A NON-NEGATIVITY PRESERVING NEWTON METHOD FOR HIGH-ORDER IMPLICIT TIME STEPPING

MATTHIAS K. GOBBERT*, MICHAEL MUSCEDERE* , THOMAS I. SEIDMAN* , AND RAYMOND J. SPITERI†

Abstract. Large classes of mathematical models approximate the evolution of quantities that are inherently non-negative because of what they represent physically, e.g., chemical concentrations or populations. However, even if a mathematical model is guaranteed to have a non-negative solution, the non-negativity of a numerical solution is often destroyed by the discretization of the mathematical model. In the case of implicit time discretizations of ordinary differential equations (ODEs), the non-negativity of the numerical solution is often destroyed by the truncated Newton method used to solve them. We propose a modification to standard implicit initial-value problem solvers that guarantees the non-negativity of the numerical solution and all intermediate Newton iterates by controlling the size of the Newton step. The algorithm is implemented in the context of the numerical differentiation formulas, a fully implicit family of high-order time-stepping methods, for a general system of first-order ODEs with mass matrix. The effectiveness of the method is demonstrated through comparisons against other non-negativity preserving methods on the classical Robertson kinetics problem and on a three-species reaction-diffusion system with transient and moving internal layers.

Key words. Numerical differentiation formulas, Newton method, reaction-diffusion equations, method of lines, finite difference method.

AMS subject classifications. 35K57, 65L06, 65M06, 65M50, 90C53.

1. Introduction.

Large classes of mathematical models approximate the time evolution of quantities that are inherently non-negative because of what they represent physically, e.g., chemical concentrations or populations. However, even if it can be guaranteed that a mathematical model has a non-negative solution, the non-negativity of its numerical solution is often destroyed by the discretization of the mathematical model, for instance by the time discretization of an ordinary differential equation (ODE) with a time step that exceeds the radius of monotonicity; see, e.g., [13, 24] for extensive discussions. In particular, if an implicit time-stepping method is used for the time evolution, so that a system of non-linear algebraic equations must be solved at each time step, then the process of approximating the solution (typically by some form of Newton's method) may destroy the non-negativity property of the numerical solution, even if the non-linear algebraic equations have a non-negative exact solution [19, page 592].

In this paper we propose an algorithm that modifies the Newton iteration inside an implicit high-order time-stepping method to guarantee non-negativity of the resulting numerical solution and all intermediate iterates. Our algorithm is designed for the general initial-value problem (IVP)

$$M\,\frac{dy}{dt} = f(t,y), \quad 0 < t \le t_{\text{fin}}, \quad y(0) = y_{\text{ini}}, \tag{1.1}$$

where $y(t)$ is a vector of time-dependent functions. Here, M denotes a mass matrix, such as would result from the use of the finite element method as the spatial discretization in a method-of-lines approach to discretize a PDE. We assume that

*Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, U.S.A. ({gobbert,mmusce1,seidman}@math.umbc.edu).

†Department of Computer Science, University of Saskatchewan, 110 Science Place, Saskatoon, SK S7N 5C9, Canada (spiteri@cs.usask.ca). The research of this author was supported in part by grants from MITACS, Martec, Ltd., and NSERC Canada.

M is non-singular and constant for now and that the given problem (1.1) is stiff. For reliability and efficiency of the code, it is vital that the ODE solver include dependable mechanisms for error estimation as well as automatic step size and order control. Well-known and recommended IVP codes for this problem include VODE [3], CVODE and IDA in SUNDIALS [10], and DASSL [2], which implement the family of backward differentiation formulas (BDFk), as well as RADAU5 [9], which implements an implicit Runge–Kutta method based on 3-stage Radau collocation. We use the `ode15s` function in Matlab [16, 23] as the starting point for our development because it implements the attractive family of numerical differentiation formulas (NDFk) that generalizes the BDFk family and has potential for greater efficiency [8, 23]. The well-known paper by Shampine and Reichelt [23] explains Matlab's ODE suite in detail.

For large classes of problems, such as reaction-diffusion equations, it has been established analytically that their mathematical models in the form of ODEs, including those resulting from appropriate spatial discretizations of PDEs, have non-negative solutions [13, Chapter 1] . Thus, we consider problems of the form (1.1) that have a non-negative solution $y(t)$ for an appropriately defined non-negative vector of initial conditions y_{ini}. Accordingly we also assume the existence of a suitable non-empty neighborhood $\mathcal{N}^+$ of non-negative solutions around $y(t)$ for all $t \in [0, t_{\text{fin}}]$.

The standard approach for handling non-negativity in a numerical solution produced by an IVP code is to perform multiple runs with increasingly tighter error tolerances, as suggested for example in the User Documentation for CVODE [12, Subsection 5.5.2]. The idea here is that as the error tolerances are tightened, the numerical solution should ultimately lie in $\mathcal{N}^+$. However, this approach is not always applicable. For example, it is not always possible to sensibly handle negative values in the definition of $f(t, y)$ or its Jacobian, e.g., if these functions contain logarithmic terms or square roots. Moreover, some problems become unstable for negative solutions of any size. It should also be noted that eliminating negative solutions entirely with this approach may incur a significant increase in computational cost because the tighter tolerances force smaller time steps.

For a given sequence of step sizes, it is also possible for a numerical solution y_n to exit $\mathcal{N}^+$ at some $t^- \in (0, t_{\text{fin}})$. In such a case, there may be no $\Delta t > 0$ for which the local solution (i.e., the exact solution starting at y_n) re-enters $\mathcal{N}^+$. In other words, the non-linear system for the numerical solution at the next time step may no longer admit a non-negative solution. In principle, it is possible to go back to some time before t^- (perhaps even as far as $t = 0$) and, with another sequence of (presumably smaller) time steps, produce another numerical solution that would be in $\mathcal{N}^+$ at t^-. However, this requires *global* error control and is not a feature of the most popular IVP codes, which rely on *local* error control. This is an understood and accepted risk of using local error control. We propose here a strategy that will nonetheless allow these codes to be used and efficiently produce non-negative solutions.

One of the first ways proposed to deal with preventing negative solution values in the context of an arbitrary time-stepping method is the so-called *clipping* method, in which negative solution components are set to zero whenever they occur. For reaction systems, this approach has the disadvantage that it destroys mass conservation: mass is effectively added to the system whenever a negative concentration is increased to zero [19]. It also adversely affects time stepping and error estimation algorithms.

An additional problem with clipping is that unless the Jacobian of $f(t, y)$ in (1.1) is re-evaluated after the solution is clipped, its slope in the next Newton step will still point "downwards," and the next iterate will likely yield negative solution values again. Partially motivated by this observation, a strategy described as *constraint-following* is proposed in [24] for the case $M = I$ in (1.1), where the components in $f(t, y)$ that correspond to zero solution components are "redefined" to be zero in the case of a negative slope, and thus the numerical solution will "follow" the constraint. The solution only moves off the constraint again if the slope becomes positive. This strategy has been implemented in several of Matlab's ODE solvers including `ode15s` (starting with Release R14 Service Pack 3). The approach implemented there incorporates part of the non-negativity preservation into the test for accepting or rejecting an ODE solution, i.e., the final iterate of the Newton solver. This makes the method efficient in that it incurs cost only when negative components are encountered. However, because it works as part of the ODE error control mechanism, it does not apply to intermediate Newton iterates; i.e., these intermediate iterates can still have negative components, and this can be problematic if $f(t, y)$ or its Jacobian cannot handle them.

The IVP code IDA in the SUNDIALS suite and its predecessor DASSL have some facility for dealing with enforcement of non-negativity in the numerical solutions they produce. These codes clip undesirable negative solution components provided their total size is smaller than the tolerance used in the Newton iteration. The idea is that the clipped solution remains an equally valid numerical solution, since the clipping error is no larger than the ODE error. If the size of the clip would have to be too large, the Newton iteration is deemed to have failed and steps are taken accordingly, e.g., the step size is reduced, etc. However, this should not be seen as a general purpose method for dealing with non-negativity preservation. Indeed the documentation for DASSL gives the advice: "If you know that the solutions to your equations will always be nonnegative, it may help to set this parameter. However, it is probably best to try the code without using this option first, and only to use this option if that doesn't work very well." We note that in this approach the user does not have direct control over the amount of non-negativity that can be tolerated in the numerical solution.

Finally we mention the approach suggested in [19] to project the solution into its non-negative domain via a constrained optimization. Drawbacks of this approach include the complication of its implementation and the computational cost associated with solving an optimization problem within an ODE solver.

To ensure the non-negativity of all Newton iterates, our algorithm limits the length of each Newton step in a technique that turns out to be similar to that described in [10, pages 373–374] for the non-linear solver KINSOL in SUNDIALS. The idea is that if any solution component becomes negative, we damp the Newton step with the largest factor such that the solution components do not become more negative than a user-prescribed amount $-1 \ll -\varepsilon^{(\text{neg})} < 0$, and then clip any remaining negative components; the clipped components are necessarily no greater than $\varepsilon^{(\text{neg})}$ in magnitude. An obvious advantage of this approach is that $\varepsilon^{(\text{neg})}$ explicitly indicates the acceptable amount of clipping error and implicitly ensures the damping parameter in the Newton step does not vanish. Moreover the right-hand side function $f(t, y)$ and its Jacobian are never called with negative arguments.

