LONG-TIME SIMULATIONS ON HIGH RESOLUTION MESHES TO MODEL CALCIUM WAVES IN A HEART CELL*

MATTHIAS K. GOBBERT†

Abstract. A model for the flow of calcium in an atrial heart cell is given by a system of time-dependent reaction-diffusion equations coupled by non-linear reaction terms. Calcium ions enter into the cell at release units distributed throughout the cell. At each position, the probability for calcium to be released increases along with the concentration of calcium, thus creating a feedback loop of waves re-generating themselves repeatedly. To validate the model, simulations on the time scale of several completed waves and on the spatial scale of the entire cell are desirable. This requires long-time studies on spatial meshes that need to have a high resolution to resolve the positions of the calcium release units throughout the entire cell. high resolution meshes needed to resolve the We detail the development of a special-purpose numerical method and parallel implementation for this problem. Parallel performance studies demonstrate the scalability of the implementation on a distributed-memory cluster. Convergence studies verify convergence to analytical expectations and confirm the appropriateness of all method parameters. Application studies on the desired time and length scales confirm that the model exhibits the desired feedback mechanism for calcium currents through the release units at suitable high levels, but the long-time studies demonstrate that more calcium may accumulate in the cell eventually than is physically reasonable.

Key words. reaction-diffusion equation, non-smooth data, finite element method, matrix-free implementation, cluster computing

AMS subject classifications. 35K57, 65F10, 65M60, 65Y05, 92C45

1. Introduction.

Diffusion waves of calcium ions in an atrial heart cell are part of the normal functioning of the heart, but can also trigger arrhythmias (irregular heart beat) [7, 8, 9]. See these sources as well as the appendix in [5] for more references to background material. The model for the calcium flow is given by a system of coupled, time-dependent reaction-diffusion equations

$$\frac{\partial u^{(i)}}{\partial t} - \nabla \cdot \left(D^{(i)} \nabla u^{(i)} \right) + a^{(i)}\, u^{(i)} = r^{(i)} + \left(- J_{\text{pump}} + J_{\text{leak}} + J_{\text{SR}} \right) \delta_{i0} + f^{(i)} \tag{1.1}$$

for the concentrations $u^{(i)}(\mathbf{x}, t)$ of the n_s chemical species $i = 0, 1, \ldots, n_s - 1$ as functions of space $\mathbf{x} \in \Omega \subset \mathbb{R}^3$ and time $0 \le t \le t_{\text{fin}}$. In the application problem, (1.1) is coupled with no flow boundary conditions, and the concentrations at the initial time are given.

The time and space derivatives on the left-hand side of (1.1) model the diffusive transport of each chemical species with diffusivities given by the diagonal, positive definite matrices $D^{(i)} \in \mathbb{R}^{3\times 3}$, $i = 0, 1, \ldots, n_s - 1$. The reaction terms $r^{(i)} \equiv r^{(i)}(u^{(0)}, \ldots, u^{(n_s-1)})$ on the right-hand side are in general non-linear functions of all species and couple all reaction-diffusion equations in (1.1). In the application problem, crucial effects related to the calcium species labeled with index $i = 0$ are contained in the right-hand side terms associated with the the Kronecker delta function δ_{i0} (defined as $\delta_{ij} = 0$ for all $i \ne j$ and $\delta_{ij} = 1$ for $i = j$). In (1.1), the application problem is combined with additional terms $a^{(i)}u^{(i)}$ with constant $a^{(i)} \ge 0$ and given function $f^{(i)} \equiv f^{(i)}(\mathbf{x}, t)$ that incorporates the scalar linear test problem

*The author acknowledges partial support from National Science Foundation under grant DMS–0215373 for the computational hardware used in the numerical studies.

†Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, U.S.A. (gobbert@math.umbc.edu)

$u_t - \nabla \cdot (D\nabla u) + au = f(\mathbf{x}, t)$ in the formulation. This combined formulation of the problems allows to switch the code from one problem to the other by turning off terms and is useful in testing to ensure correctness of the code and associated post-processing routines.

The domain $\Omega \subset \mathbb{R}^3$ denotes the interior of the cell and is chosen as $(-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$ in units of μm. Clearly, a rectangular shape is not the shape of a real cell, but this simple choice reflects the interest of this work that focuses on validating and tuning of the coefficients in the model, rather than on a realistic geometric representation. Specifically, all functions $r^{(i)}$, J_{pump}, J_{SR}, etc. contain parameters, many of which cannot be measured directly but are obtained indirectly from measurements of other observations. So, before using the model with these coefficients as a predictive tool, it needs to be validated against experiments. That is, our task is to demonstrate that the model and its computer implementation are capable of replicating the phenomena observed in the laboratory. In this context, it is sufficient to approximate the shape of the cell. But in turn, what has to be achieved for the present purpose is (i) to be able to compute to final times comparable to the laboratory scales and (ii) to do this with a domain that has a realistic size comparable to a real cell. Specifically, a final time on the order of 1000 ms is our goal, because it will allow enough time for waves to self-organize and for several waves of calcium to traverse the cell, and a domain with one long dimension of 64 μm and a cross-section of 12.8×12.8 μm^2 is a good representation of the scale and the fundamental shape of a cell in this context [7, 8, 9].

The practical problem is now to enable these simulations to complete within a reasonable amount of wall clock time on actually available computational resources. Thus, we set out to choose a numerical method that is mathematically appropriate for the problem and takes advantage of any special structure of the problem for its efficiency, while still representing the salient features of the problem and keeping the goals of the simulations in mind, and to implement this method efficiently. This paper presents an approach to this problem in this spirit that uses the properties of the application problem to design an efficient special-purpose code for reaction-diffusion equations of the form (1.1) on a domain of rectangular shape such as Ω above. Since our code and its parallelization are new, we also demonstrate how to gain confidence in its efficacy by parallel performance studies and its accuracy by convergence studies, before applying it to the application problem.

An interesting contrast to our approach can be seen in [8], in which the general-purpose code MPSalsa (`www.cs.sandia.gov/CRF/MPSalsa`) from Sandia National Laboratories is used to approach the same application problem. This code is designed for much more general problems with additional phenomena (such as heat and mass transfer, etc.) and uses unstructured finite element meshes, thus is potentially much more flexible in representing geometries. It will become apparent in the comparisons below that our choices of numerical methods are remarkable analogous, such as a low-order finite element method in space, fully implicit time stepping, Newton solver, and use of a Krylov subspace method as linear solver, but with certain interesting differences: One key difference is our use of a variable-order time stepping method with automatic step size control that decreases the step size when warranted to control transients in time as well as allows the step size to become large when possible. Another difference is our taking advantage of the fixed-in-time mesh to develop a completely matrix-free implementation of the method, including of the Jacobian in the Newton method, which is thus always up-to-date and exact without additional

computational cost. These key differences permit us to compute to significantly larger final times within reasonable wall clock time on finer meshes and for larger domains than the studies reported in [8].

The following section 2 introduces the application problem in more detail and highlights the numerical challenges that need to be addressed. Section 3 explains the choices for the components of our numerical method in detail and discusses some comparisons with the simulations in [8]. Section 4 presents results in four sections: Parallel performance studies in section 4.1 demonstrate that we can solve problems of the desired size efficiently on a distributed-memory cluster. Sections 4.2 and 4.3 present convergence studies for the numerical with smooth and non-smooth source terms, respectively, to validate the method, its implementation, and our choice of numerical parameters. Finally, section 4.4 shows long-time simulations for two cases, one in which no calcium waves self-organize and one in which two waves self-organize and traverse the domain. Movies of the results are available at the author's website `www.math.umbc.edu/~gobbert/calcium.html`. Our conclusions are summarized in section 5.

2. The application problem.

2.1. The model of the spark mechanism. The key term of the model is the function $J_{\mathrm{SR}}(u^{(0)}, \mathbf{x}, t)$ in the equation for the calcium concentration (labeled as species $i = 0$) in (1.1) that describes the release of calcium at the calcium release units (CRUs), referred to as spark events [7, 9]. On the scale of a cell, the CRUs appear as discrete points distributed uniformly throughout the cell. Specifically, we take the arrangement of the CRUs as a three-dimensional lattice with spacings of $\Delta x_s = \Delta y_s = 0.8$ μm in the x- and y-dimensions and of $\Delta z_s = 2.0$ μm in the z-dimension of the cell with no CRUs on the boundary of the cell [9, p. 105]. These and all other coefficient values and their units are collected in Table 2.1. For our domain of $\Omega = (-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$ with length units of μm, this means that the CRUs form a $15 \times 15 \times 31$ lattice with a total of 6,975 CRUs in the cell. It is immediately clear that one of the challenges for simulations of the calcium flow throughout a complete cell is the need for a numerical mesh that resolves this lattice of points, where the sources of calcium are located.

