JOURNAL OF COMPUTATIONAL PHYSICS **144,** 280–298 (1998)
ARTICLE NO. CP986008

Solving Stiff Differential Equations with the Method of Patches

David Brydon,*,† John Pearson,† and Michael Marder*

*Department of Physics, Center for Nonlinear Dynamics, University of Texas at Austin, Austin, Texas 78712; and †Los Alamos National Laboratory, MS B258, Los Alamos, New Mexico 87545
E-mail: brydon@lanl.gov or brydon@physics.utexas.edu; pearson@lanl.gov; and marder@chaos.ph.utexas.edu

Received April 7, 1997; revised January 27, 1998

We introduce a new method for solving very stiff sets of ordinary differential equations. The basic idea is to replace the original nonlinear equations with a set of equally stiff equations that are piecewise linear, and therefore can be solved exactly. We demonstrate the value of the method on small systems of equations for which some other methods are inefficient or produce spurious solutions, estimate error bounds, and discuss extensions of the method to larger systems of equations and to partial differential equations. © 1998 Academic Press

I. INTRODUCTION

Differential equations are called stiff when two or more very disparate time scales are important. Although many special methods have been developed to integrate stiff sets of differential equations, there remain problems governed by such enormously varying time scales that existing methods are inefficient or produce spurious solutions. We have developed a method that is insensitive to stiffness.

We break the domain of the problem into adjoining patches. On each patch, we approximate the original ordinary differential equations with linear equations for which analytical solutions are known. Solution of the original problem reduces to solving continuous linear approximate equations on these patches. Because the method finds the exact solution of an approximate problem, rather than an approximate solution of the exact problem, it is impervious to many numerical instabilities that plague other techniques. In addition, the approximating functions can be chosen to agree with the exact functions so as to preserve geometrical features of the original problem, such as fixed points. Many types of differential equations can be solved with the method.

The linear method of patches solves two separate sub-problems to approximate the solution of nonlinear initial-value ordinary differential equations. The first sub-problem is to

0021-9991/98 $25.00

approximate the nonlinear derivative functions on the right hand side of the original equations by linear functions $Ax + b$. The second sub-problem is to solve the linear ordinary differential equation $\dot{x} = Ax + b$ on each patch. The solution $x(t)$, continuous in time, is the approximate solution for this patch. In contrast, most numerical methods apply a time discretization scheme to the nonlinear derivative functions, and produce discrete solutions whose validity depends upon the time step used and the function discretized. Analyzing the interaction of a given discretization scheme with continuous nonlinear derivative functions can be very difficult.

Our method is not without its own difficulties. There is no complete theory of how to approximate a set of nonlinear functions by linear patches, and solving $\dot{x} = Ax + b$ for general A is not trivial. While these challenges require more research, we feel splitting the task into these separate sub-problems holds promise for obtaining verifiable and efficient approximate solutions of many difficult differential equations.

The accuracy of the solution is determined by the size h of the side of a patch within which the original problem is linearized. The error incurred in crossing a single patch is $\mathcal{O}(h^3)$, and the error involved in integrating over many patches of fixed size scales as $\mathcal{O}(h^2)$. We will show that there exists a more accurate linear approximation in one variable for which the error crossing a single patch is $\mathcal{O}(h^5)$ for sufficiently smooth functions.

This paper is organized as follows: Section II discusses a one variable example, the logistic equation, in order to introduce the basic ideas of the method and to show differences between this method and discrete methods. Computer implementation and other issues of the method are outlined in Section III. We then discuss in detail the application of these ideas to some difficult problems in two variables in Section IV. Finding an optimal procedure to fit linear functions to nonlinear functions for this method is explored in Section V. Section VI gives general bounds on the error of the method in N variables. In Section VII we discuss other types of equations, and the extension to partial differential equations. Section VIII reviews the literature, and is followed by a conclusion.

II. EXAMPLE IN ONE VARIABLE: THE LOGISTIC EQUATION

We begin by illustrating our method on the logistic equation

$$\dot{x} = f(x) = \lambda x(1 - x) \tag{1}$$

because it provides a particularly simple context in which to describe the method, not because it is particularly demanding to solve numerically. However, standard methods of discretization can easily produce spurious solutions of Eq. (1) [1], and we will show why our method is immune to these common difficulties. In one variable, we approximate the curve representing the derivative function by a set of non-overlapping line segments.

