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 time 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 true solution, the non-negativity of its approximate solution is often destroyed by the discretization of the mathematical model. Furthermore, in the case of implicit time discretizations of ordinary differential equations (ODEs), even if the discrete non-linear algebraic equations at each time step are guaranteed to admit non-negative exact solutions, non-negativity of the approximate solutions is often destroyed by the truncated Newton method used to solve them. We propose a modification that guarantees the non-negativity of the 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 a mass matrix. Thus the algorithm can be directly applied to any initial-value problem for ODEs in standard form as well as to method-of-lines discretizations of partial differential equations. The effectiveness of the method is demonstrated by 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 a mathematical model can guarantee that its solution is non-negative, the non-negativity of its approximate solution is often destroyed by the discretization of the mathematical model, e.g., by the spatial discretization of a partial differential equation (PDE) in a method-of-lines approach or by the time discretization of an ordinary differential equation (ODE) with a (stable) time step that exceeds the radius of monotonicity; see, e.g., [15, 26] 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 truncation of the Newton iteration after finitely many steps may destroy the non-negativity property of the ODE solution, even if the ODE discretization has a non-negative exact solution [21, page 592].

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

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

*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.

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 spatial discretization in the method-of-lines approach to discretize a PDE. As appropriate for that case, we assume at present that M is non-singular and constant. We assume that the given problem (1.1) is a stiff ODE system, as induced by the method-of-lines discretization of a PDE, for instance. For reliability and efficiency of the time-stepping, it is also vital that the ODE solver include a sophisticated error estimation mechanism and automatic step size and method order control. Well-known and recommended codes for this problem include VODE [5], CVODE in SUNDIALS [12], and DASSL [4], which implement the family of backward differentiation formulas (BDFk), as well as RADAU5 [11], which is an implicit Runge-Kutta code based on 3-stage Radau collocation. We use the `ode15s` function in Matlab [18, 25] as the starting point for our development because it implements the attractive family of numerical differentiation formulas (NDFk) that is a generalization of the BDFk family and has potential for significantly more efficient time-stepping [10, 25]. The paper by Shampine and Reichelt [25] documents the methods in detail, and we follow their notation below as well as use Matlab's `ode15s` as the basis for our test implementation.

In many practically important problems, the solution components represent quantities such as chemical concentrations that should be non-negative for physical reasons. For large classes of problems such as reaction-diffusion equations, it has been established analytically that their mathematical models in the form of ODEs maintain this non-negativity as well as ODEs resulting from appropriate spatial discretization in the case of PDEs [15, Chapter 1]. Thus, we consider problems of the form (1.1) that have a non-negative solution for every vector of initial conditions y_{ini} with non-negative components. However, using a truncated Newton method inside an implicit ODE solver often destroys the non-negativity of the ODE solution, even if the ODE discretization admits a non-negative solution; see, e.g., [21, page 592]. This can happen for both typical choices of initial guesses for the Newton method ([15, page 127]), namely the numerical solution at the previous time step or a predictor involving the previous solution and approximations to one or more of its derivatives. These observations suggest that one should suitably modify the way in which each iteration of the Newton solver and its initial guess are constructed.

One approach for problems, for which negative values do not cause catastrophic failure, is to control the amount of non-negativity in the ODE solution by tightening the error tolerances imposed on the ODE solver, as suggested for example in the User Documentation for CVODE [14, Subsection 5.5.2]. We used this approach in [27] to confirm that the results for the interface problem were reliable despite the small negative values in some solution components. However, to eliminate negativity entirely with this approach may incur a dramatic increase in computational cost because the tighter tolerances force smaller time steps.

One of the first ways 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 severe disadvantage that it destroys mass conservation: mass is effectively added to the system whenever a negative concentration is increased to zero [21]. It also interferes with a solver's time stepping and error estimation algorithms. Another approach, suggested in [21], is to project the solution into its non-negative domain via a constraint optimization. Drawbacks of this approach include the complication of its implementation, the computational cost associated with solving

an optimization problem inside of an ODE solver and, again, the possible failure of mass conservation.

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 very likely yield negative values again. Partially motivated by this observation, a strategy described as *constraint-following* is proposed in [26] 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 non-linear Newton solver. This makes the method efficient in that it incurs cost only if negative components are encountered. However, because it works as part of the ODE error control mechanism, it does not apply to intermediate iterates of the non-linear solver; i.e., these iterates can still have negative components. These intermediate iterates appear in turn in the evaluation of the ODE function $f(t, y)$ in each Newton step, which can thus become undefined, if it involves square root or similar functions of the solution components.

To ensure the non-negativity of all intermediate Newton iterations, our algorithm limits the length of each Newton step, borrowing an idea from line searches in optimization algorithms and from *damping* in a Newton method [3, page 111]. If any solution component becomes negative with the default damping factor of 1 in a Newton iteration, then the step is recomputed with a smaller damping parameter. In order to ensure progress of the iteration, the damping factor must not vanish. We accomplish this by limiting the Newton step such that the solution components do not become more negative than $-\varepsilon^{(\mathrm{neg})}$ and then setting the remaining negative components with magnitude less than $\varepsilon^{(\mathrm{neg})}$ to zero. Here, $0 < \varepsilon^{(\mathrm{neg})} \ll 1$ is a user-supplied parameter that explicitly indicates the acceptable amount of mass error and implicitly ensures that the damping parameter in the Newton step does not become zero. On the one hand, our approach agrees with a technique described in [12, pages 373–74] for the non-linear solver KINSOL in SUNDIALS that uses the idea to enforce constraints on the solution components, but its use inside an ODE solver is apparently new. On the other hand, the philosophy behind the user-supplied parameter $\varepsilon^{(\mathrm{neg})}$ is analogous to the strategy used by step size control algorithms in many ODE solvers: To accept the ODE solution at a new time step, the error estimator must be within a user-supplied tolerance. This can be achieved in principle by making the time step size as small as necessary. But allowing it to become too small might unacceptably increase the number of time steps to reach the desired final time. Thus, if the time step required by the error controller is smaller than a specified minimum step size, the code stops with an error. It is then left up to the user to find the right balance between desired error tolerances and minimum acceptable time step size for the particular ODE. In the same way, the user can vary $\varepsilon^{(\mathrm{neg})}$ appropriately for the problem under consideration and balance it to the ODE tolerance, from which the Newton tolerance is derived.