The use of such an algorithm inside a general IVP solver is new. Besides those already mentioned, in principle it may also be possible to reformulate a given problem as a differential-algebraic equation (DAE) where the algebraic equations are the *active constraints*, e.g., variables that are zero. However, DAEs are not ODEs, and it is more satisfying to impose non-negativity as an *invariant* (an inherent property of the exact solution of the ODE) rather than as a *constraint* (an otherwise arbitrary property of the solution). Additional drawbacks of the DAE approach include a complicated and expensive detection and accounting of the active constraints. Arguably in some situations, this may be the only way to proceed; however for the problem (1.1) for which a non-negative solution is known to exist, a specialized technique as we describe here may be more desirable.

Theoretical analyses of convergence for Newton, truncated Newton, and inexact Newton methods are available, e.g., [4, 5, 6, 7, 15]. These generally require that the initial guess be "close enough" to the solution. These analyses apply to a Newton method inside an ODE solver, if one assumes that the ODE solver can decrease the time step in response to convergence problems of the Newton method and thus implicitly improve the quality of the initial guess. The analyses apply then in the same way to our proposed method, because when the time step is sufficiently reduced, such that the solution stays in $\mathcal{N}^+$, no damping occurs. We note that a rationale for considering damping of the step size rather than any more general inexact Newton method is the preservation of mass conservation. For instance standard Runge-Kutta and linear multistep methods conserve mass (and other linear invariants) [21, 22], and changing the length of the Newton step does not destroy this property. As we show below, we do not change the Newton convergence tests, the error control, or the step-size or order selection. As in the standard usage, the user would balance tolerance and time step as needed to ensure convergence of the Newton method.

To study the effectiveness and efficiency of our method in practice, we present test calculations for two examples. The first is the well-known *Robertson problem* [18], which is a stiff system of three reaction ODEs. Computation of the solution to steady state constitutes a significant challenge for automatic step size control algorithms; because of this, it has been used as example in textbooks [9, pages 3 and 157], [1, page 61], and in the documentation for ODE packages such as EPISODE [11], VODE [3], CVODE in SUNDIALS [10], and Matlab's `ode15s` function [16, see function `hb1ode`]. This example is also interesting in the context of non-negativity preservation because it is known that for final times on the order of 10^{11}, one or more solution components can become negative, causing the solution to blow up [9, page 157], and leading to failure of the solver.

The second example is a system of ODEs obtained by spatially discretizing a system of three reaction-diffusion equations. The problem is characterized by moving internal layers at the interfaces between regions characterized by the relative size of the concentrations of two of the reactants, so we refer to it briefly as the *interface problem*. In the method-of-lines approach, the resulting problem is a system of stiff nonlinear ODEs. Similar to the Robertson problem, the solution has mostly smooth behavior, but it also has localized sharp transients [17, 25], so the use of a dependable error control and automatic step size selection algorithm is vital for a reliable and efficient solution of the problem. The two examples allow us to analyze different features of the strategies under consideration: The Robertson problem conserves mass; hence it is useful to study the effect of the different non-negativity preservation strategies on this property. However this problem is too small to observe meaningful computation times. Accordingly it is interesting to consider the more computationally demanding interface problem.

The solutions of the two examples represent chemical concentrations and hence should be non-negative for physical reasons. Moreover, it has been established theoretically that both problems maintain non-negativity [13, Chapter 1]. These examples have been chosen for the tests here so that all methods described for non-negativity preservation can be used without breaking down, thus allowing a full comparison to our proposed approach. To this end, we have deliberately selected problems which do not have any terms that would become undefined upon encountering negative solution components, so that the ODE solver can proceed to the final time in all cases, in order to allow the full comparison of all solvers considered. As discussed above however, catastrophic failure in the form of blow-up of the solution because of negative components can still and does occur for the Robertson problem. Also, although our method applies to (1.1), we restrict ourselves to $M = I$ so that Matlab's non-negativity preservation technique can be applied. We also analyze the impact of other method parameters on the effectiveness and efficiency of the ODE solver.

The remainder of the paper is organized as follows. Section 2 reviews the NDFk methods implemented in `ode15s` and explains in detail how our algorithm is implemented within it. The subsequent sections 3 and 4 present the Robertson and interface problems, respectively, with detailed descriptions and analyses of the results from the comparisons between the different strategies for non-negativity preservation. To test the methods, we compare several other choices, such as choices for the initial guess of the Newton iteration, of component-wise vs. norm-wise error control, and whether to force a Jacobian update for every linear solve. Finally, section 5 summarizes the conclusions from the numerical tests.

2. Time Discretization and Non-Negativity Preservation.

2.1. Numerical Differentiation Formulas. We review the NDFk following [23] but generalized to (1.1) with a mass matrix. Thus, the following formulas actually detail the code implemented in the Matlab function `ode15s`, using a notation that is inspired by, but slightly modified from [23]. Throughout the development, we leave other parts of the code unchanged, in particular the error control and automatic time step size and order selection algorithms.

The NDFk methods generalize the well-known BDFk methods. The BDFk methods approximate the derivative in (1.1) by backward difference approximations of order k. With pseudo-constant time steps Δt, we have $y'(t_{n+1}) \approx (1/\Delta t) \sum_{m=1}^{k} (1/m)\, \nabla^m y_{n+1}$, with$\nabla^m y_\ell := \nabla^{m-1} y_\ell - \nabla^{m-1} y_{\ell-1}$ for $m \geq 1$ and $\nabla^0 y_\ell := y_\ell$. The NDF$k$ are defined by adding $-\alpha_k \gamma_k M(y_{n+1} - p_n)$ to the BDFk to get

$$M \left(\sum_{m=1}^{k} \frac{1}{m} \nabla^m y_{n+1} \right) - \Delta t\; f(t_{n+1}, y_{n+1}) - \alpha_k\, \gamma_k\, M \left(y_{n+1} - p_n \right) = 0, \tag{2.1}$$

with $\gamma_k := \sum_{j=1}^{k} \frac{1}{j}$. The quantity defined by

$$p_n := y_n + \sum_{m=1}^{k} \nabla^m y_n \tag{2.2}$$

can be interpreted as a predictor for y_{n+1} at the new time $t = t_{n+1} = t_n + \Delta t$, using the solution and approximations to its derivatives at $t = t_n$. The truncation error of the NDFk method is $(\alpha_k \gamma_k + \frac{1}{k+1})\, (\Delta t)^{k+1} y^{(k+1)}(t_{n+1})$ and has the same order (i.e., $k+1$) as that of BDFk [23]. The parameter α_k can now be chosen for each method order $1 \leq k \leq 5$ to make the method more efficient, and [23] explains the values that appear in `ode15s`. The notation for the predictor (2.2) in [23] is $y_{n+1}^{(0)}$ because its use as initial guess for the Newton method is hard-coded in `ode15s`; we introduce a separate notation for the predictor here so that we can choose another initial guess for the Newton method later.

2.2. The Newton Method inside NDFk. The fully implicit time discretization (2.1) constitutes a non-linear system of equations for y_{n+1}. Using the identity $\sum_{m=1}^{k} \frac{1}{m} \nabla^m y_{n+1} = \gamma_k \big(y_{n+1} - p_n\big) + \sum_{m=1}^{k} \gamma_m \nabla^m y_n$, we collect terms independent of y_{n+1} and write (2.1) as a root-finding problem for y_{n+1}:

$$f^{(newt)}(y_{n+1}) := M \left(y_{n+1} - p_n\right) + M\Psi_n - \frac{\Delta t}{(1-\alpha_k)\gamma_k}\, f(t_{n+1}, y_{n+1}) = 0,$$

with $\Psi_n := \frac{1}{(1-\alpha_k)\gamma_k} \sum_{m=1}^{k} \gamma_m \nabla^m y_n$. The Newton method then reads: Choose the initial guess $y_{n+1}^{(0)}$, then iterate for $i = 0, 1, 2, \ldots$

$$\begin{aligned} &\text{Solve } \left(J^{(newt)}(y_{n+1}^{(i)})\right) \Delta^{(i)} = -f^{(newt)}(y_{n+1}^{(i)}) \text{ for } \Delta^{(i)}, \\ &\text{Update } y_{n+1}^{(i+1)} = y_{n+1}^{(i)} + \Delta^{(i)}, \end{aligned} \tag{2.3}$$

where the Jacobian $J^{(newt)}(y_{n+1})$ of $f^{(newt)}(y_{n+1})$ with respect to unknown vector y_{n+1} is given by

$$J^{(newt)}(y_{n+1}) := \nabla_{y_{n+1}} f^{(newt)}(y_{n+1}) = M - \frac{\Delta t}{(1-\alpha_k)\gamma_k}\, J(t_{n+1}, y_{n+1}) \tag{2.4}$$

involving the Jacobian $J(t,y) := \nabla_y f(t,y)$ of the right-hand side function in (1.1).