We replace $f(x)$ with $f_L(x)$,

$$f_L(x) = \sum_{i=0}^{N-1} (a_i x + b_i) C_i(x), \tag{2}$$

where $C_i(x)$ is the characteristic function of the interval $[x_i, x_{i+1}]$, equal to 1 on the interval and 0 outside it, and $x_0 < x_1 < \ldots < x_N$ partition the solution domain into N patches. It is

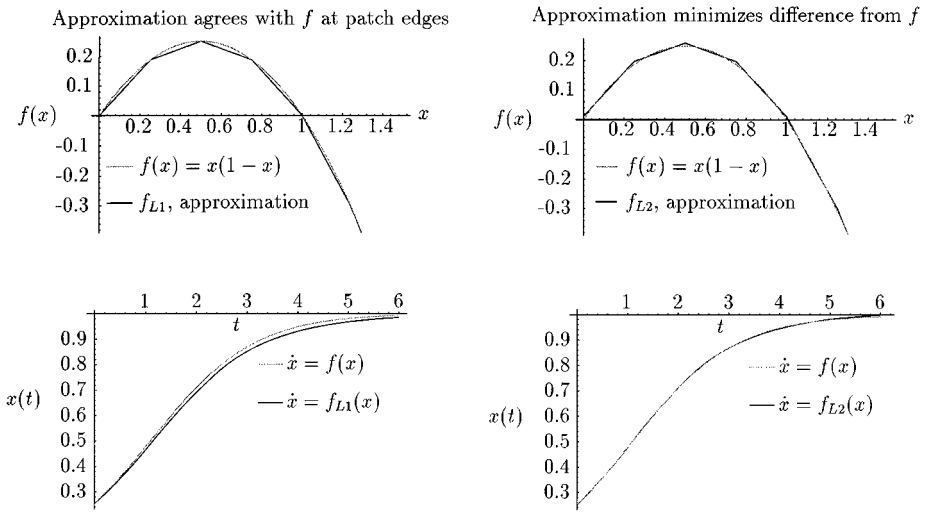


FIG. 1. Standard and fitted patches with solutions generated from them.

quite important to take $x_0 = 0$ and $x_N = 1$, since they are fixed points of f, but otherwise the partition can be chosen in any way that best serves accuracy and speed.

We choose the constants a_i, and b_i so that

$$f_L(x_i) = f(x_i) \equiv f_i. \tag{3}$$

Then

$$a_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}, \qquad b_i = f_i - a_i x_i. \tag{4}$$

The procedure of piecewise linearization is similar to the finite element method, except that the piecewise linear sum represents the right hand side of the equation rather than its solution. Figure 1 pictures f and f_L from Eq. (1), along with the exact and approximate solutions. Given an initial condition x_o that lies in patch i, we use the solution of

$$\dot{x}_L = f_L(x) = a_i x + b_i \tag{5}$$

which is

$$x_L(t) = x_o e^{a_i t} + \frac{b_i}{a_i}(e^{a_i t} - 1). \tag{6}$$

We can solve Eq. (6) exactly for the time when $x_L(t)$ reaches a given value such as the patch boundary. We find the time when the solution reaches the edge of the patch and repeat the process on the next patch. Since the solution is continuous in time, a computer code can return the solution at any times requested by the user.

II.A. *Analytic Error Results for the Logistic Equation*

The exact solution of the logistic equation (1) allows us to derive an analytical expression for the difference between the exact solution and the solution of a linearization. We call this

error $e(t) \equiv x(t) - x_L(t)$. The exact solution of (1) is $x(t) = x_o/(x_o + (1 - x_o)e^{-\lambda t})$ where $x_o = x(0)$. The exact solution of the linearization is $x_L(t) = e^{a_i t}x_o + (b_i/a_i)(e^{a_i t} - 1)$, where a_i is the slope and b_i is the intercept of the linearization on the patch. If we use a fixed patch size Δx and if x_i is the value of x on the left side of patch i, then we can calculate exactly the error after crossing one patch. If we let the linearization agree with the exact function on the patch boundaries, then a_i and b_i are given by (4). The error after crossing the patch is

$$e(t_{i+1}) = \left\{ \begin{array}{ll} 0, & x_i = 0, 1 \\ \frac{1}{6(1-x_i)x_i}\Delta x^3 + \mathcal{O}(\Delta x^4), & x_i \neq 0, 1 \end{array} \right\}, \tag{7}$$

independent of λ. This Δx^3 dependence has two parts. The maximum difference between the exact derivative function and the linearization is $\mathcal{O}(\Delta x^2)$ for twice-differentiable f [2]. The second part comes from the time to cross a patch, proportional to Δx except in the special case of a fixed point in the patch which captures the solution for all time. In Section VI, we show that this result holds generally for linearizations whose difference from f is $\mathcal{O}(\Delta x^2)$.