Theoretical analyses of convergence for Newton, truncated Newton, and inexact Newton methods are available, e.g., [6, 7, 8, 9, 17]. These generally require that the initial guess be 'close enough' to the solution. In principle, this is also true of our proposed method — since 'close enough' to a strictly positive solution will ensure that no

damping occurs — but is not the most relevant in the present context: The situation inside an ODE solver is a simpler special case, because the automatic time step size controller can decrease the time step in response to convergence problems of the Newton method. This is effectively the same strategy used by every implementation that does not stop to verify at each step the explicit quantitative convergence condition by Kantorovich. We also 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 linear multistep methods conserve mass (and other linear invariants) [23, 24], and changing the length of the Newton step does not destroy this property. The convergence of the Newton iteration is thus no more an issue for this modification than for the usual version, since we are not changing the Newton convergence tests, the error control, or the step-size or method order selection. As in the original version without damping, the user would balance tolerance and time step as needed to ensure convergence of the Newton method and the ODE solver, unless the computational effort resulting from small time steps becomes unacceptable. Practice has not indicated this occurring any more frequently with the proposed method.

To study the effectiveness and efficiency of our method in practice, we present extensive test calculations for two examples. The first is the well-known *Robertson problem* [20], which is a stiff system of three reaction ODEs. The widely varying scales of the reaction coefficient are the source of the stiffness and justify the use of implicit ODE solvers for this problem. 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 [11, pages 3 and 157], [1, page 61] and in the documentation for ODE packages such as EPISODE [13], VODE [5], CVODE in SUNDIALS [12], and Matlab's `ode15s` function [18, 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 [11, 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 with each component representing the time-dependent approximation to the solution at each spatial point. This system of ODEs is non-linear due to the non-linearities in the reaction terms. Similar to the Robertson problem, the solution has mostly smooth behavior, but it also has localized sharp transients [19, 27], so the use of a sophisticated error control and an automatic step size selection algorithm is vital for a reliable and efficient solution of the problem.

The two examples share the feature that their solutions represent chemical concentrations and hence should be non-negative for physical reasons. Moreover, for both problems (including the spatial discretization of the interface problem), it has been established theoretically that they maintain this non-negativity [15, Chapter 1]. The two examples considered here are chosen so that all available methods for non-negativity preservation can be used without break-down and can thus be compared 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, catastrophic failure in the form of blow-up of the solution can still and does occur for the Robertson problem. Also, even though our method applies to (1.1) with a mass matrix, we restrict ourselves to the case of $M = I$ so that Matlab's non-negativity preservation technique can also be applied. Besides comparing the three aforementioned methods for preserving non-negativity, we also analyze the impact of several other method parameters on the effectiveness and efficiency of the ODE solver.

Both examples are needed to analyze certain features of the methods under consideration: The Robertson problem conserves mass, hence it is useful to study the effect of the different non-negativity preservation methods on this property. But the Robertson problem is too small to observe meaningful computation times, hence it is interesting to consider the interface problem, which as PDE system requires significantly more computational effort. The test calculations for two examples together show the key advantages of our approach: (i) The non-negativity of all intermediate Newton iterates is ensured by construction, so no coefficient function of the ODE will ever 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 ODE methods without any change to their automatic time step size and method order control. (v) The method applies to general initial-value problems (IVPs) involving a mass matrix, such as arising from method-of-lines discretizations of evolutionary PDEs using the finite element method; For such problems, high-order implicit time-stepping methods can be particularly advantageous or even necessary to reach relevant final simulation times [10].

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 in 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 vary and 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.

2. Time Discretization and Non-Negativity Preservation.

2.1. Numerical Differentiation Formulas. We review the NDFk following the paper [25] but generalized to the ODE with mass matrix in (1.1). That is, the following formulas detail the code implemented in the Matlab function `ode15s` using a notation that is slightly modified from [25]. Throughout the development, we leave other parts of the code unchanged, in particular the sophisticated error control and automatic time step size and method order selection algorithms.

The NDFk generalize the well-known backward differentiation formulas of order k (BDFk) (see, e.g., [2]). The BDFk methods approximate the derivative in the ODE (1.1), written in the form $My'(t_{n+1}) - f(t_{n+1}, y_{n+1}) = 0$, by backward difference approximations of order k. In pseudo-constant time step notation with time step Δt, this approximation reads $y'(t_{n+1}) \approx (1/\Delta t) \sum_{m=1}^{k} (1/m)\, \nabla^m y_{n+1}$, with the notation for the backward differences defined by $\nabla^m y_\ell := \nabla^{m-1} y_\ell - \nabla^{m-1} y_{\ell-1}$ for $m \geq 1$ and $\nabla^0 y_\ell := y_\ell$. The NDF$k$ are then defined by adding the term $-\alpha_k \gamma_k M(y_{n+1} - p_n)$ 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 [25]. The parameter α_k can now be chosen for each method order $1\le k\le 5$ to make the method more efficient, and [25] explains the values that appear in `ode15s`. The notation for the predictor (2.2) in [25] is $y_{n+1}^{(0)}$ because its use as initial guess for the Newton method is hard-wired in the code; we introduce a separate notation for the predictor here so that we can also choose another initial guess for the Newton method later with a clear notation.

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: Start with 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 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, $d^{(0)} = y_{n+1}^{(0)} - p_n = 0$ always. This is how the algorithm is hard-coded in `ode15s` and explains why [25] does not introduce a separate notation for p_n. To allow us to study the effect of both initial guesses in our numerical studies, we write the algorithm in the following form: Start with $y_{n+1}^{(0)}$ 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 and J of the ODE problem (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 $M^{(\text{iter})} := M - \frac{\Delta t}{(1-\alpha_k)\gamma_k} J$ and then solving the linear system using the factorized form. The efficiency of this approach lies in pre-computing the decomposition and re-using it for several Newton iterations and in fact for several (potentially many) ODE 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 (which changes 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. In our problem, where the evaluation of the Jacobian is very cheap, 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 New Algorithm for Preserving Non-Negativity. 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 return to the form of the Newton update in (2.3) and introduce a damping parameter $0 < s_i \leq 1$ in

the computation of the update $y_{n+1}^{(i+1)} = y_{n+1}^{(i)} + s_i\,\Delta^{(i)}$ that guarantees that no component of the new Newton iterate $y_{n+1}^{(i+1)}$ is smaller than $-\varepsilon^{(\mathrm{neg})}$, where $0 < \varepsilon^{(\mathrm{neg})} \ll 1$ is a user-supplied parameter. The choice of a positive parameter $\varepsilon^{(\mathrm{neg})}$ in this step ensures that the damping parameter s_i never vanishes, that is, 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: Start with $y_{n+1}^{(0)}$ either as p_n or y_n, compute $d^{(0)} = y_{n+1}^{(0)} - p_n$, then iterate for $i = 0, 1, 2, \ldots$