The code must also decide whether or not to accept $y_{n+1}^{(i+1)}$, and to this end one needs an ODE error estimator for the term $(\Delta t)^{k+1} y^{(k+1)}(t_{n+1}) \approx \nabla^{k+1} y_{n+1}$ in the truncation error. From the definition of p_n from (2.2) follows the alternative expression $\nabla^{k+1} y_{n+1} = y_{n+1} - p_n$ for the approximation to the truncation error explicitly involving y_{n+1}. By introducing $d^{(i+1)} := \nabla^{k+1} y_{n+1}^{(i+1)} = y_{n+1}^{(i+1)} - p_n = y_{n+1}^{(i)} + \Delta^{(i)} - p_n = d^{(i)} + \Delta^{(i)}$ for every Newton iterate $y_{n+1}^{(i+1)}$ and $d^{(0)} := y_{n+1}^{(0)} - p_n$, one derives a method that gives the needed ODE error estimator. In turn, one can re-write the Newton update in (2.3) to use $d^{(i+1)}$ in its calculation and

$$\begin{aligned} d^{(i+1)} &= d^{(i)} + \Delta^{(i)}, \\ y_{n+1}^{(i+1)} &= p_n + d^{(i+1)}. \end{aligned} \tag{2.5}$$

These formulas are used in `ode15s` to simultaneously compute the Newton update $y_{n+1}^{(i+1)}$ and its ODE error estimator $d^{(i+1)}$.

As initial guess for the Newton iteration, one choice is the solution at the previous time step: $y_{n+1}^{(0)} = y_n$. In this case, $d^{(0)} = y_{n+1}^{(0)} - p_n$ needs to be computed from its definition. But the predictor (2.2) uses additional information about the derivatives, and thus the initial guess $y_{n+1}^{(0)} = p_n$ is expected to lead to better performance of the non-linear solver. In this case, $d^{(0)} = y_{n+1}^{(0)} - p_n \equiv 0$. To allow us to study the effect of both initial guesses in our numerical studies, we write the algorithm in the following form: Choose $y_{n+1}^{(0)}$ either as p_n or y_n, compute $d^{(0)} = y_{n+1}^{(0)} - p_n$, then iterate for $i = 0, 1, 2, \ldots$

$$\begin{aligned} &b^{(i)} = \tfrac{\Delta t}{(1-\alpha_k)\gamma_k}\, f(t_{n+1}, y_{n+1}^{(i)}) - M\left(\Psi_n + d^{(i)}\right), \\ &\text{Solve } \left(M - \tfrac{\Delta t}{(1-\alpha_k)\gamma_k}\, J(t_{n+1}, y_{n+1}^{(i)})\right) \Delta^{(i)} = b^{(i)} \text{ for } \Delta^{(i)}, \\ &d^{(i+1)} = d^{(i)} + \Delta^{(i)}, \\ &y_{n+1}^{(i+1)} = p_n + d^{(i+1)}. \end{aligned} \tag{2.6}$$

This form of the algorithm brings out how the coefficient functions $f(t, y)$ and $J(t, y)$ of (1.1) enter and is useful to explain possible trade-offs between accuracy and efficiency: The linear solve in the second step of (2.6) is accomplished by computing an LU decomposition of the iteration matrix $M^{(\text{iter})} := M - \frac{\Delta t}{(1-\alpha_k)\gamma_k} J$. The efficiency of this approach lies in re-using the decomposition for several Newton iterations and in fact for several (potentially many) time steps. In addition, Matlab's `ode15s` code further minimizes the number of Jacobian evaluations by holding J constant in memory until the error control algorithm requests a re-evaluation. This means that when the LU decomposition of $M^{(\text{iter})}$ is re-computed in response to a change in the step size Δt or the ODE method order k (hence changing α_k and γ_k in $M^{(\text{iter})}$), the Jacobian may not be up-to-date at the current time. This approach minimizes the number of times that J is evaluated and is thus appropriate if this is the costliest step in the algorithm, in particular compared to the LU decomposition. Since we will use an analytic expression for $J(t, y)$, the evaluation of the Jacobian is cheap, and the LU decomposition is the costliest step of the algorithm. Hence, we also test a different compromise in our numerical studies, in which the Jacobian is always re-evaluated whenever an LU decomposition of $M^{(\text{iter})}$ is required, that is, whenever the error control mechanism either requests the re-evaluation of J or when it changes Δt or k.

2.3. A Non-Negativity Preserving Newton Method. The classical Newton iteration may produce an iterate $y_{n+1}^{(i+1)} = y_{n+1}^{(i)} + \Delta^{(i)}$ with negative components even if $y_{n+1}^{(i)}$ is non-negative because $\Delta^{(i)}$ from the linear solve in (2.6) is not restricted in size or sign. To prevent the introduction of negative components, we introduce a damping parameter $0 < s_i \leq 1$ in the Newton update as $y_{n+1}^{(i+1)} = y_{n+1}^{(i)} + s_i\,\Delta^{(i)}$ that guarantees that no component of the new Newton iterate $y_{n+1}^{(i+1)}$ is smaller than $-\varepsilon^{(\text{neg})}$. The choice of $\varepsilon^{(\text{neg})} > 0$ ensures that s_i never vanishes; i.e., the Newton iteration is guaranteed not to stall. Reformulating the terms again as for (2.5), our damping algorithm can be written in a form comparable to (2.6) as follows: Choose $y_{n+1}^{(0)}$ either as p_n or y_n, compute $d^{(0)} = y_{n+1}^{(0)} - p_n$, then iterate for $i = 0, 1, 2, \ldots$

$$\begin{aligned}
&b^{(i)} = \tfrac{\Delta t}{(1-\alpha_k)\gamma_k}\, f(t_{n+1}, y_{n+1}^{(i)}) - M\left(\Psi_n + d^{(i)}\right),\\
&\text{Solve } \left(M - \tfrac{\Delta t}{(1-\alpha_k)\gamma_k}\, J(t_{n+1}, y_{n+1}^{(i)})\right) \Delta^{(i)} = b^{(i)} \text{ for } \Delta^{(i)},\\
&s_i = \max\{s \in (0,1] : y_{n+1}^{(i)} + s\,\Delta^{(i)} \geq -\varepsilon^{(\text{neg})}\},\\
&d^{(i+1)} = d^{(i)} + s_i\,\Delta^{(i)},\\
&y_{n+1}^{(i+1)} = p_n + d^{(i+1)}.
\end{aligned} \tag{2.7}$$

We complement this damping with two ideas from the constraint-following algorithm in `ode15s`. First, any remaining negative components are set to zero now, but by construction these components are of size less than $\varepsilon^{(\text{neg})}$. Second, we set the derivative approximations $\nabla^m y_{n+1}$ to zero for these components to help with the non-negativity of the predictor of the next step.

To ensure non-negativity of all Newton iterates by the above construction, it is also necessary to ensure that the initial guess be non-negative. The general initial guess $y_{n+1}^{(0)} = p_n$ from (2.2) can have negative components. If so, we try again with an initial guess based on a predictor with shorter memory than (2.2) by computing $y_{n+1}^{(0)} = y_n + \nabla^1 y_n$. If this still yields negative components, then we compute a damped predictor by the same construction described above with y_n in the role of $y_{n+1}^{(i)}$ and

$\nabla^1 y_n$ in the role of $\Delta^{(i)}$.

Overall, these steps construct a non-negative Newton iterate with explicit user control of the acceptable mass error because the components set to zero in the second step are no larger than the user-supplied $\varepsilon^{(\text{neg})}$. The proposed algorithm ensures that $f(t, y)$ and $J(t, y)$ are never evaluated with y having negative components. Although somewhat more computationally expensive, this approach maintains the philosophy of [24] regarding efficiency in that it costs nothing if there are no negative components, and it is reasonably inexpensive if there are. Moreover, this approach is easily implemented in the Newton method inside an implicit ODE solver of any method order k.

The Newton iterate $y_{n+1}^{(i+1)}$ is accepted as converged when the norm of $\Delta^{(i)}$ (not including the damping factor s_i) is less than the usual Newton tolerance defined from the user-supplied absolute and relative tolerances on the numerical solution. In other words, when accepted as converged, the numerical solution obtained via the proposed damping algorithm is as valid as the one obtained by the standard Newton iteration. Hence the order of the truncation error is preserved in the same way as for the standard Newton implementation, and the rest of the time-stepping algorithm (e.g., the choice of step size and order, etc.) can proceed without modification. In particular, the quantity $d^{(i+1)} = d^{(i)} + s_i\,\Delta^{(i)}$ is the correct ODE error estimator associated with the new solution $y_{n+1}^{(i+1)}$, following an analogous derivation as the one for (2.5). If a converged solution cannot be found within the maximum number of Newton iterations allowed, the time-stepping algorithm still proceeds without modification, e.g., the Jacobian is re-evaluated and/or the time step is reduced. It is possible for the damping algorithm to cause the code to halt because of non-convergence of the Newton iteration. However, this can happen in any other standard IVP code that uses local error control. In this case, the lack of convergence may be because the local solution is not able to re-enter $\mathcal{N}^+$, as discussed in section 1. In practice, the user is expected to react by changing method parameters such as ODE tolerances, the minimum acceptable ODE step size, among others, and in our algorithm additionally the amount of acceptable negativity $\varepsilon^{(\text{neg})}$. In our fully instrumented version of Matlab's `ode15s` function, the user can additionally control several other aspects of the methods, such as which initial guess is used for the Newton method, whether the errors are controlled component-wise or norm-wise, and whether the Jacobian is updated whenever the LU factorization is computed or not.