II.B. *Fitted Approximations*

If instead of asking the approximation f_L to agree with f at the endpoints of the patches, we choose f_L to minimize the mean square deviation from f over the patch, then the error is $\mathcal{O}(\Delta x^5)$, as shown in Section V. This improved accuracy is illustrated in Fig. 1.

II.C. *Comparison to Difference Methods*

Although the accuracy of our solution depends on Δx, it does not depend on λ in Eq. (1). Further, there is no stability criterion, which helps explain why our method is particularly suited to stiff equations. Consider a traditional finite-difference time discretization of the logistic equation. In the simplest case, one writes

$$x_{n+1} = x_n + \Delta t \lambda x_n(1 - x_n), \tag{8}$$

where x_{n+1} is the value of x given by this finite difference equation at time $t_{n+1} = t_n + \Delta t$.

For any given Δt, one can find λ large enough that the discrete map (8) displays the period-doubling bifurcation sequence, and exhibits solutions that bear no resemblance to those of the original ordinary differential equation. Yee [1, p. 272] points out ways in which several standard discretizations give spurious solutions. For example, the fourth order Runge–Kutta method applied to the cubic logistic equation gives stable spurious fixed points below the linearized stability limit, and these spurious fixed points could be achieved in practice (see also [3] for a review of these issues). The problem can be eliminated in the simple case of the logistic equation by a proper choice of Δt, but in complicated sets of stiff equations, Δt is not easy to choose well, and our method does not require one to choose it. Since stability is not an issue, we may focus on accuracy and efficiency, and upon geometrical features of the equations.

III. DETAILS OF BASIC IMPLEMENTATION

The method of patches is easy to implement for a single variable, but in two dimensions it acquires some difficulties which require additional explanation.

III.A. *Forming the Approximate Equations*

We wish to solve autonomous ordinary differential equations of the form

$$\begin{aligned} \frac{dx(t)}{dt} &= f(x, y) \\ \frac{dy(t)}{dt} &= g(x, y). \end{aligned} \tag{9}$$

We approximate f and g by linear equations on a set of patches in the (x, y) domain. We solve the linear equations in each patch,

$$\begin{aligned} \frac{dx_L(t)}{dt} &= ax_L + by_L + c \\ \frac{dy_L(t)}{dt} &= dx_L + ey_L + f, \end{aligned} \tag{10}$$

where a through f are constants. The approximate solution is built as it starts from a given initial (x_0, y_0) and passes continuously through many patches. Example equations are solved in Subsections IV.A and IV.B. Patches on a domain and an approximate solution are illustrated in Figs. 2 and 3 of Subsection IV.A. In this section we describe the implementation and issues that it brings up.

The method places no constraints upon the sizes and shapes of the patches used, although triangles can form a continuous surface, while rectangles cannot. Once a mesh has been chosen, and the approximate linear functions $\vec{f}_L$ constructed upon it, one should inspect $\vec{f}_L$ to find all their null-clines and fixed points for comparison with those of the original derivative functions $\vec{f}$. This can be done ahead of time for a given domain, or checked as the code runs (a compile flag could turn this checking off and on: off for greater speed, and on to verify a valid solution). If the null-clines and fixed points are not qualitatively of the correct type, they may cause spurious solutions. If the fixed points of the exact function cannot be found, or if they are too numerous or complicated, then we may have trouble knowing that we have represented them faithfully. In that, we are no worse off than traditional methods. However, many systems solved in practice have a finite number of fixed points that we can represent accurately. In the case of unknown fixed points, one can use higher and higher resolution, verifying that the solution is unchanged. In Section VI we prove that as the mesh size goes to zero, the approximate solution converges to the exact solution.