The release of calcium concentration at each CRU is modeled as a point source on the scale of the cell, mathematically represented as a Dirac delta function $\delta(\mathbf{x} - \hat{\mathbf{x}})$ for a CRU located at $\hat{\mathbf{x}}$ [7, p. 89]. The Dirac delta function is understood here in a three-dimensional sense for short, that is, $\delta(\mathbf{x} - \hat{\mathbf{x}}) = \delta(x - \hat{x})\,\delta(y - \hat{y})\,\delta(z - \hat{z})$, where we also use $\mathbf{x} = (x, y, z)$ and $\hat{\mathbf{x}} = (\hat{x}, \hat{y}, \hat{z})$. The amount of calcium released into the cell is given by the flux density σ, that is, $\int_\Omega \sigma\,\delta(\mathbf{x} - \hat{\mathbf{x}})\,d\mathbf{x} = \sigma$, by the definition of the delta function, gives the amount of calcium released into the cell in 1 ms. The effect of a CRU switching on and off is incorporated by an indicator function in time. More specifically, let the set $\Omega_s = \{\hat{\mathbf{x}} \in \Omega \,|\, \hat{\mathbf{x}} \text{ is a CRU}\}$ denote the set of all CRU locations. Then [8, p. 96]

$$J_{\mathrm{SR}}(u^{(0)}, \mathbf{x}, t) = \sum_{\hat{\mathbf{x}} \in \Omega_s} \sigma\, S_{\hat{\mathbf{x}}}(u^{(0)}, t)\, \delta(\mathbf{x} - \hat{\mathbf{x}}) \tag{2.1}$$

is the superposition of the release of calcium at all CRUs. Since the Dirac delta function is a highly non-smooth term, one critical question for the numerical method is whether its spatial discretization converges.

The indicator function $S_{\hat{\mathbf{x}}}(u^{(0)}, t)$ in each term of the sum in (2.1) houses the stochastic aspect of the sparking mechanism of the CRU at $\hat{\mathbf{x}}$. The model allows the

CRU to open with probability

$$J_{\text{prob}}(u^{(0)}) = \frac{P_{\max}\,(u^{(0)})^{n_{\text{prob}}}}{(K_{\text{prob}})^{n_{\text{prob}}} + (u^{(0)})^{n_{\text{prob}}}} \tag{2.2}$$

as a function of the local calcium concentration $u^{(0)}$ [9, p. 104]. This probabilistic model is checked at the so-called spark times that are every unit in time $\Delta t_s = 1$ ms apart. If a CRU opens at a time $t = \hat{t}$, it stays open for a duration $t_{\text{open}} = 5$ ms, that is, mathematically the indicator function $S_{\hat{\mathbf{x}}}$ is set to 1 for $t \in [\hat{t}, \hat{t} + t_{\text{open}}]$. Notice that expected effect of this setup is that the calcium released at one CRU diffuses to a neighboring CRU, whose probability for opening increases with the increased calcium concentration. If the calcium concentration then reaches a third CRU and it opens, the effect is that of a wave forming throughout the cell [9]. After a CRU closes again, it cannot release calcium again for a time period $t_{\text{closed}} = 100$ ms. Therefore, calcium waves through the cell are separated in time by approximately 100 ms. We see that to simulate a sequence of repeated calcium waves, we need to be able to calculate for long times, such as, up to the final time $t_{\text{fin}} = 1000$ ms.

The experimentally obtained coefficient σ models the amount of calcium released at one CRU [7, 9]. It is a function of the calcium current I_{SR} by $\sigma = I_{\text{SR}}/(2F)$, where F denotes the Faraday constant. The range of I_{SR} from 10 to 20 pA is "back-calculated from the size of sparks" [8, p. 96]. This quantity has crucial influence on whether calcium waves self-organize or not because it determines how much calcium is released into the cell at one CRU, σ, which via diffusion raises the value of J_{prob} in (2.2) at nearby CRUs and thus influences strongly whether they open or not. One interesting validation for this model of calcium waves is to consider the extreme values of this range for I_{SR} and observe whether calcium waves self-organize.

In summary, there are several key challenges for the numerical method and its implementation: The spatial discretization needs to resolve the cell domain with a fine mesh and we need to ensure its convergence in the face of the Dirac delta functions as highly non-smooth source terms. The time discretization needs to ensure small error when CRUs are switched on and off and needs to be efficient enough to allow long-time simulations within reasonable wall clock times.

2.2. The other terms of the application problem. The model for the calcium flow involves in addition to calcium C, labeled as species $i = 0$, additionally an endogenous calcium buffer $\mathrm{F}^{(1)}$, labeled $i = 1$, and a fluorescent indicator dye $\mathrm{F}^{(2)}$, labeled $i = 2$. The reversible binding/unbinding of the indicator and buffer species are modeled by the reaction model for the $n_s = 3$ species [7]

$$\mathrm{F}^{(i)} + \mathrm{C} \rightleftharpoons \mathrm{G}^{(i)}, \quad \text{for } i = 1, \ldots, n_s - 1, \tag{2.3}$$

where $\mathrm{F}^{(i)}$ denotes the free molecules of species i, C denotes the calcium species, and $\mathrm{G}^{(i)}$ the molecules of species i that are bound with C. The reaction model, together with the no flow boundary conditions, assures the conservation of the total concentrations of $\mathrm{F}^{(i)}$ and $\mathrm{G}^{(i)}$ together at a value $\overline{u}_i$ [7] that is determined by the initial conditions. With $u^{(i)}$ denoting the concentration of $\mathrm{F}^{(i)}$, the concentration of $\mathrm{G}^{(i)}$ is then $\overline{u}_i - u^{(i)}$. And with $u^{(0)}$ as concentration of C, the reaction rates are

$$R^{(i)} = -k_i^+ u^{(0)} u^{(i)} + k_i^- \left(\overline{u}_i - u^{(i)}\right) \quad \text{for } i = 1, \ldots, n_s - 1. \tag{2.4}$$

TABLE 2.1
Coefficients of the three-species application problem. The unit M is short for mol / L (moles per liter).

Domains in space and time		
$\Omega = (-6.4, 6.4) \times (-6.4, 6.4) \times (-32.0, 32.0)$		
$0 \le t \le t_{\text{fin}}$ with $t_{\text{fin}} = 1000$		
Reaction-diffusion equation (1.1)		
$D^{(0)} = \text{diag}(0.15, 0.15, 0.30)$ μm^2 / ms		
$D^{(1)} = \text{diag}(0.01, 0.01, 0.02)$ μm^2 / ms		
$D^{(2)} = \text{diag}(0.00, 0.00, 0.00)$ μm^2 / ms		
$a^{(i)} = 0, \quad f^{(i)} \equiv 0$ for all i		
$u_{\text{ini}}^{(0)} = 0.1$ μM,	$u_{\text{ini}}^{(1)} = 111.8182$ μM,	$u_{\text{ini}}^{(2)} = 45.9184$ μM
CRU coefficients, (2.1), and (2.2)		
$\Delta x_s = 0.8$ μm,	$\Delta y_s = 0.8$ μm,	$\Delta z_s = 2.0$ μm
$\sigma = \begin{cases} 51.8213655 \ \mu\text{M} \ \mu\text{m}^3 \text{ / ms} & \text{for } I_{\text{SR}} = 10 \text{ pA} \\ 103.6430533 \ \mu\text{M} \ \mu\text{m}^3 \text{ / ms} & \text{for } I_{\text{SR}} = 20 \text{ pA} \end{cases}$		
$F = 96{,}485.3$ C / mol Faraday constant		
$P_{\max} = 0.3$ / ms,	$K_{\text{prob}} = 15.0$ μM,	$n_{\text{prob}} = 1.6$
$\Delta t_s = 1.$ ms,	$t_{\text{open}} = 5.0$ ms,	$t_{\text{closed}} = 100.0$ ms
Reaction terms (2.4)		
$k_1^+ = 0.08$ / (μM ms),	$k_1^- = 0.09$ / ms,	$\overline{u}_1 = 50.0$ μM
$k_2^+ = 0.10$ / (μM ms),	$k_2^- = 0.10$ / ms,	$\overline{u}_2 = 123.0$ μM
Pump and leak terms (2.6)		
$V_{\text{pump}} = 0.2$ μM / ms,	$K_{\text{pump}} = 0.184$ μM,	$n_{\text{pump}} = 4$
$J_{\text{leak}} = 0.016048418$ μM / ms		

The reaction terms $r^{(i)}$ in the unified notation of (1.1) are then for all species

$$r^{(i)}(u^{(0)}, \ldots, u^{(n_s-1)}) := \begin{cases} \displaystyle\sum_{j=1}^{n_s-1} R^{(j)}(u^{(0)}, u^{(j)}), & \text{for } i = 0, \\ R^{(i)}(u^{(0)}, u^{(i)}), & \text{for } i = 1, \ldots, n_s - 1. \end{cases} \tag{2.5}$$

These reaction terms introduce non-linearity into the problem and constitute the coupling between the equations in (1.1).

Finally, calcium also leaves the cell through the non-linear drain term [7, p. 89]

$$J_{\text{pump}}(u^{(0)}) = \frac{V_{\text{pump}} \, (u^{(0)})^{n_{\text{pump}}}}{(K_{\text{pump}})^{n_{\text{pump}}} + (u^{(0)})^{n_{\text{pump}}}}. \tag{2.6}$$

At rest, that is, for the calcium concentration $u^{(0)} = 0.1$ μM, the constant source term J_{leak} balances the pump term such that $J_{\text{leak}} = J_{\text{pump}}(u^{(0)})$ [7, p. 89]. (In [8, p. 96], there is a typo in the sign of the pump term; cf. the original source [7, p. 89].) The rest concentration $u^{(0)} = 0.1$ μM is chosen as the initial condition for the calcium concentration, and the initial conditions for the other reactive species are computed such that $r^{(i)} = 0$ for all reactions.

3. The numerical method. The problem (1.1), for which a numerical method needs to be developed, is a system of reaction-diffusion equations coupled through the

non-linear reaction terms. We use a method of lines approach to develop a special-purpose code that can handle the Dirac delta functions in (2.1) and that s suitable for long-time simulations on the desired fine meshes.