3. The Robertson Problem.

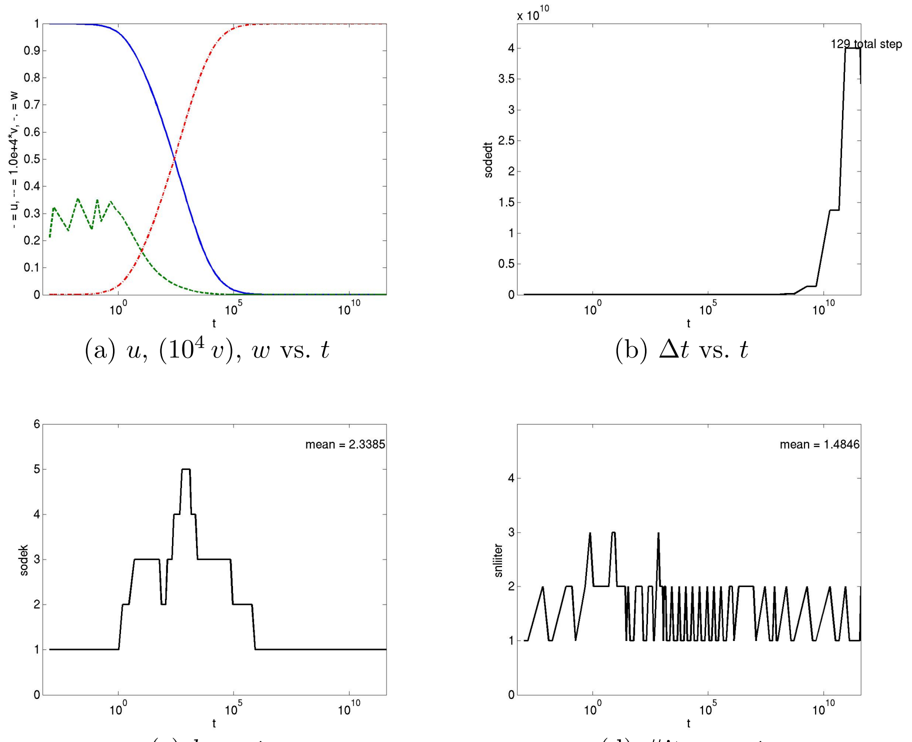
The Robertson problem [9, 18] describes chemical reactions among three reactants A, B, and C as

$$\mathrm{A} \xrightarrow{0.04} \mathrm{B}, \quad 2\,\mathrm{B} \xrightarrow{3\cdot 10^7} \mathrm{B} + \mathrm{C}, \quad \mathrm{B} + \mathrm{C} \xrightarrow{10^4} \mathrm{A} + \mathrm{C}.$$

Introducing $u(t)$, $v(t)$, $w(t)$ as the chemical concentrations of the species A, B, C, respectively, the evolution of the concentrations is described by the ODE system

$$\begin{aligned} u_t &= -0.04\,u + 10^4\,v\,w, & u(0) &= 1, \\ v_t &= 0.04\,u - 10^4\,v\,w - 3\cdot 10^7\,v^2, & v(0) &= 0, \\ w_t &= 3\cdot 10^7\,v^2, & w(0) &= 0. \end{aligned} \tag{3.1}$$

We choose to consider the final time $t_{\text{fin}} = 4 \cdot 10^{11}$, which is beyond the time interval over which many IVP solvers are stable [9, 16]. To phrase the problem in the standard form $y' = f(t, y)$, we define $y(t) = [u(t), v(t), w(t)]^T$.



(a) u, $(10^4\, v)$, w vs. t

(b) Δt vs. t

(c) k vs. t

(d) #iter vs. t

FIG. 3.1. *Results for the Robertson problem. (a) Plots of the solution components u (solid), $10^4\, v$ (dashed), and w (dash-dotted) vs. t, (b) ODE time step Δt vs. t, (c) ODE method order k vs. t, (d) number of Newton iterations vs. t. (The results shown in this figure are computed by the NDFk method with non-negativity preservation using damping, with the predictor as initial guess, with Jacobian update forced whenever Δt or k change, and with norm-wise error control.)*

Figure 3.1(a) plots the three solution components as functions of time on a logarithmic time scale; the second solution component is scaled by 10^4 to make it visible as done in, e.g., [16, function `hb1ode`]. Figures 3.1(b)–(d) show plots of some performance indicators. Figure 3.1(b) shows that the time steps Δt selected by the automatic step size control increase exponentially up to the maximum allowed value of $\Delta t_{\max}$. Figure 3.1(c) shows the method order k selected by the automatic order control in the NDFk family with $1 \le k \le 5$. Finally, Figure 3.1(d) shows the number of Newton iterations needed to solve the non-linear system of equations at each time step; typically, 1 or 2 Newton iterations are required.

Figure 3.1 shows results of one particular choice of method parameters, but the behavior for other (convergent) cases is qualitatively similar. To compare the effect of the different cases more precisely, Tables 3.1–3.4 list indicators that quantify the effectiveness and efficiency of the different method and parameter choices. Following the defaults for `ode15s` in Matlab, we use the NDFk method with $1 \le k \le 5$, a relative tolerance of 10^{-3}, an absolute tolerance of 10^{-6}, an initial time step $\Delta t_{\mathrm{ini}} = 5.48{\cdot}10^{-4}$, and a maximum $\Delta t_{\max} = t_{\mathrm{fin}}/10 = 4 \cdot 10^{10}$. The non-linear Newton solver uses an analytically supplied Jacobian matrix. The tolerance for accepting a Newton solution is $100\,\varepsilon_{\mathrm{mach}} \approx 2.22 \cdot 10^{-14}$, and the maximum number of Newton iterations is 4, again following the choices implemented in `ode15s`. We use $\varepsilon^{(\mathrm{neg})} = 10^{-12}$; we also tested the values 10^{-10} and 10^{-14} and observed comparable results. The focus of this work is on *comparing* the effect and cost of different non-negativity preservation methods in the context of an IVP solver with error control for a set of fixed ODE tolerances for all methods; thus no results for other tolerances are reported. For the methods with non-negativity preservation, tighter tolerances lead to higher computational cost, as expected. But for the case without non-negativity preservation, tests with different tolerances for the Robertson problem show in fact that one of the solution components always eventually becomes negative, and this is followed by blow-up of the solution (no matter how small the magnitude of the negative component). This blow-up behavior is only postponed, not eliminated, by the use of tighter tolerances, thus this is a good example of the need for non-negativity preservation.

Each table collects results (i) with no enforcement of non-negativity ("none") and with non-negativity enforced by (ii) clipping ("clip"), (iii) clipping as implemented by DASSL ("dassl"), (iv) constraint following ("constraint") in Matlab using the `NonNegative` option for all solution components, and (v) damping the Newton iterates ("damp") as described in section 2.3.

The first four quantities listed for each method quantify the effectiveness of the method, i.e., its ability to compute physically correct results. Specifically, we track the number of times that any *intermediate* Newton iterate contains a negative component as well as the value of the smallest negative component among all *intermediate* iterates over all times; these are reported in the first two rows as `nneg` and $\min(y)$. Next, we report $\max(y) = \|y(t)\|_\infty$ over all times; this value should be no larger than 1 for this problem. The Robertson problem conserves mass, and the total mass $m(t) := u(t) + v(t) + w(t)$ should satisfy $m(t) \equiv 1$; so we list the maximum of the error in total mass as $\max |m(t) - 1|$ over all times. In exact arithmetic, the NDFk (without any non-negativity preservation) should conserve mass. In finite-precision arithmetic, one thus expects mass error on the order of a small multiple of unit round-off. However, in cases where the solution does not converge (e.g., blows up), the formally reported mass error becomes meaningless and can have any value due to effects of cancellation of significant digits between the components that are blowing up. In any case, a solution with blow-up is meaningless, and its results are reported in the tables only for completeness.

The remaining results for each method quantify the efficiency of the method. Specifically, we report the number of (successful) time steps in `nsteps`, the number of failed time steps in `nfailed`, the number of evaluations of the ODE function $f(t, y)$ in `nfevals`, the number of evaluations of the Jacobian $J(t, y) = \nabla_y f(t, y)$ in `npds`, the number of LU decompositions in `ndecomps`, and the number of linear solves (using a pre-computed decomposition) in `nsolves`; these are the same statistics as reported by Matlab's `ode15s`. Additionally, in `nclips`, we report the number of times that one or more negative solution components are set to zero in "clip," "dassl," and "constaint" or the number of times that a Newton iteration is damped in "damp." Finally, the entries for mean(k) and mean(`it`) report the ODE method order used and the number of Newton iterations taken, respectively, averaged over all time steps. For the Robertson problem, we do not report any observed wall clock times because they are small.