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

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 $\varepsilon^{(\mathrm{neg})}$. 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. But the predictor p_n from (2.2) can have negative components, and thus also the initial guess $y_{n+1}^{(0)} = p_n$ of the Newton iteration. If that is the case, 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 without introducing significant mass error because (contrary to clipping) the components set to zero in the second step are no larger than $\varepsilon^{(\mathrm{neg})}$. This proposed algorithm ensures that $f(t, y)$ and $J(t, y)$ are never evaluated with y having negative components (contrary to merely choosing tighter ODE tolerances or constraint-following in `ode15s`). Although somewhat more computationally expensive than others, this approach maintains the philosophy of [26] regarding efficiency in that it does not cost anything if there are no negative components, and it is reasonably inexpensive if there are. Moreover, this approach is easily implemented in the framework of the Newton method inside an implicit ODE solver of any method order k. We contrast this in complexity of programming as well as computational cost to the approach suggested in [21] based on projecting solutions with negative components into the positive cone via a constrained optimization method.

The Newton iterate $y_{n+1}^{(i+1)}$ is accepted as converged when the norm of the update vector $\Delta^{(i)}$ (without the damping factor s_i) is less than its tolerance; this ensures that the damped Newton iterations compute a solution of the same quality as the undamped iterations. The control strategy proceeds then as for the original code: If a converged solution cannot be found within the maximum number of Newton iterations allowed, the Jacobian is re-evaluated and/or the time step is reduced. The quantity $d^{(i+1)} = d^{(i)} + s_i\,\Delta^{(i)}$ computes the ODE error estimator associated with the new solution $y_{n+1}^{(i+1)}$ following analogous formulas as in the derivation of (2.5).

Provided the ODE problem (1.1) of the type under consideration has a non-negative solution, it can be shown that the implicit Euler method (i.e., BDF1) admits a non-negative solution [15]. This is neither true for BDFk methods with $k > 1$ nor for NDFk methods with any k, and these methods can admit solutions with negative components. However, these solutions are unphysical, and tighter tolerances on the ODE error estimator, i.e., sufficiently small time steps, can be used to control their magnitude [14, Subsection 5.5.2]. For sufficiently small time steps, a converged ODE solution will thus eventually be non-negative for the problems under consideration. Under these conditions, our approach that controls non-negativity by construction will converge to the same solution, albeit with potentially larger time steps.

3. The Robertson Problem.

3.1. Problem Statement. The Robertson problem [11, 20] describes chemical reactions among three reactants A, B, and C as

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

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

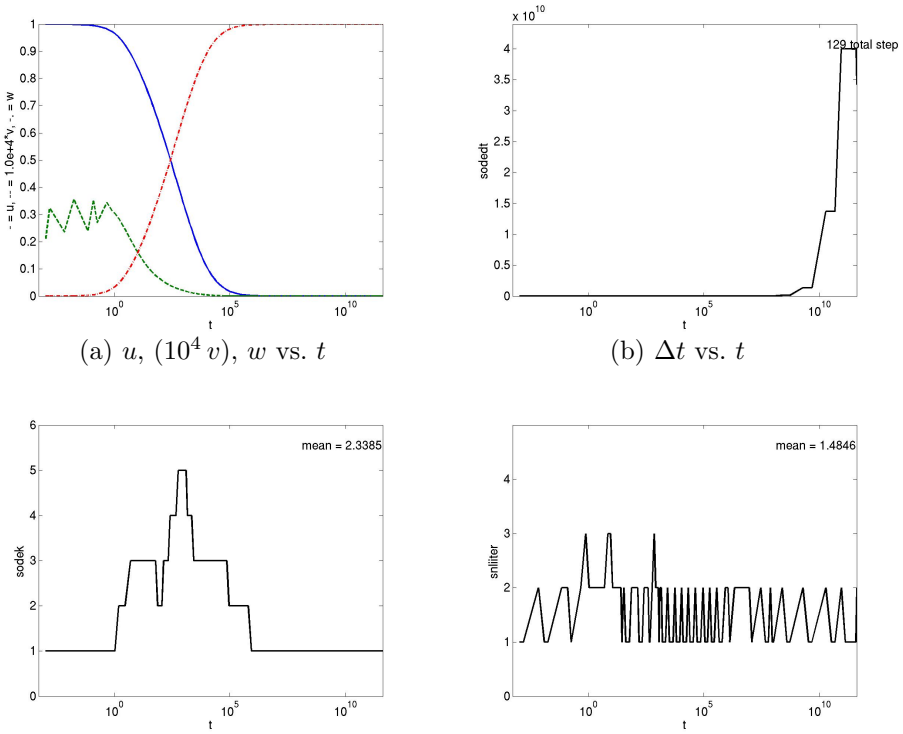
$$\begin{aligned} u_t &= -0.04\,u + 10^4\,v\,w, & u(0) &= 1, \\ v_t &= 0.04\,u - 10^4\,v\,w - 3\cdot 10^7\,v^2, & v(0) &= 0, \\ w_t &= + 3\cdot 10^7\,v^2, & w(0) &= 0. \end{aligned} \tag{3.1}$$

The stiffness of the ODE system results from the widely varying rates of the reactions, as indicated by the reaction rate coefficients ranging from 0.04 to $3 \cdot 10^7$. We choose to consider the final time $t_{\mathrm{fin}} = 4 \cdot 10^{11}$, which goes beyond the time interval over which many ODE solvers are stable [11, 18]. To phrase the problem in the standard form $y' = f(t, y)$, we collect the concentrations as solution components of the vector $y(t) = [u(t), v(t), w(t)]^T$.