3.1. Spatial discretization. Recall that $\Omega = (-X, X) \times (-Y, Y) \times (-Z, Z)$ with $X = Y = 6.4$ and $Z = 32.0$ has a regular shape and that the calcium release units (CRUs) at which the Dirac delta functions are centered form a regular lattice Ω_s inside the domain Ω with spacings of Δx_s, Δy_s, Δz_s between the CRUs in the three coordinate directions. We take advantage of this structure by using a uniform numerical mesh $\Omega_h \subset \overline{\Omega}$ with mesh spacings Δx, Δy, Δz chosen such that Ω_h includes the CRU locations as mesh points, that is, we have $\Omega_s \subset \Omega_h \subset \overline{\Omega}$ by design. More specifically, the mesh has $N_x \times N_y \times N_z$ brick-shaped elements and thus there are a total of $N = N_x N_y N_z$ nodes in the mesh. For a problem with $n_s = 3$ chemical species as in the application problem, there will be $n_{eq} = n_s N = 3N$ degrees of freedom (DOF) to be solved for in each time step. Table 3.1 lists several resolutions that we will use in the following and their associated number of nodes N and degrees of freedom n_{eq}. The simulations shown in section 4.4 in fact use the $64 \times 64 \times 256$ mesh and have thus 'only' over 3 million DOFs. But the finer meshes are crucial to study the convergence of the spatial discretization and the reference solution in sections 4.2 and 4.3 in fact used the $256 \times 256 \times 1024$ mesh, albeit in a single-species run, so having as DOFs the nodes N of "only" nearly 68 million listed in the table.

TABLE 3.1
A finite element mesh with $N_x \times N_y \times N_z$ elements has $N = (N_x + 1)(N_y + 1)(N_z + 1)$ nodes and $n_{eq} = n_s N = 3N$ degrees of freedom (DOF) for the application problem with $n_s = 3$ species.

$N_x \times N_y \times N_z$	N	$n_{eq} = \text{DOF}$
$16 \times 16 \times 64$	18,785	56,355
$32 \times 32 \times 128$	140,481	421,443
$64 \times 64 \times 256$	1,085,825	3,257,475
$128 \times 128 \times 512$	8,536,833	25,610,499
$256 \times 256 \times 1024$	67,700,225	203,100,675

The need to discretize the Dirac delta functions in (2.1) motivates the use of the finite element method (FEM), because the weak formulation of the FEM is obtained by integrating (1.1) over Ω, which is well-defined for the Dirac delta functions in J_{SR}. We choose nodal FEM basis functions $\varphi_k(\mathbf{x})$, $0 \le k < N$, that satisfy $\varphi_k(\mathbf{x}_\ell) = \delta_{k\ell}$ for all nodes $\mathbf{x}_\ell \in \Omega_h$. Using φ_k as test function in the weak formulation allows for an explicit evaluation of the the J_{SR} term in the discretization and we find

$$\Sigma_k := \int_\Omega J_{\text{SR}}\, \varphi_k \, d\mathbf{x} = \sigma \sum_{\hat{\mathbf{x}} \in \Omega_s} S_{\hat{\mathbf{x}}} \int_\Omega \delta(\mathbf{x} - \hat{\mathbf{x}})\, \varphi_k(\mathbf{x})\, d\mathbf{x} = \sigma \sum_{\hat{\mathbf{x}} \in \Omega_s} S_{\hat{\mathbf{x}}}\, \varphi_k(\hat{\mathbf{x}}). \tag{3.1}$$

Since $\hat{\mathbf{x}} \in \Omega_s \subset \Omega_h$ by construction of the mesh, $\hat{\mathbf{x}}$ is a node and we obtain $\varphi_k(\hat{\mathbf{x}}) = 1$ if and only if $\hat{\mathbf{x}} = \mathbf{x}_k$. Moreover, $S_{\hat{\mathbf{x}}}(u^{(0)}, t) = 1$ if the CRU at $\hat{\mathbf{x}}$ is open and 0 otherwise, so $\Sigma_k = \sigma$ if $\mathbf{x}_k$ is a CRU and it is open, and 0 otherwise.

The discretization of the other terms in (1.1) is standard, see, e.g., [11, 16] for general information, and see [5] for the concrete derivation of all terms. Let $\mathbf{u}_k^{(i)}(t) \approx u^{(i)}(\mathbf{x}_k, t)$ denote the approximation to the true solution at node $\mathbf{x}_k \in \Omega_h$ at time t, then the finite element solution is denoted by $u_h(\mathbf{x}, t) = \sum_k \mathbf{u}_k^{(i)}(t)\, \varphi_k(\mathbf{x})$. The methods of lines approach yields the semi-discrete problem in vector form for $\mathbf{u}^{(i)} =$

$(\mathbf{u}_k^{(i)})$ for $i = 0, 1, \ldots, n_s - 1$

$$\hat{M}\,\frac{d\mathbf{u}^{(i)}}{dt} = -(K^{(i)} + M_a^{(i)})\,\mathbf{u}^{(i)} + \mathbf{r}^{(i)} + \delta_{i0}\,(\mathbf{j}_{\rm pl} + \Sigma) + \mathbf{f}^{(i)}. \tag{3.2}$$

Here, $K^{(i)}$ denotes the stiffness matrix, $\hat{M}$ is the lumped mass matrix (which is the same for all species) [16], and $M_a^{(i)}$ is mass matrix involving the constant $a^{(i)}$ (and is thus species dependent). The remaining vector terms come from the discretization of their corresponding terms in (1.1), with Σ from above and the pump and leak terms combined in $\mathbf{j}_{\rm pl}$.

The error in the finite element solution $u_h(\cdot, t)$ of the semi-discrete problem (3.2) is measured in the $L^2(\Omega)$-norm. Under standard assumptions [11, 16], it has the form

$$\|u_h(\cdot, t) - u(\cdot, t)\|_{L^2(\Omega)} \le C\,h^q, \quad \text{as } h \to 0, \tag{3.3}$$

where $q > 0$ for convergence and C denotes a generic constant of moderate size independent of the maximum mesh spacing $h := \max\{\Delta x, \Delta y, \Delta z\}$. To guarantee the optimal convergence order of $q = 2$ in (3.3) for linear FEM basis functions, the source terms of (1.1) need to lie in the function space $L^2(\Omega)$, which is not the case due to the Dirac delta functions in $J_{\rm SR}$. In [5], we have in the past argued heuristically and demonstrated computationally that convergence does hold for linear basis functions also in this case, with a convergence order of $q = 0.5$. Notice that we cannot expect better convergence for higher-order elements, which explains our use of the low-order linear finite elements.

The simulations in [8] use the same finite elements, but on an unstructured mesh of tetrahedra and hexahedra with about 200,000 DOFs. A small difference is that their domain is of cylindrical shape with a diameter slightly smaller than our x-y-cross-section. The significant difference is that their domain is really a cross-section of the cell, encompassing only two x-y-planes of CRUs [8, pp. 96–97]. They take advantage of the unstructured mesh by using a higher mesh resolution at and around the points where the CRUs are located. However, extensive studies using COMSOL Multiphysics (formerly FEMLAB; `www.comsol.com`) indicate that the FEM converges also our regular meshes [3], using potentially unnecessary high resolution away from the CRUs.

3.2. Time discretization. The next step in the method of lines approach is the time discretization for (3.2), which becomes a standard initial value problem with mass matrix $M^{(ode)}y'(t) = f^{(ode)}(t, y)$, if the unknown vectors $\mathbf{u}^{(i)}(t)$ for all species are concatenated into one vector $y(t)$ of unknowns. This is one large problem of $n_{eq} = n_s N$ non-linear ordinary differential equations (ODEs). For reaction-diffusion equations with second-order spatial derivatives, the time step restrictions due to the CFL condition are generally considered too severe to allow for explicit time stepping methods. One might also try various splitting methods that decouple the ODEs and solve the ODEs for only one species $\mathbf{u}^{(i)}(t)$ at a time. In [5], we tried a very simple splitting method by lagging the reaction terms in time, which also makes each smaller ODE system linear for this application problem. It turns out that this approach has degraded stability properties and might also not conserve mass as well as desired. In the terminology of [12], which reports extensive tests of time stepping methods for problems of the type considered here, a "fully implicit and balanced" is preferable, in which the ODE system is discretized fully implicitly (and thus does not decouple by species) and the discretization has all reaction terms at the same time. The same

applies to the pump and leak terms, however, the CRU term J_{SR} is necessarily taken explicitly to implement its probabilistic model.

The reaction-diffusion problem (1.1) is very smooth as such, hence when using an implicit time stepping, we expect to be able to take fairly large time steps most of the time. However, the J_{SR} term in (2.1) also includes the $S_{\dot{\mathbf{x}}}$ functions at all CRUs that may switch between 0 and 1 or vice versa at the spark times. At these times, a small time step will be necessary to enable the non-linear and linear solvers to converge and also to control the ODE error reasonably. Therefore, to reach the very large final times desired, we must adopt some variable time stepping strategy with automatic error control. It is understood that the theory behind any error control will probably not be rigorously valid at the spark times due to the discontinuities in the source terms. One approach would be to simply set Δt to a small value manually at these times. We are not using any special strategy at the moment, because the error control appears to overcome the spark times successfully without it. We must now expect that the computer code to be developed will be quite complex. With this the case anyway, we decided to also implement a variable-order scheme with automatic selection of the ODE method order. Concretely, we use the numerical differentiation formulas (NDFk) with method order $1 \leq k \leq 5$ from [14], which are generalizations of the well-known BDFk methods, that are the basis of MATLAB's `ode15s` function (`www.mathworks.com`).