Our working code calculates the linearizations on a regular triangular grid, given the origin of the grid, as the solution proceeds. Since the code calculates the triangles as it goes, the domain is infinite unless the user specifies boundaries for valid solutions. For example, solutions to the Gaspar–Showalter equations (13) are positive for positive initial conditions. An earlier code stored to disk a pre-calculated grid over a fixed domain. This proved to be slower, as well as restrictive in memory requirements and the need to know the solution domain before solving.

The simple way in which we construct the function $\vec{f}_L$ in two variables is to evaluate $\vec{f}$ at the nodes of a rectangle, divide each rectangle into two triangular patches, and build $\vec{f}_L$ as the unique linear interpolation of $\vec{f}$ between these nodes. In patches containing a fixed point, one may insist that the location of the fixed points of $\vec{f}$ and $\vec{f}_L$ coincide, if possible. It may be valuable to change variables so as to make their variations as uniform as possible. An example of this procedure is given in Subsection IV.B when we change variables from ϕ to $f = -\log_{10}\phi$.

III.B. *Solving the Approximate Equations*

For the one variable autonomous approximation, $\dot{x} = ax + b$, we can solve exactly for the time when the solution leaves the patch (if the solution never leaves the patch, the code simply continues to report the solution at requested times). But for non-autonomous or nonlinear approximating equations in one variable, or for general N-variable equations, there is no analytical formula for the time when the current solution leaves its patch. Bisection is a robust and simple way to find the time at which the approximate solution leaves its patch. Local time for the initial point is chosen to be zero. We then guess the time at which the solution leaves the patch using the initial $\dot{x}$ and the patch size. The time is increased or decreased until we have a point inside and a point outside the patch. We then choose an intermediate time and iterate, always keeping one point inside and one point outside the patch.

When the solution leaves the current patch, we must know to which patch it goes. Therefore a tolerance is specified, and we bisect until the solution is within tolerance *outside* the boundary of the current patch. Then we see which patch we have fallen into and repeat the entire cycle.

The only danger of bisecting is that our first guess of the exit time might be at a point when the solution has left the patch and returned, not giving us the first exit time that we want, but a later one. We try to avoid this error by taking the first time guess, obtained by dividing the patch size by the derivative at the point, and dividing it by a large number, for example, 10^4. This small first time ensures that the corresponding first point is well within the patch, near the initial point at $t = 0$. We could skip this drastic reduction of the initial time guess, which makes the code much less efficient, by analyzing the solution to see if its time derivatives change sign for values of the variables on or near the patch. Since we are solving a linear system, this is a trivial analysis. Sacks [4] does it for two variables. If the time derivatives do not change sign, there is no danger of the solution leaving and re-entering the patch.

III.C. *Solving the Inhomogeneous System in Each Patch*

We wish to solve nonlinear autonomous ordinary differential equations with N dependent variables by solving a piecewise linearization. In summary, we will solve some nonlinear ODE $d\mathbf{x}/dt = f(\mathbf{x})$, where $\mathbf{x}(t)$ is a vector of dependent variables and $f(\mathbf{x})$ is a vector of nonlinear functions of the components of $\mathbf{x}$, by making a set of local linear approximations to the right-hand side of the equation, $f(\mathbf{x}) \approx f_L(\mathbf{x}) = \{A_i\mathbf{x} + b_i\}$, with a distinct $A_i\mathbf{x} + b_i$ defined on each patch.

The solution to $d\mathbf{x}(t)/dt = A\mathbf{x} + \mathbf{b}$, $\mathbf{x}(t)$ a time-dependent N-vector, A a constant $N \times N$ matrix, $\mathbf{b}$ a constant N-vector, is [5]

$$\mathbf{x}(t) = e^{At}\mathbf{x}(0) + \int_0^t e^{At'}\mathbf{b}\, dt', \tag{11}$$

which can be integrated,

$$\mathbf{x}(t) = e^{At}\mathbf{x}(0) + A^{-1}(e^{At} - I)\mathbf{b}, \tag{12}$$

where e^{At} is the matrix exponential of At, A^{-1} is the matrix inverse of A, and I is the identity matrix. This solution is always correct if A^{-1} exists, although degenerate eigenvectors in A require the Jordan form. Also, if A has imaginary eigenvalues, then the solution involves the sine and cosine functions. Efficient and accurate evaluation of Eq. (11) is crucial to