Figure 3.1 (a) plots the three solution components as functions of time, using a logarithmic time scale to accommodate the large final time. Notice that the second solution component is very small and thus is scaled by a factor 10^4 to make it visible on the same scale as the other solution components, as done in, e.g., [18, function `hb1ode`]. Starting from an initial condition with species A being the only one present (solid line for u), the concentrations tend to a steady state containing species C only (dash-dotted line for w), with the intermediate reactant B (dashed line for $10^4 v$) being significant only temporarily.

Figures 3.1 (b)–(d) show plots of some crucial performance indicators of the numerical method. Figure 3.1 (b) indicates that the time steps Δt selected by the automatic step size control increase exponentially up to the maximum allowed value of $\Delta t_{\max} = t_{\mathrm{fin}}/10$; notice the vertical scale in the plot. 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.

3.2. Numerical Studies. 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 through 3.4



(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 (notice the scaling in v), (b) ODE time step Δt vs. t, (c) ODE method order k vs. t, (d) number of Newton iterations vs. t. Notice that the time axis has logarithmic scale. (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.)*

list indicators that quantify the effectiveness and efficiency of the different method and parameter choices. In all cases, we use the NDFk method with $1 \le k \le 5$, a relative tolerance of 10^{-3} and an absolute tolerance of 10^{-6}, following the defaults for `ode15s` in Matlab. The initial time step is chosen as $\Delta t_{\text{ini}} = 5.48 \cdot 10^{-4}$ and the maximum as $\Delta t_{\max} = t_{\text{fin}}/10 = 4 \cdot 10^{10}$, consistent with Matlab's default behavior for this problem. Furthermore, the non-linear Newton solver uses an analytically supplied Jacobian matrix. The tolerance for accepting a Newton solution is $100\, \varepsilon_{\text{mach}} \approx 2.22 \cdot 10^{-14}$, and the maximum number of Newton iterations is 4, again following the choices implemented in `ode15s`. All cases of our method of damping each Newton iterate use the value $\varepsilon^{(\text{neg})} = 10^{-12}$; 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 ODE solver with sophisticated error control using 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 (larger numbers of ODE steps, etc.), 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 will always become negative eventually, followed shortly by blow-up of

the solution. This behavior is not eliminated, but only delayed, by the use of tighter tolerances. Thus, we use the default choices from `ode15s` in these studies, and choose a final time suggested by [11, page 157], for which negativity and blow-up occur and thus the use of a non-negativity preservation algorithm is necessary.

Each table collects results for the NDFk method (i) with "no enforcement" of the non-negativity of the solution and with non-negativity enforced by the methods defined previously, namely (ii) by "clipping," i.e., setting any negative component in every Newton iterate to zero, (iii) by following the "constraint" as in Matlab using the `NonNegative` option for all solution components, and (iv) by "damping" the Newton iterates as described in section 2.3.

The first four quantities listed for each method quantify the effectiveness of the method, that is, 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 over all times; these are reported in the first two rows as `nnegative` and $\min(y)$. Notice that it is possible for Matlab's non-negativity preserving method to have negative values here, since we are tracking the negativity of intermediate Newton iterations. Next, we report the maximum over all components of the vector $y(t)$ over all times; this value should be no larger than 1. Moreover, the Robertson problem conserves mass, and the total mass $m(t) := u(t)+v(t)+w(t)$ should satisfy $m(t) = 1$ at all times; 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 cannot be trusted, and its results are reported in the tables only for completeness. The remaining results for each method quantify the efficiency of the method, that is, its computational cost. 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 clipping or in constraint-following, while in damping, `nclips` counts the number of times that a Newton iteration is damped. Finally, the entries for mean(k) and mean(`iter`) 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`, namely with component-wise error control (`NormControl` switched off) and with the Newton method using the predictor p_n as initial guess. This means that in the cases of no enforcement and of constraint-following our code gives identical results 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. Since the solution blows up, no other results (e.g., mass error) can be trusted for this case.

TABLE 3.1
Solution statistics for the Robertson problem with Jacobian update not *forced whenever* Δt *or* k *change and* component-wise *error control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	696	0	4	0
$\min(y)$	–1.49e+08	0.0	–1.54e+02	0.0
$\max(y)$	1.49e+08	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	6.31e+00	1.40e–05	1.03e–14	8.77e–15
`nsteps`	462	241	237	238
`nfailed`	177	20	18	18
`nfevals`	1155	479	464	463
`npds`	74	13	13	13
`ndecomps`	279	71	67	68
`nsolves`	1154	478	462	462
`nclips`	N/A	44	0	17
mean(k)	2.33	2.81	2.84	2.84
mean(`iter`)	1.68	1.81	1.79	1.79
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	0	0	0	0
$\min(y)$	0.0	0.0	0.0	0.0
$\max(y)$	1.0	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	5.61e–14	5.61e–14	5.61e–14	5.61e–14
`nsteps`	218	218	218	218
`nfailed`	17	17	17	17
`nfevals`	568	568	569	568
`npds`	14	14	14	14
`ndecomps`	63	63	63	63
`nsolves`	567	567	567	567
`nclips`	N/A	0	0	0
mean(k)	2.74	2.74	2.74	2.74
mean(`iter`)	2.42	2.42	2.42	2.42

In particular, reaching the final time is purely formal here, as the solution has long before stopped converging. 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, moreover, the number of ODE steps and all other performance statistics deteriorate rapidly for tighter tolerances. Accordingly, our proposed algorithm aims to improve the physical correctness of the solution (and avoid blow-up) for any final time without paying a 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 following methods aim to prevent this kind of unphysical violation of mass conservation. Indeed, 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 significant negative values in the intermediate Newton iterations. This would be fatal in cases where the ODE function $f(t, y)$ or its Jacobian become undefined for negative values.