We use relative and absolute tolerances of 10^{-6} and 10^{-8}, respectively, on the relative and absolute error estimators of the ODE method. In the application studies in section 4.4, the steps vary widely in size, with small steps on the order of 10^{-4} immediately after CRUs open or close; the time steps increase steadily afterwards up to 10^{-1}, when the next spark time is usually reached. This high variation in step size allows the solver to reach the desired final time of 1000 ms in under 60,000 time steps. The most interesting observation regarding the ODE solver is that the average method order is about 3, with typical orders ranging from 2 to 4. This shows that we are definitely profiting significantly from the variable order method, as compared to using fixed order methods such as implicit Euler or the trapezoidal rule in time. This is the case in [8], where the trapezoidal rule with a fixed step size of 0.01 takes 10,000 steps to reach the final time of 100 ms.

3.3. The non-linear solver. One price of the fully implicit time discretization is that we have to solve the fully coupled non-linear system of $n_s N$ equations that arises from discretizing (3.2) in time. We choose the Newton method and follow the control structure for managing error control and convergence in MATLAB's `ode15s` function. It is here that we profit from the strategy of using a relatively simple spatial discretization on a uniform mesh, because we are able to compute analytically all matrices in (3.2) [5]. We use this here to supply an analytic Jacobian to the Newton method. Since the matrix-vector products in the linear solver are implemented in matrix-free form, this Jacobian is automatically evaluated at the current step without any additional cost.

A classical problem encountered when solving reaction-diffusion equations numerically is the problem maintaining the non-negativity of the numerically computed concentrations. Clearly, they should be non-negative physically. Also, it can be rigorously established that the unique true solution to the problem (1.1) along with its boundary and initial conditions is non-negative [6]. In combination with a suitable spatial discretization, such as ours, it can be shown that an implicit Euler time discretization admits a non-negative solution [6]. Notice however that it is well-understood that

despite this fact, the Newton method does not necessarily find this non-negative solution, even when started with the non-negative solution at the previous time step as initial guess [6, 13].

Techniques such as clipping, in which negative solution components are set to 0, clearly degrade mass conservation properties, because these corrections add mass whenever they increase a component from a negative value to zero. Recently, MATLAB has introduced a non-negativity preserving feature in its `ode15s` function, discussed in [15]. Its idea is to modify the right-hand side function of the ODE problem (3.2) such that it returns 0 instead of any negative slope; the intention is to have the solution "follow the constraint" [15]. Both these techniques are applied on the level of the ODE solver, though. This means that the Newton steps themselves can still have negative components. This can be a problem, for instance, if the right-hand side function of the ODE involves square roots or other expressions that become invalid even for small negative values. Therefore, we have developed a line-search strategy for the Newton step. If a negative solution component is encountered in a full Newton step, it determines the smaller step size necessary to ensure non-negativity of all components [4]. This strategy is also applied to the computation of the initial guess, which thus can still be based on the predictor formula usually used in the NDFk methods [14], as opposed to having to use the solution at the previous time step. Analogously to MATLAB's strategy, our approach is free of cost if all components are non-negative and only of modest cost otherwise. We note that we have run studies with and without non-negativity preservation and found that it is not crucial for this application problem, as none of the coefficient functions becomes invalid and the solution in fact recovers on its own from negative components. We use a relatively tight tolerance of 10^{-8} in the Newton solver, but it still converges typically within 2 steps on average due to the tight ODE tolerance and the good initial guess thus available. This agrees with the observations in [8], where also 1 to 2 Newton steps are reported; it is not clear what type of Jacobian this method used.

3.4. The linear solver. At each Newton step, we need to solve a linear system of equations, also of size $n_{eq} = n_s N$. Here, we profit again from the fact that we are able to compute analytically all matrices in (3.2), because this enables a matrix-free implementation of products of vectors with any of these matrices, which is all that is needed for each iteration of the Krylov subspace methods. Based on a decision tree in [1, p. 321] (non-symmetric system matrix, with matrix transpose available) and backed up by tests of various Krylov subspace methods in MATLAB, we choose the quasi minimum residual (QMR) method for our non-symmetric problem. We usually use a tolerance of 10^{-3} on the relative residual and a stagnation tolerance of 10^{-14} for the linear solver.

We do not use any pre-conditioning at this point, because a matrix-free preconditioner that avoids parallel communications was not readily apparent. This is probably not a significant problem, since the tests show that the linear solver tends to converge within fewer than 4 iterations on average due to the tight ODE and Newton tolerances enforced. This compares reasonably to the results reported in [8], where a GMRES method, with an approximate incomplete LU preconditioner on each parallel subdomain, took fewer than 10 iterations.

3.5. Parallel implementation. Parallel computing is a crucial ingredient to our code for two reasons: (i) It enables the simulations on the fine meshes listed in Table 3.1 and (ii) it speeds up the computations sufficiently to enable simulations up to large final times within a manageable amount of wall clock time. The code is written

in C with MPI commands for the parallel communications for maximum portability. We split the domain Ω into non-overlapping subdomains, with one on each of the P parallel processes, by cutting in the long dimension of Ω. This choice makes the x-y-planes of nodes whose values need to be exchanged between neighboring processes as small as possible, i.e., is the optimal decomposition in a graph partitioning sense. Our code is capable of using any number of parallel processors. The communications between neighbors occur in each matrix-vector product and are implemented by non-blocking `MPI_Isend/MPI_Irecv` commands. These have proven to be faster than blocking communication commands on our system, but not faster than other MPI point-to-point communication commands available. `MPI_Allreduce` commands are needed for all norm computations as well as for various diagnostic quantities such as minimum and maximum values of the solutions. The wall clock times are measured using `MPI_Barrier` and `MPI_Wtime` in sequence.

One interesting issue is the implementation of the random number generator. We use the `genrand()` function that generates a sequence of uniformly distributed pseudo-random numbers [10]. We use this function on Process 0 only to avoid any dependence of the simulation result on the number of parallel processes used. Then, `MPI_Bcast`, `MPI_Scatterv`, and `MPI_Gatherv` commands are involved in setting up the vector Σ in (3.2).

The simulations reported below were performed on the Beowulf cluster kali in the Department of Mathematics and Statistics at the University of Maryland, Baltimore County. This distributed-memory cluster has 32 compute nodes with two 2.0 GHz Intel Xeon CPUs and 1 GB of memory. One of the compute nodes also serves as storage node and is connected to a 0.5 TB RAID array. The compute nodes are connected by a Myrinet interconnect as well as by a fast ethernet for file serving. An additional management and user node connects the cluster to the outside network. The cluster runs the Linux RedHat EL3 operating system, and the Intel compiler suite is used for this code. See `www.math.umbc.edu/~gobbert/kali` for more information. For the application studies, we typically use approximately 16 dual-processor nodes. For the simulations up to $t_{\text{fin}} = 1000$ ms in section 4.4, the code ran about 4 hours for the $32 \times 32 \times 128$ mesh and about 40 hours for the $64 \times 64 \times 256$ mesh. These times are comparable to the times in the range of 12 to 36 hours reported in [8] for simulations up to the 100 ms, using from 4 to 16 dual-processor nodes, whose processor speeds range from 1 to 3 GHz.

4. Results. The numerical results are presented in four sections. The results in section 4.1 analyze first what size of problem the parallel code will be able to solve and that it does so efficiently. This is used in the following sections 4.2 and 4.3 because to compute the reference solution on the finest mesh requires the parallel code. These sections summarize results from convergence studies to confirm that the numerical method is reliable, for a linear test problem with a smooth source term in section 4.2 and with a single non-smooth Dirac delta function in section 4.3. Finally, section 4.4 presents two representative long time simulations of the full application problem up to the large final time $t_{\text{fin}} = 1000$ that show that our method allows for the thorough investigation of the given model.

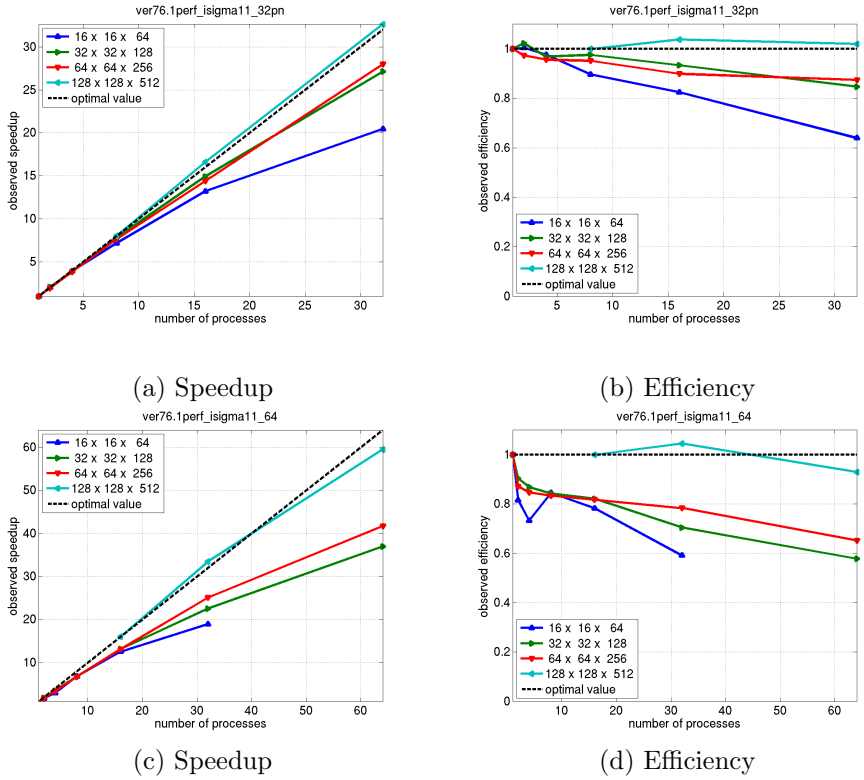
4.1. Parallel Performance Results. The first purpose of parallel computing is to enable the solution of larger problems. To demonstrate this ability, Table 4.1 (a) estimates the amount of memory in MB required to solve the application problem with $n_s = 3$ species on the meshes listed in Table 3.1 that lists the degrees of freedom n_{eq} for each mesh. Table 4.1 (a) is based on a count of the variables with major

TABLE 4.1
Memory usage in MB per process for the application problem with $n_s = 3$ species. The storage requirements are 8 auxiliary vectors for the QMR method and 13 additional vectors for the ODE solver, each of length n_{eq}.