We start in Table 3.1(a) with the NDFk method using settings that are default in Matlab for `ode15s`, i.e., component-wise error control (`NormControl` switched off) and with the predictor p_n as initial guess to the Newton iteration. This means that, in the cases of no enforcement and constraint-following, our code gives identical results

Table 3.1
Solution statistics for the Robertson problem with Jacobian update not *forced and* component-wise *error control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	654	0	0	4	0
$\min(y)$	–1.49e+08	0.0	0.0	–1.54e+02	0.0
$\max(y)$	1.49e+08	1.0	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	6.17e+00	1.40e–05	4.99e–07	2.58e–14	4.33e–15
`nsteps`	450	241	237	237	238
`nfailed`	164	20	18	18	18
`nfevals`	1113	479	461	464	463
`npds`	70	13	13	13	13
`ndecomps`	262	71	68	67	68
`nsolves`	1112	478	460	462	462
`nclips`	N/A	44	26	0	17
mean(k)	2.36	2.81	2.84	2.84	2.84
mean(`iter`)	1.69	1.81	1.80	1.79	1.79
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	0	0	0	0	0
$\min(y)$	0.0	0.0	0.0	0.0	0.0
$\max(y)$	1.0	1.0	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	1.03e–13	1.03e–13	1.03e–13	1.03e–13	1.03e–13
`nsteps`	218	218	218	218	218
`nfailed`	17	17	17	17	17
`nfevals`	568	568	568	569	568
`npds`	14	14	14	14	14
`ndecomps`	63	63	63	63	63
`nsolves`	567	567	567	567	567
`nclips`	N/A	0	0	0	0
mean(k)	2.74	2.74	2.74	2.74	2.74
mean(`iter`)	2.42	2.42	2.42	2.42	2.42

to `ode15s` without and with the `NonNegative` option used, respectively. As the large magnitudes of $\min(y)$ and $\max(y)$ show, the solution from `ode15s` blows up in the case of no enforcement and default settings of the tolerances. We point out that for a fixed final time, in principle there exists a sufficiently tight tolerance to avoid negative values and hence blow-up, but this tolerance gets tighter for larger final times, and all performance statistics deteriorate rapidly. Accordingly, our proposed algorithm aims to maintain the physical correctness of the solution for any final time without paying a large penalty in efficiency. Next we notice that clipping avoids negative solution values and hence prevents blow-up. However, a fairly significant amount of mass is gained. The clipping as implented in DASSL, which ensures that clipping remains within the ODE error, indeed has less mass error than standard clipping. The results for both constraint-following, which uses the `NonNegative` option in `ode15s`, and damping exhibit physically correct behavior, with mass being conserved to within round-off and essentially the same efficiency as clipping. However, we see here that the algorithm of constraint-following, which only considers the final Newton solution, can produce negative values in the intermediate Newton iterations.

Table 3.1(b) shows the results with the Newton method using y_n as initial guess

Table 3.2
Solution statistics for the Robertson problem with Jacobian update not *forced and* norm-wise *error control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	559	0	0	42	0
min(y)	–1.92e+08	0.0	0.0	–1.39e+03	0.0
max(y)	1.92e+08	1.0148	1.0013	1.0023	1.0
$\max\|m(t)-1\|$	8.39e+00	1.48e–02	1.26e–03	2.29e–03	6.67e–09
`nsteps`	310	147	145	175	140
`nfailed`	144	25	30	49	13
`nfevals`	781	297	295	376	278
`npds`	69	14	18	26	12
`ndecomps`	221	61	66	92	46
`nsolves`	780	296	294	374	277
`nclips`	N/A	89	64	0	18
mean(k)	2.00	1.91	1.99	1.98	2.32
mean(`iter`)	1.56	1.66	1.66	1.57	1.79
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	2	0	0	1	0
min(y)	–1.87e+00	0.0	0.0	–3.67e–03	0.0
max(y)	1.0	1.0	1.0	1.0	1.0
$\max\|m(t)-1\|$	1.70e–11	1.70e–11	1.70e–11	1.70e–11	1.70e–11
`nsteps`	127	127	127	127	127
`nfailed`	10	10	10	10	10
`nfevals`	300	300	299	301	301
`npds`	11	11	11	11	11
`ndecomps`	40	40	40	40	40
`nsolves`	299	299	298	299	300
`nclips`	N/A	1	1	0	2
mean(k)	2.30	2.30	2.30	2.30	2.30
mean(`iter`)	2.19	2.19	2.19	2.19	2.19

instead of p_n. We notice that `ode15s` without non-negativity enforcement no longer blows up. In fact, using the initial guess y_n for the Newton iterations allows the method to maintain non-negativity of the solution at all times without any enforcement. However, there is no theoretical guarantee of such behavior for the NDFk method and certainly not for the non-linear solver; hence one cannot rely on this behavior. This is demonstrated in section 4 for a larger system of ODEs, where this choice of initial guess produces negative solution values. Because there is no non-negativity to enforce, all other methods give identical performance statistics and only differences to within round-off in the physical quantities. We also note that the ODE-related efficiency for the cases with initial guess of y_n is slightly better than for the predictor p_n. This is somewhat counter-intuitive because the predictor p_n contains more information about the solution than y_n; this fact itself is borne out by the non-linear solver taking more iterations per ODE step as reported in mean(`it`).

Table 3.2 shows the results of the methods when norm-wise error control is used (`NormControl` on). Table 3.2(a) shows that the method with no non-negativity enforcement still blows up. The methods with non-negativity enforcement do not blow up, but all of them introduce non-negligible error in the total mass; in some cases,

TABLE 3.3
Solution statistics for the Robertson problem with Jacobian update forced and component-wise error *control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	569	0	0	5	0
$\min(y)$	–1.71e+08	0.0	0.0	–2.84e–03	0.0
$\max(y)$	1.71e+08	1.0	1.0	1.0	1.0
$\max\lvert m(t)-1\rvert$	6.70e+00	5.24e–08	5.24e–08	1.38e–14	1.38e–14
`nsteps`	447	226	226	232	226
`nfailed`	108	2	2	6	2
`nfevals`	858	298	298	311	296
`npds`	207	51	51	57	51
`ndecomps`	207	51	51	57	51
`nsolves`	858	297	297	309	295
`nclips`	N/A	22	22	0	8
mean(k)	2.27	2.64	2.64	2.60	2.64
mean(`iter`)	1.42	1.30	1.30	1.28	1.29
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	0	0	0	0	0
$\min(y)$	0.0	0.0	0.0	0.0	0.0
$\max(y)$	1.0	1.0	1.0	1.0	1.0
$\max\lvert m(t)-1\rvert$	4.97e–14	4.97e–14	4.97e–14	4.97e–14	4.97e–14
`nsteps`	222	222	222	222	222
`nfailed`	1	1	1	1	1
`nfevals`	421	421	421	422	421
`npds`	49	49	49	49	49
`ndecomps`	49	49	49	49	49
`nsolves`	420	420	420	420	420
`nclips`	N/A	0	0	0	0
mean(k)	2.68	2.68	2.68	2.68	2.68
mean(`iter`)	1.87	1.87	1.97	1.87	1.87

$\max(y)$ is even above its theoretical maximum of 1. DASSL clipping is an order of magnitude better than standard clipping. The constraint-following method suffers again from negative intermediate Newton iterates. The effectiveness and efficiency of damping is seen to be somewhat better than that of the other methods. As before, we notice in Table 3.2(b) that using the initial guess of y_n for the Newton iterations leads to better efficiency results. Some smaller, but still significant negative intermediate results appear in the cases with no non-negativity preservation and constraint-following. Overall, comparing Tables 3.1 and 3.2, we notice that norm-wise error control leads to somewhat less accurate physical results but also to significantly better efficiency.

Tables 3.1 and 3.2 report results for cases where the Jacobian $J(t,y)$ of the ODE function $f(t,y)$ is held constant in memory as long as possible and only re-evaluated when deemed necessary by the error control mechanism. This is the default implementation in `ode15s` and reflects the common situation that the evaluation of $J(t,y)$ is the most expensive part of the algorithm. The next most expensive cost is usually the LU decomposition of the iteration matrix $M^{(\text{iter})}$ that is necessary whenever either $J(t,y)$ is updated or when Δt or the method order k are changed by the error controller. This means that, in situations where Δt or k change and an LU decompo-

Table 3.4
Solution statistics for the Robertson problem with Jacobian update forced and norm-wise *error control.*

(a) Initial guess for Newton iteration: predictor p_n	no enforcement	clipping	dassl	constraint	damping
`nnegative`	30	0	0	19	0
$\min(y)$	–1.68e–03	0.0	0.0	–6.63e+00	0.0
$\max(y)$	1.0	1.0022	1.0005	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	8.88e–15	2.23e–03	4.79e–04	5.99e–15	4.44e–15
`nsteps`	139	143	151	155	129
`nfailed`	8	9	16	18	4
`nfevals`	203	212	237	245	201
`npds`	45	47	57	61	35
`ndecomps`	45	47	57	61	35
`nsolves`	202	211	236	243	200
`nclips`	N/A	52	52	0	15
mean(k)	1.73	1.74	1.63	1.65	2.34
mean(`iter`)	1.34	1.35	1.38	1.33	1.48

(b) Initial guess for Newton iteration: old solution y_n	no enforcement	clipping	dassl	constraint	damping
`nnegative`	0	0	0	0	0
$\min(y)$	0.0	0.0	0.0	0.0	0.0
$\max(y)$	1.0	1.0	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	4.01e–13	4.01e–13	4.01e–13	4.01e–13	4.01e–13
`nsteps`	127	127	127	127	127
`nfailed`	0	0	0	0	0
`nfevals`	224	224	224	225	224
`npds`	30	30	30	30	30
`ndecomps`	30	30	30	30	30
`nsolves`	223	223	223	223	223
`nclips`	N/A	0	0	0	0
mean(k)	2.27	2.27	2.27	2.27	2.27
mean(`iter`)	1.75	1.75	1.75	1.75	1.75

sition of $M^{(\text{iter})}$ is necessary, this decomposition may not use the latest Jacobian. This means that the error controller tries to minimize the number of Jacobian evaluations at the cost of more decompositions.