TABLE 3.2
Solution statistics for the Robertson problem with Jacobian update not *forced whenever* Δt *or* k *change and* norm-wise *error control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	530	0	42	0
$\min(y)$	–1.92e+08	0.0	–1.39e+03	0.0
$\max(y)$	1.92e+08	1.0148	1.0023	1.0
$\max \lvert m(t) - 1 \rvert$	8.39e+00	1.48e–02	2.29e–03	6.67e–09
`nsteps`	298	147	175	140
`nfailed`	138	25	49	13
`nfevals`	752	297	376	278
`npds`	70	14	26	12
`ndecomps`	211	61	92	46
`nsolves`	751	296	374	277
`nclips`	N/A	89	0	18
mean(k)	2.04	1.91	1.98	2.32
mean(`iter`)	1.57	1.66	1.57	1.79
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	2	0	1	0
$\min(y)$	–1.87e+00	0.0	–3.67e–03	0.0
$\max(y)$	1.0	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	1.44e–11	1.44e–11	1.44e–11	1.44e–11
`nsteps`	127	127	127	127
`nfailed`	10	10	10	10
`nfevals`	300	300	301	301
`npds`	11	11	11	11
`ndecomps`	40	40	40	40
`nsolves`	299	299	299	300
`nclips`	N/A	1	0	2
mean(k)	2.30	2.30	2.30	2.30
mean(`iter`)	2.19	2.19	2.19	2.19

Table 3.1 (b) shows the results with the Newton method using y_n as initial guess instead of the predictor p_n. We notice that `ode15s` without non-negativity enforcement no longer blows up. In fact, for the small ODE system of the Robertson problem, 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(`iter`) as a result of less information about the solution contained in its initial guess. There can however not be any guarantee that using y_n as initial guess for the Newton method improves the efficiency of the ODE

TABLE 3.3
Solution statistics for the Robertson problem with Jacobian update forced whenever Δt or k change and component-wise error *control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	619	0	5	0
$\min(y)$	–1.71e+08	0.0	–2.84e–03	0.0
$\max(y)$	1.71e+08	1.0	1.0	1.0
$\max\|m(t)-1\|$	7.34e+00	5.24e–08	8.88e–15	8.66e–15
`nsteps`	465	226	232	226
`nfailed`	121	2	6	2
`nfevals`	909	298	311	296
`npds`	226	51	57	51
`ndecomps`	226	51	57	51
`nsolves`	908	297	309	295
`nclips`	N/A	22	0	8
mean(k)	2.22	2.64	2.60	2.64
mean(`iter`)	1.42	1.30	1.28	1.29
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	0	0	0	0
$\min(y)$	0.0	0.0	0.0	0.0
$\max(y)$	1.0	1.0	1.0	1.0
$\max\|m(t)-1\|$	5.65e–14	5.65e–14	5.65e–14	5.65e–14
`nsteps`	222	222	222	222
`nfailed`	1	1	1	1
`nfevals`	421	421	422	421
`npds`	49	49	49	49
`ndecomps`	49	49	49	49
`nsolves`	420	420	420	420
`nclips`	N/A	0	0	0
mean(k)	2.68	2.68	2.68	2.68
mean(`iter`)	1.87	1.87	1.87	1.87

method which will be shown in the next section.

Table 3.2 shows the results of the methods when norm-wise error control is used, corresponding in Matlab to switching `NormControl` on. We first notice in Table 3.2 (a) that the method without non-negativity enforcement still blows up. The other methods with enforcement of non-negativity of the solution prevent blow-up, but all of them introduce error in the total mass that is not negligible; in some cases, $\max(y)$ is even above its theoretical maximum of 1. Moreover, the constraint-following method suffers again from negative intermediate Newton iterates. The 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 significant, negative intermediate results still appear in the cases without non-negativity preservation and with constraint-following. Overall, comparing Tables 3.1 and 3.2, we notice that norm-wise error control leads to somewhat less accurate physical results for the Robertson problem, but also to significantly better efficiency.

The previous Tables 3.1 and 3.2 reported 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

TABLE 3.4
Solution statistics for the Robertson problem with Jacobian update forced whenever Δt or k change and norm-wise *error control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	30	0	19	0
$\min(y)$	–1.68e–03	0.0	–6.63e+00	0.0
$\max(y)$	1.0	1.0022	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	1.51e–14	2.23e–03	4.88e–15	6.00e–15
`nsteps`	139	143	155	129
`nfailed`	8	9	18	4
`nfevals`	203	212	245	201
`npds`	45	47	61	35
`ndecomps`	45	47	61	35
`nsolves`	202	211	243	200
`nclips`	N/A	52	0	15
mean(k)	1.73	1.74	1.65	2.34
mean(`iter`)	1.34	1.35	1.33	1.48
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	0	0	0	0
$\min(y)$	0.0	0.0	0.0	0.0
$\max(y)$	1.0	1.0	1.0	1.0
$\max \lvert m(t) - 1 \rvert$	2.16e–13	2.16e–13	2.16e–13	2.16e–13
`nsteps`	127	127	127	127
`nfailed`	0	0	0	0
`nfevals`	224	224	225	224
`npds`	30	30	30	30
`ndecomps`	30	30	30	30
`nsolves`	223	223	223	223
`nclips`	N/A	0	0	0
mean(k)	2.27	2.27	2.27	2.27
mean(`iter`)	1.75	1.75	1.75	1.75

and only re-evaluated when deemed necessary by the error control mechanism. This reflects the common situation that the evaluation of the Jacobian 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 the Jacobian J is updated or when the time step Δt or the ODE method order k are changed by the error control algorithm. This means that, in situations where Δt or k change and an LU decomposition of $M^{(\text{iter})}$ is necessary, this decomposition may not take the latest Jacobian into account. This means that the error control algorithm is optimized to minimize the number of Jacobian evaluations at the potential cost of more decompositions.