(a) Predicted memory usage in MB per process							
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$16 \times 16 \times 64$	9	5	2	1	1	< 1	< 1
$32 \times 32 \times 128$	68	34	17	8	4	2	1
$64 \times 64 \times 256$	522	261	130	65	33	16	8
$128 \times 128 \times 512$	4103	2052	1026	513	256	128	64
$256 \times 256 \times 1024$	32540	16270	8135	4068	2034	1017	508
(b) Observed memory usage using dedicated nodes							
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$16 \times 16 \times 64$	11	27	25	23	23	22	N/A
$32 \times 32 \times 128$	71	57	40	31	27	25	N/A
$64 \times 64 \times 256$	523	283	153	91	57	41	N/A
$128 \times 128 \times 512$	N/A	N/A	N/A	541	285	157	N/A
(c) Observed memory usage using non-dedicated nodes							
	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
$16 \times 16 \times 64$	11	8	27	25	25	24	N/A
$32 \times 32 \times 128$	71	39	42	33	29	27	26
$64 \times 64 \times 256$	523	267	155	93	60	43	35
$128 \times 128 \times 512$	N/A	N/A	N/A	N/A	287	159	98

memory usage in the code. This count reveals that the QMR method uses 8 auxiliary vectors and that (our implementation of) the ODE solver requires $8 + K$ additional vectors, re-using as many of the QMR auxiliary vectors as possible. Here, $1 \le K \le 5$ denotes a chosen maximum method order for the NDFk, $1 \le k \le K$, method, which is commonly $K = 5$; we use this value, as well, but it has turned out that the application problem only uses orders up to 4, so we could save a little memory here by selecting K smaller. Each vector has length $n_{eq} = n_s N$ specified in Table 3.1. With 8 B per double-precision number, we obtain the total amount of memory needed for each mesh resolution in the $P = 1$ processor column of Table 4.1 (a). We note that resolutions finer than $64 \times 64 \times 256$ cannot be accommodated on a single node. This memory gets divided into the P processors as shown in the following columns.

Tables 4.1 (b) and (c) list the memory in MB per process actually used by the code for each case, as observed by monitoring the output of the Linux utility `top` during the program execution. Table 4.1 (b) considers the case of 'dedicated' nodes, that is, only one CPU per node is used with the second one idling. This dedicates the full memory of each node available for the simulation and avoids any contention for resources of the node among the 2 CPUs, but is also wasteful in the sense that half of the CPUs reserved by the scheduler for the job are idling. Table 4.1 (c) shows the memory usage observed in the case of 'non-dedicated' nodes, that is, with both CPUs on each node being used. Thus, the two CPUs do not have dedicated memory, but rather share the memory of the node and might also suffer from resource contention, e.g., for the use of the single Myrinet port on the node, of the memory or cache, and of the bus that connects both CPUs to all components of the node. The results in Tables 4.1 (b) and (c) are in agreement with the predictions in Table 4.1 (a), with a modest amount of baseline usage associated probably with the use of MPI functions;



(a) Speedup (b) Efficiency

(c) Speedup (d) Efficiency

FIG. 4.1. *(a) Observed speedup and (b) observed efficiency up to 32 processors using dedicated nodes (only one CPU per node used). (c) Observed speedup and (d) observed efficiency up to 64 processors using non-dedicated nodes (both CPUs per node used). Notice the different scales on the axes.*

notice the much smaller overhead for the $P = 1$ cases. These considerations indicate problems of which size we will be able to attack and it gives confidence in the proper understanding of the code's memory usage.

The second purpose of parallel computing is to solve a problem of a given size faster. Ideally, a run using P processors should be P times as fast as the 1-processor run. To quantify this, we first need to time the code. In the context of a true parallel code that inherently includes the need for the processors to communicate with each other, the correct measure of time is wall clock time T_P when using P processors, because it includes both the calculation time associated with arithmetic and similar operations that are local to a CPU and the communication time associated with the sending and receiving of messages between the parallel processes. The speedup defined as $S_P := T_1/T_P$ quantifies how much faster the P-processor run is over the 1-processor one; for the $256 \times 256 \times 1024$ mesh, the definition of speedup is modified to $S_P := 8T_8/T_P$, since the 8-processor case is the first one to fit in memory. The optimal value of S_P is P. Thus, by plotting S_P vs. P, one can get a visual impression how fast the actual performance deteriorates from the ideal one. Figure 4.1 (a) shows the speedup observed for simulations with up to 32 dedicated nodes, as described above. We see that the scalability of the code is excellent and gets better as the size of the problem increases. For the finest mesh, the results plotted are better than optimal by

a small margin. We believe that this reflects the slight variability of timing results. Another way to quantify how close the speedup S_P is to its optimal value P is to plot efficiency $E_P = S_P/P$ vs. P, as shown in Figure 4.1 (b). Even though the data is essentially the same as in the previous plot, efficiency plots are very useful to bring out whether there is any abrupt deterioration of the parallel performance for small values of P, where a speedup plot is too cluttered to see. In this case, there is no noticeable deterioration in that area; rather, the efficiency decreases slowly, but is at very respectable levels of over 80% throughout for all meshes but the coarsest one.

Figures 4.1 (c) and (d) contain the speedup and efficiency plots for the studies with non-dedicated nodes associated with Table 4.1 (c). The speedup is nearly as good up to 32 processors as for the dedicated nodes; notice the scale on the axes here. However, the efficiency plot clearly brings out an abrupt deterioration of performance, as we go from 1 processor to 2 processors; this is the value of the efficiency plot, because this effect is not readily visible in the speedup plot. This effect is caused by the code running on both CPUs on one node competing for the node's resources, in particular the single bus connecting both CPUs to memory. We have been able to reproduce this effect by running two completely independent serial jobs running on one node, so we conclude that the problem is *not* associated with the Myrinet interconnect. This effect is quite typical on commodity clusters because the two CPUs on one node indeed share all other components on the node with each other. One might think now that therefore it would be best to use always only one CPU per node, and this is true when comparing runs using the same total number of *CPUs*. But in practice, a user can reserve *nodes* from the scheduler, and then a run using both CPUs per node will use twice as many CPUs and be faster than one that only uses one CPU per node, though not necessarily twice as fast. Hence, unless the memory of a node cannot accommodate a run with using both CPUs, one should use both CPUs when available. The parallel runs in the following sections use this approach.

4.2. Scalar test problem with smooth source term. We first consider briefly the scalar linear test problem $u_t - \nabla \cdot (\nabla u) = 0$, already used to test an earlier version of this code [2], obtained from (1.1) by setting $n_s = 1$ to get a scalar problem and then $D = 1$, $a = 0$, $f \equiv 0$, and all application-related functions to 0. We consider this problem on the same domain as the application problem $\Omega = (-X, X) \times (-Y, Y) \times (-Z, Z)$ with $X = Y = 6.4$ and $Z = 32.0$. This test problem also continues to use the no flow boundary conditions of the application problem. The initial condition is specified in agreement with the chosen true solution

$$u(x, y, z, t) = \frac{1 + \cos(\lambda_x x)\, e^{-D_x \lambda_x^2 t}}{2} \, \frac{1 + \cos(\lambda_y y)\, e^{-D_y \lambda_y^2 t}}{2} \, \frac{1 + \cos(\lambda_z z)\, e^{-D_z \lambda_z^2 t}}{2}$$

using the notation $\lambda_x = x/X$, $\lambda_y = y/Y$, $\lambda_z = z/Z$ and the notation $\mathbf{x} = (x, y, z) \in \overline{\Omega}$.

The finite element solution for this problem using linear basis functions satisfies (3.3) with $q = 2$ at every point in time t. Therefore, we expect the L^2-error of the numerical solution against the true solution to decrease by a factor 4, whenever the mesh spacings Δx, Δy, Δz are halved. This is born out by the results in Table 4.2 (a) at three representative times $t = 2, 3, 4$. Notice that this confirms that the ODE tolerances are chosen small enough for the spatial error to dominate. To formally estimate the convergence order q in (3.3) from numerically observed errors, one can

TABLE 4.2
Convergence study for scalar test problem with smooth source term.