However, for this problem and an analytically supplied Jacobian, the evaluation of $J(t, y)$ incurs negligible cost. We modify `ode15s` to force an update of the Jacobian whenever Δt or k change so that $M^{(\text{iter})}$ is the most up-to-date whenever its LU decomposition is computed. This reflects the fact that the cost incurred by the LU decomposition of $M^{(\text{iter})}$ is the most expensive part of the method here and, to minimize the number of decompositions, we effectively supply more up-to-date physical information by updating the Jacobian more often.

Tables 3.3 and 3.4 report the results for the modified algorithm with the update of the Jacobian also forced whenever Δt or k change, while all other method parameters are the same as for Tables 3.1 and 3.2, respectively. We see that in all cases, the values reported for `npds` and `ndecomps` agree in Tables 3.3 and 3.4. These numbers lie in between those reported for `npds` and `ndecomps` in the corresponding previous tables, meaning that the cost has been shifted from LU decompositions to Jacobian evaluations, as intended. We note that the significance of this difference in cost becomes clearer in the next section, where computation times are large enough to bring out the overall advantage. We also notice that the number of ODE steps remains practically the same as before, but `nsolves` decreases in all cases against the corresponding previous cases, presumably as a result of the better physical information provided in $M^{(\mathrm{iter})}$. This immediately implies a decrease in `nfevals`, an observation that is also confirmed by lower values of mean(`it`). Using better physical information in $J(t, y)$ also improves the effectiveness of the methods. First, the method with no non-negativity enforcement no longer blows up in Table 3.4(a), although it still does in Table 3.3(a); this is counter-intuitive because one would have expected more accuracy with component-wise error control versus norm-wise error control. For the cases that converged and had a mass error larger than round-off, the accuracy of the mass conservation is improved in all cases in Tables 3.3 and 3.4 compared to Tables 3.1 and 3.2, respectively. Specifically, non-negativity preservation by constraint-following as well as by damping both give errors of the total mass that are small, whereas clipping can still incur much more significant errors in the mass. Although the behavior is improved, negative numbers still appear in some intermediate Newton iterates for the constraint-following method. The converged solution Table 3.4(a) with no non-negativity enforcement finally allows a comparison of its efficiency with those that have. We see that clipping, DASSL clipping, and constraint-following exhibit roughly comparable efficiency, whereas damping is slightly more efficient, apparently at the cost of a slightly larger mass error. In fact, damping in Table 3.4(a) is the most efficient method with $y_{n+1}^{(0)} = p_n$ in Tables 3.3(a) and 3.4(a) and nearly as efficient as the corresponding case in Table 3.2(a) but with much better mass error. Therefore, this is the preferred method among all methods considered here, and its results were used for the plots in Figure 3.1. In Tables 3.3 and 3.4, we see that using $y_{n+1}^{(0)} = y_n$ does not lead to negative solution components even with no non-negativity enforcement. However, there is no guarantee of this behavior in general, as is demonstrated in the next section. These cases again exhibit better ODE efficiency, but the non-linear solver uses again more iterations because y_n contains less information about the solution than p_n.

4. The Interface Problem.

This example considers the diffusive flow of chemical species inside a membrane that separates two tanks with unlimited supplies of the reactants A and B participating in the chemical reaction $2\,\mathrm{A} + \mathrm{B} \to (*)$. Classical modeling for this process results in a system of reaction-diffusion equations coupled through non-linear reaction terms. Despite its classical nature, the resulting model is mathematically intriguing as well as numerically challenging if one considers a particular reaction pathway comprising two reactions with widely varying rate coefficients [25]: molecules of A and B combine in a "fast" reaction to produce an intermediate C, and a "slow" reaction combines A and C to form the product $(*)$, which is not explicitly tracked in the model. This reaction pathway is expressed by

$$\mathrm{A} + \mathrm{B} \xrightarrow{\lambda} \mathrm{C}, \quad \mathrm{A} + \mathrm{C} \xrightarrow{\mu} (*),$$

in which the reaction coefficients λ and μ are scaled so that $\lambda \gg \mu = 1$.

Because the membrane is assumed to be thin compared to the directions normal to it, it is reasonable to use a one-dimensional spatial domain with variable x, scaled so that $x \in \Omega := (0, 1)$. In time, we compute from the initial time 0 to the final time t_{fin}, which is chosen such that the solution has reached its steady state. If we denote the concentrations of the chemical species A, B, C by functions $u(x,t)$, $v(x,t)$, $w(x,t)$, respectively, the reaction-diffusion system reads

$$\left.\begin{aligned} u_t &= u_{xx} - \lambda uv - uw, \\ v_t &= v_{xx} - \lambda uv, \\ w_t &= w_{xx} + \lambda uv - uw, \end{aligned}\right\} \quad \text{for } x \in (0,1) \text{ and } 0 < t \leq t_{\text{fin}}. \tag{4.1}$$

We assume that no molecules flow through the boundary, but the species A is supplied with a fixed concentration $\alpha > 0$ at $x = 0$ and species B with $\beta > 0$ at $x = 1$. This results in the mixed Dirichlet and Neumann boundary conditions

$$\begin{array}{llll} u = \alpha, & v_x = 0, & w_x = 0, & \text{at } x = 0, \\ u_x = 0, & v = \beta, & w_x = 0, & \text{at } x = 1. \end{array} \tag{4.2}$$

The problem statement of this initial-boundary value problem is completed by specifying the non-negative initial concentrations

$$u(x,0) = u_{\text{ini}}(x),\ v(x,0) = v_{\text{ini}}(x),\ w(x,0) = w_{\text{ini}}(x) \ \text{ for } x \in (0,1) \text{ at } t = 0, \tag{4.3}$$

with consistent boundary and initial data; i.e., $u_{\text{ini}}(0) = \alpha$ and $v_{\text{ini}}(1) = \beta$.

Because the first chemical reaction is much faster than the second one, rapid consumption of A and B to form C is expected at all spatial points x where A and B co-exist, leaving only one of them present with a positive concentration after an initial transient. Inside the regions dominated either by A or by B, the reaction rate of the fast reaction $q := \lambda uv$ will then become 0. However at the interfaces between the regions, where positive concentrations of A and B make contact due to diffusion, q will be non-zero; in fact, q will be large because $\lambda \gg 1$. For the corresponding stationary problem, analytical results in [14, 20] prove that there is one internal layer at a point $0 < x^* < 1$ of width $O(\varepsilon)$ and height $O(1/\varepsilon)$, where $\varepsilon = \lambda^{-1/3}$. Because initial conditions to the transient problem can have the internal layer at a different position than x^* or can have multiple internal layers, it is interesting to investigate the evolution of the internal layers and their coalescence to the single layer present at steady state. See [25] for studies of several representative initial conditions for this problem and [17] for studies on the asymptotic behavior of the transient problem.

To select an interesting transient behavior, we specify the initial condition

$$\begin{aligned} u_{\text{ini}}(x) &= \begin{cases} 4(0.25 - x)\,\alpha, & 0.00 \leq x \leq 0.25, \\ 0, & 0.25 < x < 0.50, \\ 64(0.50 - x)(x - 0.75)\,\gamma, & 0.50 \leq x \leq 0.75, \\ 0, & 0.75 < x \leq 1.00, \end{cases} \\ v_{\text{ini}}(x) &= \begin{cases} 0, & 0.00 \leq x < 0.25, \\ 64(0.25 - x)(x - 0.50)\,\delta, & 0.25 \leq x \leq 0.50, \\ 0, & 0.50 < x < 0.75, \\ 4(x - 0.75)\,\beta, & 0.75 \leq x \leq 1.00, \end{cases} \\ w_{\text{ini}}(x) &\equiv 0. \end{aligned} \tag{4.4}$$

The parameters α and β come from the boundary conditions (4.2), and their use in (4.4) guarantees that the initial conditions are consistent with the boundary conditions; therefore there are no boundary layers in the solutions, and we can focus our attention on the internal layers. The design in (4.4) produces linear functions in u and v near their respective Dirichlet boundary conditions and one quadratic hump for u and v each in the interior of the spatial domain, such that u and v are not non-zero simultaneously. For the parameters that affect the steady-state solution, we pick $\alpha = 1.6$, and $\beta = 0.8$. For the values γ and δ that control the height of the humps of u and v in (4.4), we choose $\gamma = \delta = 0.25$. For the final time, we select $t_{\text{fin}} = 20$; experiments show that this time is sufficient to reach the steady state solution using the criterion that the location x^* of the internal layer at steady state is approximated up to the resolution achievable by the spatial discretization. The reference value for x^* is taken from high-resolution simulations of the associated steady-state problem.

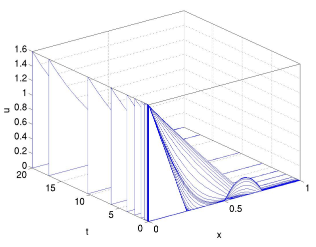
We introduce $u_j(t) \approx u(x_j, t)$, $v_j(t) \approx v(x_j, t)$, and $w_j(t) \approx w(x_j, t)$ at mesh points x_j, $j = 1, \ldots, N$. Spatial discretization by finite differences leads to an ODE system of the form $y' = f(t, y)$, where for computational efficiency, it is crucial to use interleaved ordering of the component functions as defined by $y(t) := [u_1, v_1, w_1, u_2, v_2, w_2, \ldots, u_N, v_N, w_N]^T$. An alternative approach for the spatial discretization is the finite element method, leading to $M \neq I$ in (1.1). We use finite differences here for the spatial discretization to allow for a direct comparison with the non-negativity preserving method implemented in Matlab's `ode15s` function that requires $M = I$.