However, for this problem and with an analytically supplied Jacobian, the evaluation of J incurs negligible cost, and we modify `ode15s` to force an update of the Jacobian whenever Δt or k change so that the iteration matrix $M^{(\text{iter})}$ 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 the iteration matrix 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 the number of Jacobian evaluations in `npds` and of the decompositions in `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 will become 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 that the number of linear solves in `nsolves` decreases in all cases against the corresponding cases in the previous tables, presumably as a result of the better physical information provided in the system matrix $M^{(\text{iter})}$. This immediately implies a decrease, compared to the corresponding cases in the previous tables, of the number of function evaluations in `nfevals`, an observation that is also confirmed by lower averages of Newton iterations reported in mean(`iter`). Using the better physical information in the Jacobian also improves the effectiveness of the methods. First of all, the method without 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 compared to norm-wise error control. For the cases that converged and that had a mass error larger than round-off, the accuracy of the mass conservation is now 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 gives errors of the total mass that are acceptably small in Tables 3.3 and 3.4, whereas clipping can still incur much more significant errors in the mass. However, although the behavior is better than in the earlier tables, significant negative numbers still appear in some intermediate Newton iterates for the constraint-following method. Notice that Table 3.4 (a) with a converged solution without enforcement finally allows a comparison of its efficiency with those of the methods with non-negativity enforcement. It turns out that 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 in fact the most efficient of all cases with the predictor p_n as initial guess for the Newton method 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 case among all cases considered in this section, and its results were used for the plots in Figure 3.1. In Tables 3.3 and 3.4, it turns out for the Robertson problem that using the initial guess y_n for the Newton method does not lead to negative solution components even without non-negativity enforcement. However, there is no guarantee of this behavior in general, as is demonstrated in the next section. These cases also exhibit better ODE efficiency again, whereas the nonlinear solver uses again more iterations, because y_n contains less information about the solution than the predictor p_n.

4. The Interface Problem.

4.1. The Problem. The example in this section 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 $2\,\mathrm{A}+\mathrm{B} \rightarrow (*)$. Classical modeling for this process results in a system of reaction-diffusion equations coupled

through the non-linear reaction terms. Despite its classical nature, the resulting model becomes mathematically intriguing as well as numerically challenging if one considers a particular reaction pathway comprising two reactions with widely varying rate coefficients [27]: molecules of A and B combine in a first, 'fast' reaction to produce an intermediate C, while a second, 'slow' reaction combines A and C to form the product $(*)$, which is not explicitly tracked in the model. This reaction pathway is expressed by

$$\begin{aligned} \mathrm{A} + \mathrm{B} &\xrightarrow{\lambda} \mathrm{C}, \\ \mathrm{A} + \mathrm{C} &\xrightarrow{\mu} (*), \end{aligned}$$

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

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

We assume that no molecules flow through any part of the boundary, except that 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

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

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

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

We assume that the boundary and initial data are posed consistently; 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 due to the large coefficient $\lambda \gg 1$. For the corresponding stationary problem, given by (4.1)–(4.2) without time derivatives, analytical results in [16, 22] prove that the reaction rate of the fast reaction q has one internal layer at a point $0 < x^* < 1$ of width $O(\varepsilon)$ and height $O(1/\varepsilon)$ with the scaling $\varepsilon = \lambda^{-1/3}$. 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 [27] for studies of several representative initial conditions for this problem and [19] for studies on the asymptotic behavior of the transient problem.

To select a transient problem for testing that has the stationary solution just described and an interesting transient behavior, we select an initial condition with three interfaces specified by the initial condition functions for (4.3) chosen as

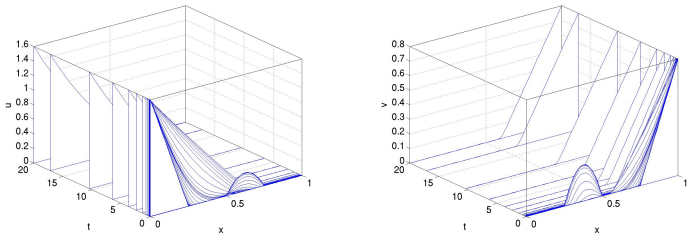
$$
\begin{aligned}
u_{\text{ini}}(x) &= \begin{cases} 4(0.25-x)\,\alpha, & 0.00 \le x \le 0.25, \\ 0, & 0.25 < x < 0.50, \\ 64(0.50-x)(x-0.75)\,\gamma, & 0.50 \le x \le 0.75, \\ 0, & 0.75 < x \le 1.00, \end{cases} \\
v_{\text{ini}}(x) &= \begin{cases} 0, & 0.00 \le x < 0.25, \\ 64(0.25-x)(x-0.50)\,\delta,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

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 at their respective Dirichlet boundary conditions and one parabolic 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.

In the method-of-lines approach, spatial approximations $u_j(t) \approx u(x_j, t)$, $v_j(t) \approx v(x_j, t)$, and $w_j(t) \approx w(x_j, t)$ at mesh points x_j, $j = 1, \ldots, N$, are introduced. These approximations are used to discretize the spatial derivatives in the system of PDEs in (4.1). Specifically, if the finite difference method is used for these approximations, an ODE system of the form $y' = f(t, y)$ is obtained by arranging the unknown functions in vector form such as $y(t) = [u_1, u_2 \ldots, u_N, v_1, v_2 \ldots, v_N, w_1, w_2, \ldots, w_N]^T$, and collecting all other terms than y' on the right-hand side. We note that for computational efficiency, the components of $y(t)$ should be re-ordered in interleaved ordering, that is, $y(t) = [u_1, v_1, w_1, u_2, v_2, w_2, \ldots, u_N, v_N, w_N]^T$, which we accomplish by a remapping whenever needed.

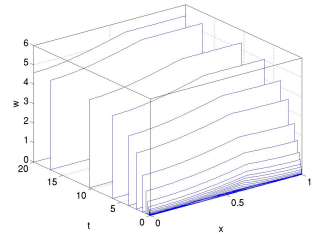
Alternative approaches for the spatial discretization include the finite element method, whose formulation naturally leads to a mass matrix M in the ODE system $My' = f(t, y)$ with M not being the identity matrix; see, e.g., [10] for a reaction-diffusion system in three dimensions using a slightly different version of our non-negativity preserving algorithm. We use a finite difference method in this paper for the spatial approximation in order to arrive automatically at an ODE system with $M = I$ to allow for a direct comparison with the non-negativity preserving method implemented in Matlab's `ode15s` function which requires this condition.