(a) Error on Ω against true solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	4.0121e-02	5.6277e-02	7.0185e-02
$32 \times 32 \times 128$	1.0100e-02 (1.990)	1.4148e-02 (1.992)	1.7650e-02 (1.992)
$64 \times 64 \times 256$	2.5074e-03 (2.010)	3.5055e-03 (2.013)	4.3869e-03 (2.008)
$128 \times 128 \times 512$	6.0012e-04 (2.063)	8.3361e-04 (2.072)	1.0591e-03 (2.050)
(b) Error on Ω against reference solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	3.9999e-02	5.6112e-02	6.9959e-02
$32 \times 32 \times 128$	9.9770e-03 (2.0033)	1.3984e-02 (2.0046)	1.7424e-02 (2.0054)
$64 \times 64 \times 256$	2.3849e-03 (2.0647)	3.3408e-03 (2.0655)	4.1608e-03 (2.0661)
$128 \times 128 \times 512$	4.7750e-04 (2.3204)	6.6881e-04 (2.3205)	8.3285e-04 (2.3207)

use the estimation formula

$$q^{(est)} = \log_2 \left(\frac{\|u_{2h}(\cdot,t) - u(\cdot,t)\|_{L^2(\Omega)}}{\|u_h(\cdot,t) - u(\cdot,t)\|_{L^2(\Omega)}} \right). \tag{4.1}$$

The results of this formula are listed in the parentheses in Table 4.2 (a), and we note them to be consistent with $q = 2$.

The above procedure uses the true solution $u(\mathbf{x}, t)$, which is not available in practice. An alternative in that case is to use the numerical solution on the finest possible mesh as reference solution in place of $u(\mathbf{x}, t)$. We use here the numerical solution on the mesh $256 \times 256 \times 1024$. The results of the observed errors and the convergence order estimates in Table 4.2 (b) show agreement with the results based on the true solution above. The purpose of these tests was validate the choice of ODE tolerances, to carefully test the code, and to confirm that the post-processing procedure to estimate the convergence order $q^{(est)}$ only using available numerical data is reliable.

4.3. Scalar test problem with non-smooth source term. As in the previous section, we solve the scalar test problem obtained from (1.1) by setting the application functions to 0 and choosing $n_s = 1$, $D = 1$, $a = 0$, but now with $f(\mathbf{x}, t)$ as one Dirac delta function centered at $\mathbf{x} = 0$ that opens at $t = 1$ and stays open for the duration of the simulation. We again use the domain of the application problem in this test. In the terminology of the application, we have 1 centered CRU in the cell that starts firing at $t = 1$. For this case, the classical theory does not apply, but considerations and computational results in [5] lead us to expect $q = 0.5$ in (3.3). Since no true solution is available for this problem on a finite domain, the errors in Table 4.3 (a) against a reference solution on the finest mesh $256 \times 256 \times 1024$ are computed by the post-processing procedure tested in the previous section. They do converge, but it is hard to tell by what ratio the errors decrease. But the $q^{(est)}$ in the parentheses show that the errors decrease with a convergence order that is consistent with the expected number $q = 0.5$.

Another way to check convergence is possible by combining the results of Tables 4.3 (b) and (c). Table 4.3 (b) considers the L^2-norm on the domain Ω with a small area centered about the Dirac delta function's center removed. Specifically, let

TABLE 4.3
Convergence study for scalar test problem with non-smooth source term.

(a) Error on Ω against reference solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	1.8651e+03	1.8503e+03	1.8415e+03
$32 \times 32 \times 128$	1.7120e+03 (0.124)	1.6974e+03 (0.124)	1.6951e+03 (0.120)
$64 \times 64 \times 256$	1.4537e+03 (0.236)	1.4531e+03 (0.224)	1.4529e+03 (0.222)
$128 \times 128 \times 512$	9.6843e+02 (0.586)	9.6832e+02 (0.586)	9.6829e+02 (0.585)
(b) Error on $\Omega \backslash \Omega_0$ against reference solution (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	2.6478e-01	9.8039e-01	2.2494e+00
$32 \times 32 \times 128$	1.2526e-01 (1.080)	3.7741e-01 (1.377)	6.6924e-01 (1.749)
$64 \times 64 \times 256$	3.7324e-02 (1.747)	1.1385e-01 (1.729)	1.8863e-01 (1.827)
$128 \times 128 \times 512$	8.0971e-03 (2.205)	2.3743e-02 (2.262)	3.8368e-02 (2.298)
(c) Error in species mass (estimated convergence order)			
	$t = 2$	$t = 3$	$t = 4$
$16 \times 16 \times 64$	1.4332e+00	3.6992e+00	4.4979e+00
$32 \times 32 \times 128$	1.2033e+00 (0.252)	1.2032e+00 (1.620)	1.2032e+00 (1.902)
$64 \times 64 \times 256$	3.1122e-01 (1.951)	3.1111e-01 (1.951)	3.1110e-01 (1.951)
$128 \times 128 \times 512$	7.8784e-02 (1.982)	7.8737e-02 (1.982)	7.8707e-02 (1.983)

$\Omega_0 := (-\varepsilon_x, \varepsilon_x) \times (-\varepsilon_y, \varepsilon_y) \times (-\varepsilon_z, \varepsilon_z)$ with ε_x, ε_y, ε_z chosen as the mesh spacings Δx, Δy, Δz of the coarsest mesh $16 \times 16 \times 64$, then we consider the $L^2(\Omega \backslash \Omega_0)$-norm. The errors in this norm listed in Table 4.3 (b) clearly converge quadratically again. Since we have removed Ω_0 from consideration, the question remains whether the solution at the center of the domain, where the single CRU is located can be trusted. We answer this question by considering the total mass in the system $m_h(t) := \int_\Omega u_h(\mathbf{x}, t)\, d\mathbf{x}$ at time t; we indicate by the subscript h that $m_h(t)$ is the mass associated with the numerical solution u_h on mesh Ω_h. For this scalar problem with no flow boundary conditions and the delta function as the only source, we know that this mass should equal the mass at the initial time $m(0) = \int_\Omega u_{\text{ini}}(\mathbf{x})\, d\mathbf{x}$ plus the mass released into the cell up to time t given by $\int_0^t \sigma\, n_{\text{open}}(t')\, dt'$, where $n_{\text{open}}(t')$ denotes the number of open CRUs at time t'. Since the sole CRU in this system opens at $t = 1$, the latter integral is equal to $(t-1)\sigma$ for $t \geq 1$. Table 4.3 (c) lists the errors $|m_h(t) - m(0) - (t-1)\sigma|$ for $t = 2, 3, 4$, and we observe quadratic convergence for this quantity; note that the apparently large values for the mass error must be viewed in the context of the large size of the domain which is $(12.8)(12.8)(64.0) = 10485.76$. Thus, we conclude from Table 4.3 (c) that mass is conserved in the system, which shows together with the convergence on $\Omega \backslash \Omega_0$ in Table 4.3 (b) that also the error at $\mathbf{x} = 0$ is converging.

4.4. Case studies for the full application problem. In this section, we present two case studies for the full application problem described in section 2. Recall from section 2.1 that the calcium current through a CRU, I_{SR}, determines the amount of calcium released into the cell through an open CRU and thus influences whether a calcium wave self-organizes or not [8]. We consider here the values $I_{\text{SR}} = 10$ pA and $I_{\text{SR}} = 20$ pA resulting in the corresponding values of σ in Table 2.1. The initial concentration of calcium is chosen at rest, $u^{(0)} = 0.1$ μM, throughout the cell and the other species concentrations computed such that $r^{(i)} = 0$ for all i. Also, we have $J_{\text{leak}} = J_{\text{pump}}$ for $u^{(0)} = 0.1$ μM. Initially, all CRUs are closed, hence, $J_{\text{SR}} = 0$. The

numerical parameters used for the studies are specified and discussed in section 3.

Since no CRUs are triggered artificially, the first test for the model is whether any CRUs will open randomly and whether the diffusion of the calcium released causes any neighboring CRUs to open. Figures 4.2 and 4.3 show results for the case study with $I_{\mathrm{SR}} = 10$ pA. Figure 4.2 shows the plots indicating any CRU open in the domain at ten selected times from 100 ms to 1000 ms; each dot indicates that the CRU at the spatial point is open and does not represent the value of any quantity. We observe that a number of CRUs open randomly over time, thus the probability mechanism of the model works. However, no systematic opening of CRUs develops. Figure 4.3 shows isosurface plots of the calcium concentration $u^{(0)}$ throughout the cell with an isolevel of 65 μM indicated by each surface. It is clear that the concentration around open CRUs does increase. It then diffuses and thus the concentration in the area falls below the isolevel chosen again. We notice that the level of calcium concentration does not rise high enough anywhere to cause more CRUs in the area to open.

Figures 4.4 and 4.5 show results for the case study with $I_{\mathrm{SR}} = 20$ pA. We see in Figure 4.4 that at time $t = 100$ ms a number of CRUs are open without any discernible pattern. But by $t = 200$ ms, two waves of CRUs opening have self-organized in this case. At some time between 100 and 200 ms, the concentration near the left end of the domain as well as just to the right of center at the top has reached higher levels that caused several neighboring CRUs to open at the same time. In turn, more neighboring CRUs of those opened and by $t = 200$ ms we have one wave traveling left to right with its front at about $z = -12$ and a second wave expanding from a center at about $(x, y, z) = (5, 5, 5)$. The formation of the waves is clearly visible in the movies available at the website mentioned in the Introduction. After the CRUs, where the two waves started, have been closed for a time period of $t_{\mathrm{closed}} = 100$ ms, they open again and re-start a wave similar to each first one. This process is repeated several times throughout the simulation. The reason why several of the snapshots look remarkably similar is that the time between them of 100 ms approximates the period of time between wave initiations of $t_{\mathrm{open}} + t_{\mathrm{closed}} = 105$ ms. These results demonstrate that the model at this higher value of current $I_{\mathrm{SR}} = 20$ pA promotes the self-organization of waves, as intended by the model. Notice that it took some time of between 100 and 200 ms for the first wave to form, which demonstrates the importance of being able to perform long-time simulations for this application problem. Even more so, the effect of waves traveling through the cell several times could only be seen by simulating to a large final time such as 1000 ms.