We recall that the width of the internal layer at steady state is $\mathcal{O}(\varepsilon) = \mathcal{O}(\lambda^{-1/3})$. For $\lambda = 10^6$, $\varepsilon = 0.01$. Hence, to ensure a sufficient spatial resolution, we use $N = 513$ mesh points in $\Omega = (0, 1)$; this results in a mesh spacing of $\Delta x = 1/512$ and guarantees at least 5 mesh points within the length ε, which is the order of the width of the interface region. Calculations with both coarser and finer spatial meshes have shown the calculations to be reliable. The ODE tolerances used include a relative tolerance of 10^{-6} and absolute tolerance of 10^{-8}, which are tight enough to ensure a good initial guess for the Newton solver at every time step. The remaining method parameters for the Newton solver are chosen as for the Robertson problem in the previous section; in particular $\varepsilon^{(\text{neg})} = 10^{-12}$.

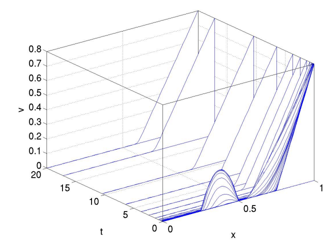
Figures 4.1(a), (b), and (c) show waterfall plots of the solution components $u(x, t)$, $v(x, t)$, and $w(x, t)$ vs. (x, t). Figure 4.1(d) shows the fast reaction rate q vs. (x, t). At $t = 0$, q is zero in most of Ω, where either u or v is zero, but it is large at the interfaces of the regions where u and v are non-zero. This reaction produces the intermediate reactant represented by w, which we see increasing over time in Figure 4.1(c), consuming the concentrations of u and v in the interior of Ω. After the initial transient, we see in Figures 4.1(a) and (b) that both u and v are non-zero only in one region each. This can also be seen in Figure 4.1(d), where for larger times only one spike exists in q instead of the three at $t = 0$. To analyze the interface evolution between the regions dominated by u or v, we track the locations of transitions of $u < v$ to $u > v$ and vice versa. Starting from the locations $x = 0.25, 0.50, 0.75$ at $t = 0$, these interfaces are tracked in Figure 4.1(e) up to time $t = 0.1$. We see that by this time, the three interfaces have coalesced to one. Beyond $t = 0.1$ (not shown), the interface moves slowly and smoothly to its steady state location at $x^* \approx 0.6$.

Figures 4.1(f)–(h) display several performance indicators of the ODE and non-linear solvers. We see that the time step Δt increases over time, reflecting the fact that the solution gets smoother over time and easier to approximate. We see the ODE method order k is fairly high, often 3 or 4. We see that also the non-linear solver behaves well with only 1 or 2 iterations needed at most time steps.

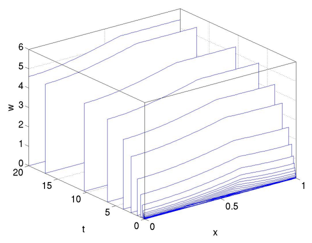
The interface problem is not as badly behaved as the Robertson problem in that small negative components do not lead to blow-up and can be controlled by tightening the ODE tolerance. This was done in [25] to establish our confidence in the solution obtained. However, the Robertson problem is a small ODE system; thus the



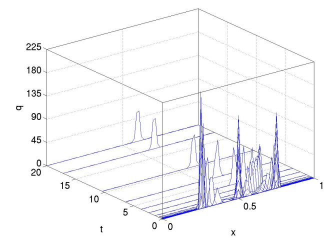
(a) u vs. (x,t)



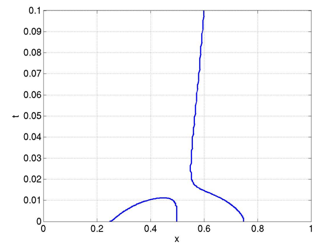
(b) v vs. (x,t)



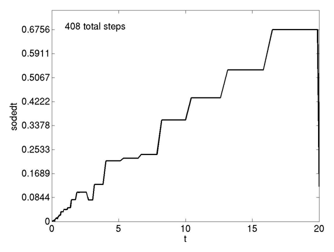
(c) w vs. (x,t)



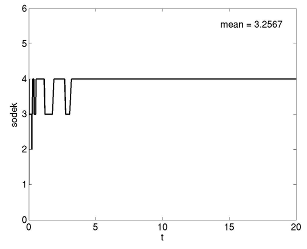
(d) $q = \lambda uv$ vs. (x,t)



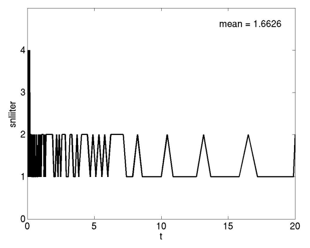
(e) interface vs. (x,t)



(f) Δt vs. t



(g) k vs. t



(h) #iter vs. t

FIG. 4.1. *Results for the interface problem. (a), (b), (c) Plots of the solution components u, v, w vs. (x,t), respectively, (d) Plot of the reaction rate $q = \lambda uv$ vs. (x,t), (e) Plot of the interface movement in the (x,t)-plane with zoom on the times $0 \leq t \leq 0.1$, (f) ODE time step Δt vs. t, (g) ODE method order k vs. t, (h) number of Newton iterations vs. t. (The results shown in this figure are computed by the NDFk method with non-negativity preservation using damping, with the predictor as initial guess, with Jacobian update forced whenever Δt or k change, and with norm-wise error control.)*

Table 4.1
Solution statistics for the interface problem with Jacobian update not *forced and* component-wise *error control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	1095	0	0	856	0
min(y)	–1.06e–08	0.0	0.0	–5.93e–09	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	770	775	775	775	795
`nfailed`	44	42	42	46	54
`nfevals`	1586	1592	1592	1599	1728
`npds`	28	30	30	30	36
`ndecomps`	143	142	142	145	156
`nsolves`	1585	1591	1591	1597	1727
`nclips`	N/A	1140	1140	358	27
mean(k)	4.69	4.66	4.66	4.65	4.55
mean(`iter`)	1.94	1.94	1.94	1.94	2.02
time (seconds)	11.61	10.718	10.75	11.13	11.66
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	1099	0	0	957	0
min(y)	–5.56e–07	0.0	0.0	–5.56e–07	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	896	882	883	896	880
`nfailed`	304	299	299	304	303
`nfevals`	3635	3598	3598	3636	3588
`npds`	287	284	284	287	287
`ndecomps`	427	424	424	427	429
`nsolves`	3634	3597	3597	3634	3587
`nclips`	N/A	1203	1204	198	7
mean(k)	4.21	4.25	4.25	4.21	4.21
mean(`iter`)	3.32	3.35	3.35	3.32	3.33
time (seconds)	23.76	23.48	24.69	25.48	24.43

computational cost to maintain non-negativity is not significant. The interface problem described in this section consists of $N = 513$ equations for each of the 3 species, leading to a system of 1,539 ODEs. Hence all algorithmic costs take on an increased importance, and we thus also report the wall clock time used to compute the solution in Tables 4.1–4.4. The other entries in the tables have the same meaning as before. Because the product of the chemical reactions is not tracked, mass is not conserved by this model, and we cannot readily compute a mass error.

We begin by noting that no simulations in Tables 4.1–4.4 blew up. However, in all tables and for both choices of $y_{n+1}^{(0)}$, the method without non-negativity enforcement as well as the constraint-following method suffer from negative intermediate Newton iterates, whereas clipping, DASSL clipping, and damping do not. This demonstrates that using $y_{n+1}^{(0)} = y_n$ does not guarantee non-negativity, as it happened to do for many cases in the Robertson problem.

As shown in Tables 4.1–4.4, the four methods studied exhibit approximately the same numerical efficiency. That is, none of the non-negativity preserving methods costs much additional effort. Comparing parts (a) and parts (b) in each of the four tables, we see that for this large ODE system, it is generally more efficient to use

Table 4.2
Solution statistics for the interface problem with Jacobian update not *forced and* norm-wise *error control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	598	0	0	450	0
min(y)	–7.71e–06	0.0	0.0	–3.93e–06	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	344	338	338	368	460
`nfailed`	33	32	32	39	84
`nfevals`	755	742	742	805	1046
`npds`	30	29	29	30	61
`ndecomps`	90	89	89	99	166
`nsolves`	754	741	741	803	1045
`nclips`	N/A	577	577	175	340
mean(k)	3.59	3.60	3.60	3.62	3.09
mean(`iter`)	1.94	1.94	1.94	1.92	1.86
time (seconds)	5.25	5.18	5.20	5.77	7.58
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	1008	0	0	792	0
min(y)	–3.93e–04	0.0	0.0	–4.01e–04	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	583	910	910	583	424
`nfailed`	167	182	182	167	158
`nfevals`	1732	2428	2425	1734	1286
`npds`	69	68	68	69	125
`ndecomps`	271	324	324	271	223
`nsolves`	1731	2427	2424	1732	1285
`nclips`	N/A	1665	1162	135	171
mean(k)	2.80	2.43	2.43	2.80	3.16
mean(`iter`)	2.24	2.18	2.18	2.24	2.24
time (seconds)	11.80	16.93	16.87	12.53	9.66

$y_{n+1}^{(0)} = p_n$. This is brought out by the number of ODE steps and the wall clock times as overall measures of efficiency, but also by the Newton solver taking fewer iterations to converge and the ODE method order being higher on average in almost all cases.