We recall that the analytical results in [16, 22] prove that the width of the internal layer at steady state is of order $\varepsilon = \lambda^{-1/3}$. For our choice of $\lambda = 10^6$, this is $\varepsilon = 0.01$. Hence, to ensure a sufficient spatial resolution, we use $N = 513$ mesh points in

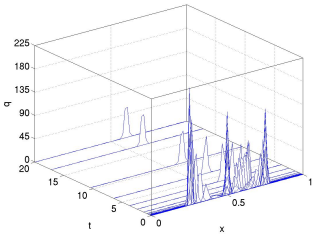


(a) u vs. (x,t)

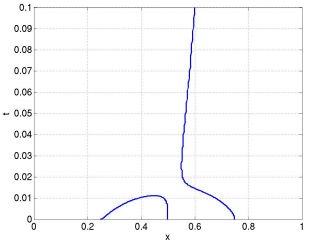
(b) v vs. (x,t)



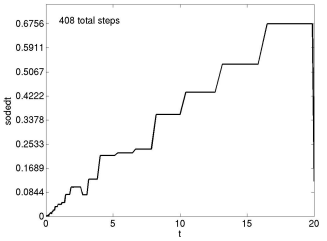
(c) w vs. (x,t)



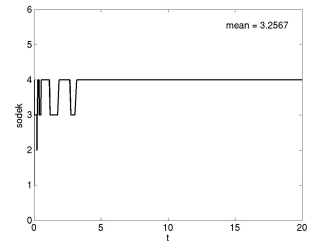
(d) $q = \lambda uv$ vs. (x,t)



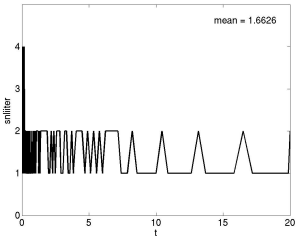
(e) interface vs. (x,t)



(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 $0 \le t \le 0.1$, (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.)*

$\Omega = (0, 1)$; 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; we have checked the calculations with both coarser and finer spatial meshes and have found the calculations to be reliable. The ODE tolerances used include a relative tolerance of 10^{-6} and absolute tolerance of 10^{-8}, which are chosen fairly tightly so as 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 $\varepsilon^{(\text{neg})} = 10^{-12}$ for our method of non-negativity preservation by damping.

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). At the initial time, u and v have a linear shape at their respective Dirichlet boundary conditions of $u = \alpha = 1.6$ at the left and $v = \beta = 0.8$ at the right end of the interval Ω, and both components are also non-zero in complementary regions in the interior of Ω. Figure 4.1 (d) shows the reaction rate $q = \lambda uv$ of the fast reaction vs. (x,t). At the initial time, it is zero in most of the interval 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, where each is fed by its Dirichlet boundary condition. 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 the initial time. To analyze the evolution of the interface 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 three locations at $x = 0.25, 0.50, 0.75$ at the initial time, 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), (g), and (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 outside of the initial transient. It turns out that the ODE method order used is fairly high with order 3 or 4 used at many time steps. We see that also the non-linear solver behaves well with only 1 or 2 iterations needed at most time steps.

4.2. Numerical Studies. The Robertson problem in section 3 is an excellent test case for the important property of mass conservation and also shows well the importance of maintaining non-negativity because the solution may otherwise be unstable. We note here that the interface problem is not as badly behaved as this, in that small negative components do not lead to blow-up and can be controlled by tightening the ODE tolerance. This was done in [27] to establish our confidence in the solution obtained. However, the Robertson problem is a small ODE system of only three equations; thus the 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 greatly increased importance, and we thus also report the wall clock time used to compute the solution in Tables 4.1 through 4.4. The other entries in the tables have the same meaning as those in the tables of the previous section. Because the product of the chemical reactions is not tracked by the model for the interface problem, mass is not conserved by this three-species model and we cannot readily

TABLE 4.1
Solution statistics for the interface problem with Jacobian update not *forced whenever* Δt *or* k *change and* component-wise *error control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	1095	0	860	0
min(y)	–1.06e–08	0.0	–5.93e–09	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	770	775	775	784
`nfailed`	44	42	46	57
`nfevals`	1586	1592	1599	1669
`npds`	28	30	30	38
`ndecomps`	143	142	145	156
`nsolves`	1585	1591	1597	1668
`nclips`	N/A	1140	358	23
mean(k)	4.69	4.66	4.65	4.63
mean(`iter`)	1.94	1.94	1.94	1.96
time (seconds)	35	35	36	37
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	1099	0	957	0
min(y)	–5.56e–07	0.0	–5.56e–07	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	896	882	896	880
`nfailed`	304	299	304	303
`nfevals`	3635	3598	3636	3588
`npds`	287	284	287	287
`ndecomps`	427	424	427	429
`nsolves`	3634	3597	3634	3587
`nclips`	N/A	1203	198	7
mean(k)	4.21	4.25	4.21	4.21
mean(`iter`)	3.32	3.35	3.32	3.33
time (seconds)	80	80	83	80

compute a mass error.

We begin by noting that all simulations in Tables 4.1 through 4.4 converged without blow-up. However, in all tables and for both choices of the initial guess for the Newton method, the method without non-negativity enforcement as well as the constraint-following method suffer from negative intermediate Newton iterates, whereas clipping and damping do not. This demonstrates that using y_n as initial guess of the Newton method does not guarantee non-negativity, as it happened to do in the Robertson problem.

As the comparison of all efficiency data in Tables 4.1 through 4.4 shows, the four methods studied exhibit approximately the same numerical efficiency. That is, none of the non-negativity preserving methods costs much additional effort. Comparing now part (a) and part (b) of each of the four tables indicates that, for this large ODE system resulting from the semi-discretization of a PDE, it is generally more efficient to use the predictor p_n as initial guess for the Newton method than to use the old solution y_n. This is brought out by the number of ODE steps and the wall clock times as overall measures of efficiency, but also particularly by the Newton solver taking fewer steps to converge and the ODE method order being higher on average

TABLE 4.2

Solution statistics for the interface problem with Jacobian update not *forced whenever* Δt *or* k *change and* norm-wise *error control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	598	0	450	0
min(y)	–7.71e–06	0.0	–3.93e–06	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	344	338	368	471
`nfailed`	33	32	39	92
`nfevals`	755	742	805	1081
`npds`	30	29	30	61
`ndecomps`	90	89	99	177
`nsolves`	754	741	803	1080
`nclips`	N/A	577	175	343
mean(k)	3.59	3.60	3.62	3.13
mean(`iter`)	1.94	1.94	1.92	1.85
time (seconds)	18	17	19	26
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	1008	0	792	0
min(y)	–3.90e–04	0.0	–4.01e–04	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	583	910	583	398
`nfailed`	167	182	167	147
`nfevals`	1732	2428	1734	1210
`npds`	69	68	69	129
`ndecomps`	271	324	271	211
`nsolves`	1731	2427	1732	1209
`nclips`	N/A	1665	135	183
mean(k)	2.80	2.43	2.80	3.19
mean(`iter`)	2.24	2.18	2.24	2.26
time (seconds)	39	54	40	30

for the predictor as initial guess in practically all cases.