Figure 4.5 shows isosurface plots of the calcium concentration $u^{(0)}$ throughout the cell with an isolevel of 65 μM. We see at $t = 100$ ms that the concentration has crossed this isolevel only around a few CRUs that happen to be open. By $t = 200$ ms however, in the wake of both waves, we see significantly increased levels of calcium, which by $t = 400$ ms have reached levels above the isolevel throughout the domain. Clearly, the model exhibits the feedback mechanism of open CRUs releasing calcium and the diffused calcium in turn promoting the initiation of a new wave after the time period of closure t_{closed} has passed. In fact, we note from our log files that about 300 CRUs are open at any given moment in time for all times $t \geq 200$ ms. Plots of the maximum each of the species concentration vs. time (not shown) demonstrate that the calcium concentration grows without bound and the other free species are practically completely consumed. It would appear that this behavior is not physical, because the calcium concentration is not be permitted to grow without bound in a cell. This points to some effect being missing or coefficient values being not appropriate. Notice here

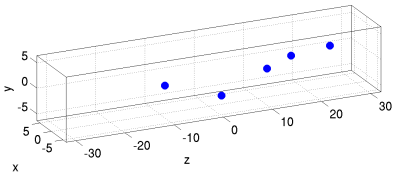
that this fact is simply not apparent until simulations to times significantly larger than $t_{\text{closed}} = 100$ ms are performed, because we have to wait for waves to travel through the cell several times to see this effect.

5. Conclusions. We consider a model for calcium waves in an atrial heart cell proposed in [7, 8, 9]. To validate this model and recreate the conditions of an experiment, it is desirable to be able to perform simulations up to large final times, to allow for waves to re-generate several times, and on a domain that encompasses the entire cell. This requires a high resolution mesh to adequately discretize the given lattice of calcium release units throughout the cell. Section 3 presents all choices for the development of a special-purpose simulation code for this model from the ground up. The key to our ability to perform the desired long-time simulations on a high-resolution mesh are the use of a variable-order, variable step size ODE solver and the matrix-free implementation of all linear solves.

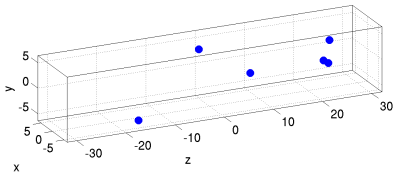
The code is applied to the application problem in section 4.4. The results show that the model successfully allows for the self-organization of a wave at a random location in the cell, without any artificial triggering of calcium release. To see this result, it was already necessary to simulate up to final times over 100 ms. But to see waves re-generate and travel through the cell several times, we need to be able to reach final times of at least 1000 ms. This is possible for our special-purpose code, and the results indicate that the model, in the form stated in [7, 8, 9] and with the model parameters available from these references, may eventually accumulate more calcium in the cell than is physically reasonable. This observation can only be made because our code can compute to sufficiently large final times.

We note that our application studies follow careful convergence studies in sections 4.2 and 4.3 for a scalar linear test problem with smooth and non-smooth source term, respectively, designed to evaluate the accuracy and reliability of our method and its implementation as thoroughly as possible. The results also demonstrate that the method converges on our uniform mesh without refinement around the locations of the calcium release units. Notice that a convergence study for the full application problem including the probabilistic term is problematic, because its behavior is influenced by the random number generator and also by the calcium concentration, which is not exactly the same at different resolutions. Finally, section 4.1 demonstrated the effectiveness of using parallel computing to solve this large problem and the scalability of our implementation.

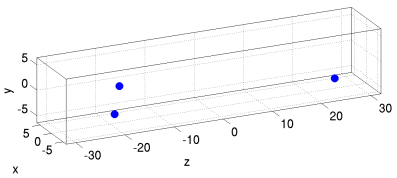
In section 3, we also include a number of comparisons to the simulator used in [8]. One apparent difference is the use of our uniform mesh to their unstructured mesh. They use this to refine the mesh around the locations of the calcium release units. But according to our convergence studies, this is not necessary. So, the difference between [8] and this work is actually the use of a general-purpose code vs. the development of a special-purpose one. This distinction is independent of the mesh used, because, as long as the mesh is regular in some way and fixed in time, we can still compute all matrices analytically and implement all linear solves in matrix-free form, which is one key to the efficiency of our code. But we believe this is not the direction of most promise to pursue, because this still does not take full advantage of the knowledge about the application: A fine mesh around the location of a calcium release unit is needed only if that release unit is open; not when it is closed, as is the case far most of the time (compare $t_{\text{open}} = 5$ ms with $t_{\text{closed}}100$ ms). So, the most significant advance in efficiency could be achieved by an automatic mesh refinement and coarsening strategy that uses a finer mesh only around those release units that are open.



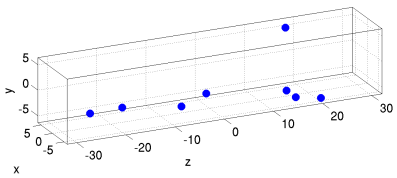
$t = 100$



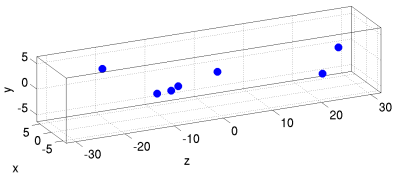
$t = 200$



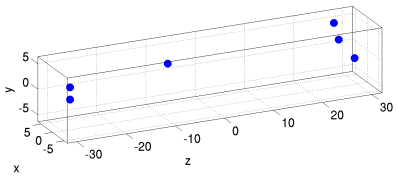
$t = 300$



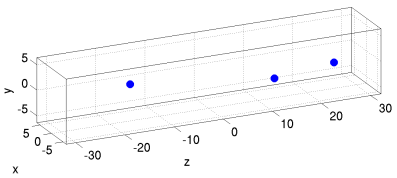
$t = 400$



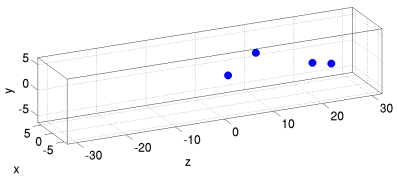
$t = 500$



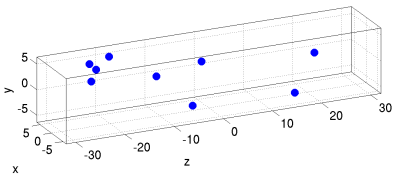
$t = 600$



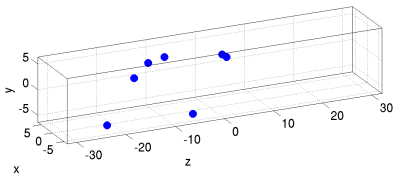
$t = 700$



$t = 800$

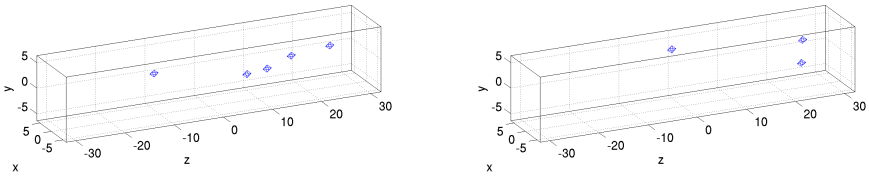


$t = 900$

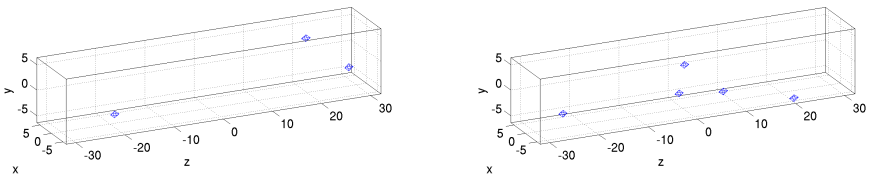


$t = 1000$

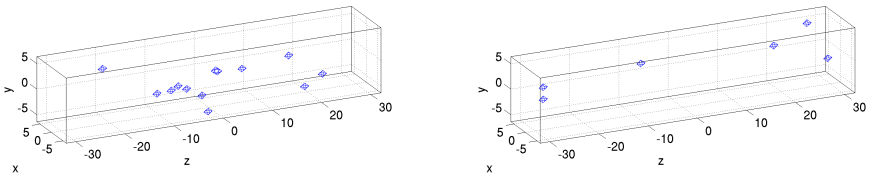
FIG. 4.2. *Open calcium release units throughout cell with* $I_{SR} = 10$ *pA.*



