looks complex, the splitting of the problem makes it much easier to implement the algorithm.
- The modular nature of the discretized problem fits particularly well with object oriented programming.

The simulator software:

- Numerical intensive part implemented in compiled languages (C++/C/Fortran).
- Python layer on top calling the different components

Design advantages

- Ease of debugging and maintenance.
- Easy to switch solvers (and models) for smaller parts of the problem.

Summary

- Developing an efficient and reliable simulator based on the bidomain model is a challenging task, because of the complexity of the equations.
- The most commonly used techniques are different forms of explicit schemes. These have poor numerical properties for the problem, but are easy to construct and implement.
- Various operator splitting techniques are good alternatives for constructing numerically efficient schemes for the complex problem.

- Strang splitting gives second order accuracy, as long as the sub-problems are solved to at least this accuracy.
- The splitting algorithm reduces the non-linear PDE problem to linear PDEs and non-linear ODEs.
- PDEs:
  - A Crank-Nicolson scheme is used for time discretization.
  - Standard finite element techniques can be used for the spatial discretization.

- ODEs:
  - Stiff, non-linear, fairly complex systems.
  - Because of stiffness, explicit methods perform poorly.
  - Implicit RK methods perform well for the present application.
  - Implicit methods are more complicated to implement than explicit methods, and require more work per time step. This is outweighed by the improved stability (larger time steps).

- The modular nature of the splitting algorithms suits particularly well for object oriented programming, and makes it easier to develop and maintain the simulation software.