To analyze the effect of component-wise vs. norm-wise error control, we compare Table 4.1 against Table 4.2 and Table 4.3 against Table 4.4. Although the ODE method order is lower on average for norm-wise error control, it is clear from the overall measures of efficiency that the use of norm-wise error control is more efficient. This is consistent with the expectation that component-wise error control is more stringent because it controls the error in each component. However, the results for this system, such as min(y) and max(y), do not show any advantage of this added stringency. This suggests the use of norm-wise error control in the time stepping if the ODE components represent spatial approximations to PDE components.

To analyze the effect of whether to update the Jacobian only when required by the error control or also when Δt or k change, we compare Table 4.1 against Table 4.3 and Table 4.2 against Table 4.4. In most cases, the ODE method turns out to be roughly equally efficient with similar numbers of ODE steps and average ODE orders. However, the wall clock time indicates a distinct advantage of using better physical

TABLE 4.3
Solution statistics for the interface problem with Jacobian update forced and component-wise error *control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	745	0	0	566	0
min(y)	–3.46e–09	0.0	0.0	–4.17e–09	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	764	765	765	769	802
`nfailed`	16	17	17	16	36
`nfevals`	1133	1147	1147	1156	1295
`npds`	114	116	116	114	140
`ndecomps`	114	116	116	114	140
`nsolves`	1132	1146	1146	1154	1294
`nclips`	N/A	736	736	346	21
mean(k)	4.75	4.72	4.72	4.75	4.57
mean(`iter`)	1.44	1.45	1.45	1.47	1.51
time (seconds)	9.01	9.05	8.805	9.03	9.90
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	997	0	0	865	0
min(y)	–8.54e–11	0.0	0.0	–8.54e–11	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	867	867	867	867	871
`nfailed`	235	235	235	235	233
`nfevals`	3207	3207	3207	3208	3202
`npds`	352	352	352	352	352
`ndecomps`	352	352	352	352	352
`nsolves`	3206	3206	3206	3206	3201
`nclips`	N/A	1101	1101	198	9
mean(k)	4.20	4.20	4.20	4.20	4.22
mean(`iter`)	3.12	3.12	3.12	3.12	3.10
time (seconds)	21.98	21.80	22.33	22.53	22.13

information. This results from fewer failed steps and the shift of (relatively more expensive) *LU* decompositions to (relatively cheaper) Jacobian evaluations, along with fewer Newton iterates and thus fewer function evaluations and linear solves.

5. Conclusions.

The numerical tests in the previous sections, particularly for the interface problem, confirm the efficacy of supplying better physical information by re-evaluating the Jacobian more often and by using the predictor as initial guess for the Newton method by choosing $y_{n+1}^{(0)} = p_n$. Also, norm-wise error control turns out to be significantly more efficient, at the price of some accuracy only for the Robertson problem. We believe that these observations carry over to other large ODE systems, particularly those obtained by a method of line discretization of PDEs. These observations hold independent of any non-negativity preserving strategies.

The tests demonstrate that all non-negativity preserving strategies considered do not significantly degrade ODE method order, ODE step size changing history, or other efficiency measures, compared to the original method without non-negativity preservation. Only the non-negativity preservation methods of clipping, DASSL clipping, and damping are effective for all intermediate Newton iterates, as seen for both test problems and for all choices of initial guess for the Newton method, type of error

TABLE 4.4
Solution statistics for the interface problem with Jacobian update forced and norm-wise *error control.*

(a) Initial guess for Newton iteration: predictor p_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	482	0	0	336	0
min(y)	–2.88e–06	0.0	0.0	–2.88e–06	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	345	344	344	365	397
`nfailed`	13	12	12	19	50
`nfevals`	617	605	605	650	780
`npds`	69	69	69	78	119
`ndecomps`	69	69	69	78	119
`nsolves`	616	604	604	648	779
`nclips`	N/A	471	471	156	236
mean(k)	3.67	3.51	3.51	3.66	3.23
mean(`iter`)	1.69	1.68	1.67	1.66	1.68
time (seconds)	4.77	4.74	4.78	5.24	6.52
(b) Initial guess for Newton iteration: old solution y_n					
	no enforcement	clipping	dassl	constraint	damping
`nnegative`	666	0	0	485	0
min(y)	–6.51e–05	0.0	0.0	–6.51e–05	0.0
max(y)	5.4211	5.4211	5.4211	5.4211	5.4211
`nsteps`	470	470	470	470	424
`nfailed`	82	83	83	82	113
`nfevals`	1260	1264	1261	1261	1155
`npds`	151	152	152	151	177
`ndecomps`	151	152	152	151	177
`nsolves`	1259	1263	1260	1259	1154
`nclips`	N/A	660	657	91	123
mean(k)	3.14	3.14	3.12	3.14	3.14
mean(`iter`)	2.16	2.16	2.16	2.16	2.16
time (seconds)	9.23	9.21	9.20	9.45	9.29

control, or Jacobian evaluation strategy. Based on the results for the interface problem alone, we could not recommend one of these methods over the others; in fact, in some cases, clipping and DASSL clipping turn out to be slightly more efficient than the dampening method. This is why the Robertson problem is insightful: it demonstrates that non-negativity preservation by damping the Newton method has the potential for better mass conservation ability of the ODE method, which is an important issue in the solution of many problems such as problems involving reactions.

The two numerical examples together thus illustrate the key advantages of our proposed approach: (i) The non-negativity of all intermediate Newton iterates is ensured by construction, so no coefficient function of the ODE can be called with arguments for which it is not valid. (ii) The method avoids significant degradation of mass conservation. (iii) The method incurs only minor additional computational cost when it is active, i.e., when potentially negative solution components are encountered, and no additional cost when it is not active. (iv) Because our method is implemented within the framework of the Newton method inside an implicit time-stepping algorithm, it applies to families of high-order implicit time-stepping methods without any change to their automatic time step size and method order control. (v) The method applies to general IVPs (1.1) with mass matrix, such as resulting from method of lines discretizations using the finite element method.

REFERENCES

[1] U. M. Ascher, *Numerical Methods for Evolutionary Differential Equations*, SIAM, 2008.

[2] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, vol. 14 of Classics in Applied Mathematics, SIAM, 1996.

[3] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, *VODE: A variable-coefficient ODE solver*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1038–1051.

[4] P. N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 450–481.

[5] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[6] J. E. Dennis, J. J. E. Dennis, and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, 1996.

[7] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Sci. Comput., 17 (1996), pp. 16–32.

[8] M. K. Gobbert, *Long-time simulations on high resolution meshes to model calcium waves in a heart cell*, SIAM J. Sci. Comput., 30 (2008), pp. 2922–2947.

[9] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, vol. 14 of Springer Series in Computational Mathematics, Springer-Verlag, 1991.

[10] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers*, ACM Trans. Math. Software, 31 (2005), pp. 363–396.

[11] A. C. Hindmarsh and G. D. Byrne, *Applications of EPISODE: An experimental package for the integration of systems of ordinary differential equations*, in Numerical Methods for Differential Systems, L. Lapidus and W. E. Schiesser, eds., Academic Press, Inc., New York, 1976, pp. 147–166.

[12] A. C. Hindmarsh and R. Serban, *User documentation for CVODE v2.5.0*, tech. rep., Lawrence Livermore National Laboratory, 2006. URL `www.llnl.gov/casc/sundials`.

[13] 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.

[14] L. V. Kalachev and T. I. Seidman, *Singular perturbation analysis of a stationary diffusion/reaction system whose solution exhibits a corner-type behavior in the interior of the domain*, J. Math. Anal. Appl., 288 (2003), pp. 722–743.

[15] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, vol. 16 of Frontiers in Applied Mathematics, SIAM, 1995.

[16] *MATLAB Release R2006b (August 03, 2006)*. The MathWorks, Inc., `www.mathworks.com`.

[17] M. Muscedere and M. K. Gobbert, *Parameter study of a reaction-diffusion system near the reactant coefficient asymptotic limit*, Dynamics of Continuous, Discrete and Impulsive Systems Series A Supplement, (2009), pp. 29–36.

[18] H. H. Robertson, *The solution of a set of reaction rate equations*, in Numerical Analysis: An Introduction, J. Walsh, ed., Academic Press, 1966, pp. 178–182.

[19] A. Sandu, *Positive numerical integration methods for chemical kinetic systems*, J. Comput. Phys., 170 (2001), pp. 589–602.

[20] T. I. Seidman and L. V. Kalachev, *A one-dimensional reaction/diffusion system with a fast reaction*, J. Math. Anal. Appl., 209 (1997), pp. 392–414.

[21] L. F. Shampine, *Conservation laws and the numerical solution of ODEs*, Comput. Math. Appl. Part B, 12 (1986), pp. 1287–1296.

[22] ———, *Linear conservation laws for ODEs*, Comput. Math. Appl., 35 (1998), pp. 45–53.

[23] L. F. Shampine and M. W. Reichelt, *The MATLAB ODE suite*, SIAM J. Sci. Comput., 18 (1997), pp. 1–22.

[24] 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.

[25] A. M. Soane, M. K. Gobbert, and T. I. Seidman, *Numerical exploration of a system of reaction-diffusion equations with internal and transient layers*, Nonlinear Anal.: Real World Appl., 6 (2005), pp. 914–934.