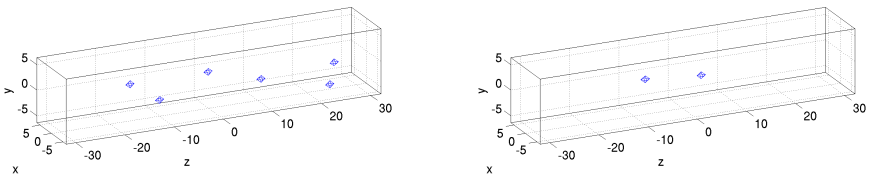
$t = 100$ $t = 200$



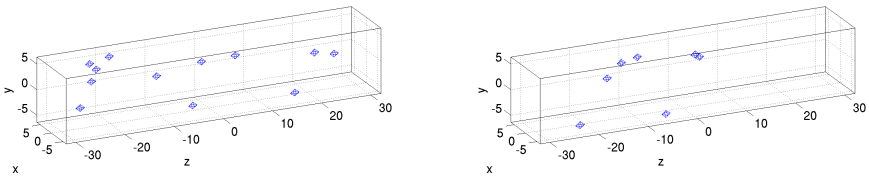
$t = 300$ $t = 400$



$t = 500$ $t = 600$

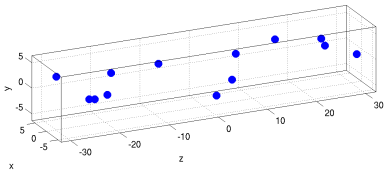


$t = 700$ $t = 800$

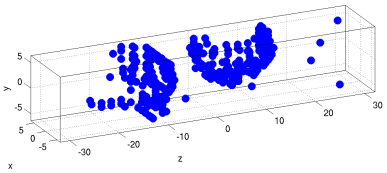


$t = 900$ $t = 1000$

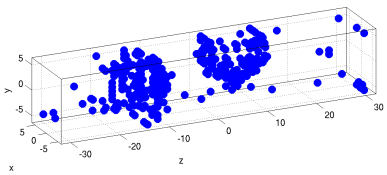
FIG. 4.3. *Isosurface plot of calcium concentration in cell with* $I_{SR} = 10$ *pA.*



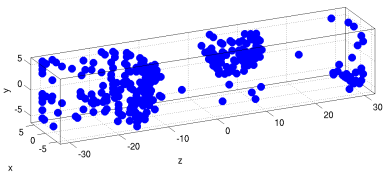
$t = 100$



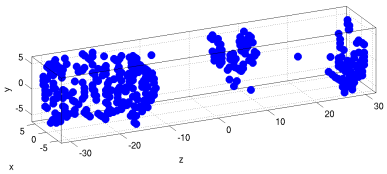
$t = 200$



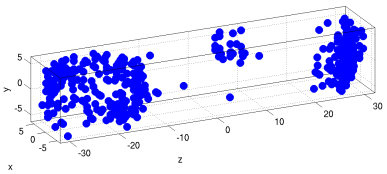
$t = 300$



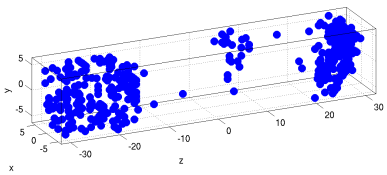
$t = 400$



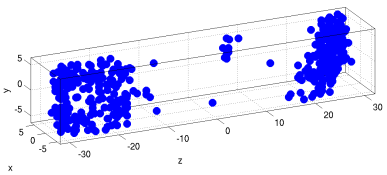
$t = 500$



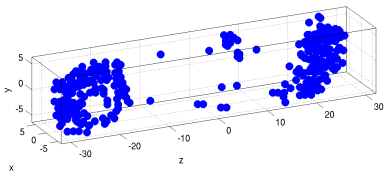
$t = 600$



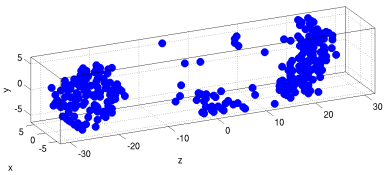
$t = 700$



$t = 800$

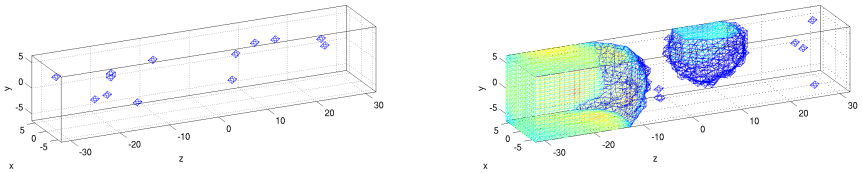


$t = 900$

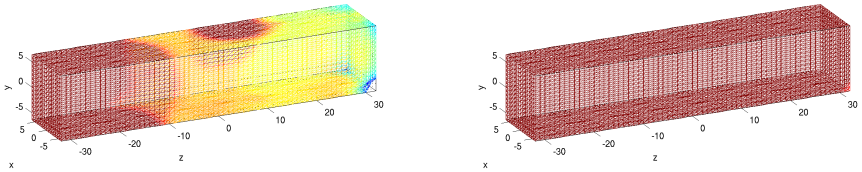


$t = 1000$

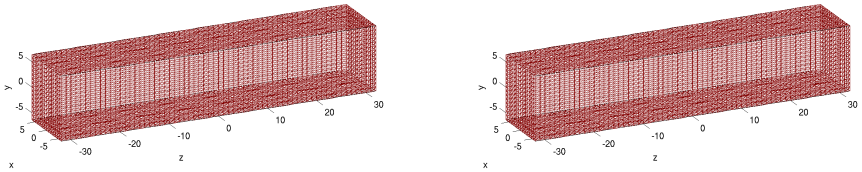
FIG. 4.4. *Open calcium release units throughout cell with* $I_{SR} = 20\ pA$.



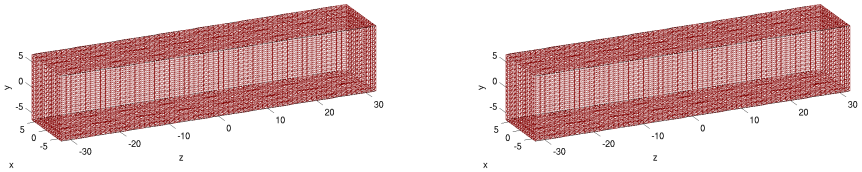
$t = 100$ $t = 200$



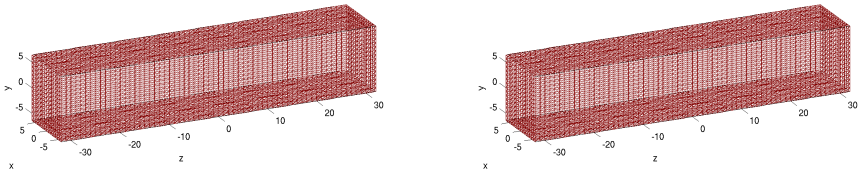
$t = 300$ $t = 400$



$t = 500$ $t = 600$



$t = 700$ $t = 800$



$t = 900$ $t = 1000$

FIG. 4.5. *Isosurface plot of calcium concentration in cell with* $I_{SR} = 20$ *pA.*

REFERENCES

[1] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, 1997.
[2] M. K. Gobbert, *Configuration and performance of a Beowulf cluster for large-scale scientific simulations*, Comput. Sci. Eng., 7 (2005), pp. 14–26.
[3] M. K. Gobbert, M. Kružík, and T. I. Seidman, *Numerical approximation of a heat equation with measure-valued data*. In preparation.
[4] M. K. Gobbert, T. I. Seidman, and R. J. Spiteri, *A non-negativity preserving Newton method for high-order implicit time stepping*. In preparation.
[5] A. L. Hanhart, M. K. Gobbert, and L. T. Izu, *A memory-efficient finite element method for systems of reaction-diffusion equations with non-smooth forcing*, J. Comput. Appl. Math., 169 (2004), pp. 431–458.
[6] W. Hundsdorfer and J. Verwer, *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*, vol. 33 of Springer Series in Computational Mathematics, Springer-Verlag, 2003.
[7] L. T. Izu, J. R. H. Mauban, C. W. Balke, and W. G. Wier, *Large currents generate cardiac Ca^{2+} sparks*, Biophysical Journal, 80 (2001), pp. 88–102.
[8] L. T. Izu, S. A. Means, J. N. Shadid, Y. Chen-Izu, and C. W. Balke, *Interplay of ryanodine receptor distribution and calcium dynamics*, Biophysical Journal, 91 (2006), pp. 95–112.
[9] L. T. Izu, W. G. Wier, and C. W. Balke, *Evolution of cardiac calcium waves from stochastic calcium sparks*, Biophysical Journal, 80 (2001), pp. 103–120.
[10] M. Matsumoto and T. Nishimura, *Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation, 8 (1998), pp. 3–30.
[11] A. Quarteroni and A. Valli, *Numerical Approximation of Partial Differential Equations*, vol. 23 of Springer Series in Computational Mathematics, Springer-Verlag, 1994.
[12] D. L. Ropp, J. N. Shadid, and C. C. Ober, *Studies of the accuracy of time integration methods for reaction-diffusion equations*, J. Comput. Phys., 194 (2004), pp. 544–574.
[13] A. Sandu, *Positive numerical integration methods for chemical kinetic systems*, J. Comput. Phys., 170 (2001), pp. 589–602.
[14] L. F. Shampine and M. W. Reichelt, *The MATLAB ODE suite*, SIAM J. Sci. Comput., 18 (1997), pp. 1–22.
[15] L. F. Shampine, S. Thompson, J. A. Kierzenka, and G. D. Byrne, *Non-negative solutions of ODEs*, Appl. Math. Comput., 170 (2005), pp. 556–569.
[16] V. Thomée, *Galerkin Finite Element Methods for Parabolic Problems*, vol. 25 of Springer Series in Computational Mathematics, Springer-Verlag, second ed., 2006.