To analyze the effect of the choice of component-wise vs. norm-wise error control, compare Table 4.1 against Table 4.2, and Table 4.3 against Table 4.4. Even though 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 and hence is more costly. The physical results for this PDE system, such as min(y) and max(y) do not bear out any advantage of this stringency, though. This justifies 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 the choice of whether to update the Jacobian only when the error control requires its re-evaluation or also when Δt or k change, 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 as evidenced by similar numbers of ODE steps and average ODE orders. However, the wall clock time clearly indicates a distinct advantage of using the better physical information. This results from fewer

TABLE 4.3
Solution statistics for the interface problem with Jacobian update forced whenever Δt *or* k *change and* component-wise error *control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	745	0	560	0
min(y)	–3.46e–09	0.0	–4.17e–09	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	764	765	764	772
`nfailed`	16	17	17	30
`nfevals`	1133	1147	1160	1263
`npds`	114	116	113	129
`ndecomps`	114	116	113	129
`nsolves`	1132	1146	1158	1262
`nclips`	N/A	736	340	21
mean(k)	4.75	4.72	4.75	4.69
mean(`iter`)	1.44	1.45	1.47	1.55
time (seconds)	29	29	30	32
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	997	0	865	0
min(y)	–8.54e–11	0.0	–8.54e–11	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	867	867	867	877
`nfailed`	235	235	235	235
`nfevals`	3207	3207	3208	3230
`npds`	352	352	352	355
`ndecomps`	352	352	352	355
`nsolves`	3206	3206	3206	3229
`nclips`	N/A	1101	198	13
mean(k)	4.20	4.20	4.20	4.23
mean(`iter`)	3.12	3.12	3.12	3.11
time (seconds)	73	73	75	73

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

In summary, the interface problem exhibits a somewhat different behavior than the Robertson problem. The ODE method shows a much higher average method order for the interface problem, pointing to its smoothness if the time step is sufficiently small. The interface problem confirms 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, thus the methods of Tables 4.3 (a) and 4.4 (a) should be preferred. We believe this can be generally expected, especially for large ODE systems. Between these two tables, the norm-wise error control turns out to be significantly more efficient; thus Table 4.4 (a) lists the preferred methods for the interface problem. Only the non-negativity preservation methods of clipping and damping are effective for all intermediate Newton iterates. Based on the results for the interface problem alone, we could not recommend one method over the other; in fact, in several tables, notably Table 4.4 (a), clipping turns out to be slightly more efficient, though this is

TABLE 4.4

Solution statistics for the interface problem with Jacobian update forced whenever Δt or k change and norm-wise *error control in the ODE method.*

(a) Initial guess for Newton iteration: predictor p_n				
	no enforcement	clipping	constraint	damping
`nnegative`	482	0	336	0
min(y)	–2.88e–06	0.0	–2.88e–06	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	345	344	365	408
`nfailed`	13	12	19	57
`nfevals`	617	605	650	800
`npds`	69	69	78	124
`ndecomps`	69	69	78	124
`nsolves`	616	604	648	799
`nclips`	N/A	471	156	242
mean(k)	3.67	3.51	3.66	3.26
mean(`iter`)	1.69	1.68	1.66	1.66
time (seconds)	16	15	17	21
(b) Initial guess for Newton iteration: old solution y_n				
	no enforcement	clipping	constraint	damping
`nnegative`	666	0	485	0
min(y)	–6.51e–05	0.0	–6.51e–05	0.0
max(y)	5.4211	5.4211	5.4211	5.4211
`nsteps`	470	470	470	424
`nfailed`	82	83	82	113
`nfevals`	1260	1264	1261	1155
`npds`	151	152	151	177
`ndecomps`	151	152	151	177
`nsolves`	1259	1263	1259	1154
`nclips`	N/A	660	91	122
mean(k)	3.14	3.14	3.14	3.14
mean(`iter`)	2.16	2.16	2.16	2.16
time (seconds)	30	30	31	29

not the case in some other tables. This is why the Robertson problem is insightful: it demonstrates the better mass conservation ability of non-negativity preservation by the damping method. As the efficiency comparisons here show, this better physical behavior may cost a modest amount of additional computational effort.

REFERENCES

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

[2] U. M. ASCHER AND L. R. PETZOLD, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, 1998.

[3] K. E. ATKINSON, *An Introduction to Numerical Analysis*, John Wiley & Sons, second ed., 1989.

[4] 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.

[5] 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.

[6] P. N. BROWN AND Y. SAAD, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 450–481.

[7] R. S. DEMBO, S. C. EISENSTAT, AND T. STEIHAUG, *Inexact Newton methods*, SIAM J. Numer.

Anal., 19 (1982), pp. 400–408.
[8] J. E. Dennis, J. J. E. Dennis, and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, 1996.
[9] 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.
[10] M. K. Gobbert, *Long-time simulations on high resolution meshes to model calcium waves in a heart cell. SIAM J. Sci. Comput.*, in press (2008).
[11] 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.
[12] 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.
[13] 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.
[14] 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`.
[15] 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.
[16] 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.
[17] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, vol. 16 of Frontiers in Applied Mathematics, SIAM, 1995.
[18] *MATLAB Release R2006b (August 03, 2006).* The MathWorks, Inc., `www.mathworks.com`.
[19] 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)*, accepted (2008).
[20] 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.
[21] A. Sandu, *Positive numerical integration methods for chemical kinetic systems*, J. Comput. Phys., 170 (2001), pp. 589–602.
[22] 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.
[23] L. F. Shampine, *Conservation laws and the numerical solution of ODEs*, Comput. Math. Appl. Part B, 12 (1986), pp. 1287–1296.
[24] ———, *Linear conservation laws for ODEs*, Comput. Math. Appl., 35 (1998), pp. 45–53.
[25] L. F. Shampine and M. W. Reichelt, *The MATLAB ODE suite*, SIAM J. Sci. Comput., 18 (1997), pp. 1–22.
[26] 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.
[27] 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.