the success of the method of patches, and there are many schemes which might be used. Our goal in picking a scheme for evaluating the solution of $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{b}$ is to find the time when the solution leaves some region, which usually requires evaluating the solution for many different values of t. Discussion of these schemes will be left to future papers, but we reference as an excellent starting point the paper by Moler and Van Loan [6]. Putzer's method [7] for evaluating e^{At} applied to solving $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{b}$ has given robust solutions in the two-variable case. It requires only the eigenvalues of A in addition to A, $\mathbf{b}$, and $\mathbf{x}(0)$, and returns only scalar functions of time. It does not require A^{-1} or the Jordan form. We use it in our two-variable code. Putzer's method will be more fully discussed in a later paper. The main limitation of Putzer's method is that it requires the eigenvalues of A. If A is ill-conditioned for eigenvalues, the error in the eigenvalues will lead to errors in the solution of the linear differential equation. The impact of such errors requires further research. Since it needs all the eigenvalues of A, Putzer's method is not practical for more than ten variables or so, unless we make further approximations. Making further approximations to solve PDEs is discussed in Section VII.

III.D. *Issues of Continuity and Fitting*

As shown in the one variable example in Fig. 1, fitting the approximate linear surface to the exact nonlinear function, as opposed to connecting points in the nonlinear function, can increase the accuracy of the solution for fixed patch size. Fitting the linearization to the exact function separately in each patch (or finite precision roundoff) can result in discontinuities between the linearizations in neighboring patches. Such discontinuities can cause an error if the solution leaves the edge of one patch where the approximate derivative function is of one sign and enters a patch where that approximate derivative function is of the opposite sign. This is possible with discontinuous approximations because of our bisection method of solution, which follows the approximation until it is just outside of the patch boundary. The solution can leave the edge of the first patch that has, say, a positive derivative in the direction perpendicular to the patch boundary crossed. Control is passed to the neighboring patch, which has a negative derivative. But the solution in this patch then returns to the first patch. A long "game of ping-pong" can be played in this way across the edge between two such patches. We observed this behavior in Eqs. (13). To avoid this ping-pong effect, a code can check for such an event and either make the nodes agree to machine precision or move zero from the edge to within one of them. Since this problem is so easily corrected locally, we plan to use rectangles in future codes. Although rectangles form a discontinuous approximation, they simplify coding and make the N-variable problem easy to implement.

IV. EXAMPLES IN TWO VARIABLES

We now intend to demonstrate that the method of patches provides an efficient, reliable, and easy way to integrate notoriously stiff differential equations. We will focus upon two pairs of equations in two variables, one describing chemical kinetics, and the other describing a laser oscillator.

Our demonstration is based upon direct comparison of the new method with packages designed for stiff ODEs. Of the packages, we had the greatest satisfaction with CVODE [8], which successfully solved equations for which other packages failed, and used much less computer time to return solutions to equations that the others could solve. Other packages tried were LSODE, IMSL stiff routines, and Numerical Recipes stiff routines.

One notable difference between the method of patches and finite-difference methods is that many standard methods fail to solve the example equations, and the variable-order, variable time-step packages that can solve them require the tolerances to be set very precisely in order to get valid solutions. While the small tolerances often give very accurate solutions, they cannot be increased in order to get faster but less accurate solutions. With the method of patches, there is no stability requirement to meet, so efficiency and accuracy can be traded, provided key aspects such as fixed points are approximated correctly. Thus, while all methods can fail when numerical grids are too coarse, the failure of a package like CVODE can be somewhat mysterious, while in the method of patches failure is directly tied to failure of the approximation to capture a qualitative geometrical feature of the original problem. Numerical causes of failure are also more easily analyzed, because they arise in the solution of the continuous linear ODE $dx/dt = Ax + b$.

IV.A. Gaspar–Showalter Equations

The Gaspar–Showalter equations arise in modeling the ferrocyanide–iodate–sulfite chemical reaction [9]. The variable X represents the concentration of HSO_3^-, and Y represents the concentration of H^+,

$$\begin{aligned}\frac{dX}{dt} &= -(k_{-1} + k_o + k_2)X - \frac{k_3 k_4 X Y^2}{k_o + k_5 + k_4 X} + \frac{k_1(A_o k_o + k_{-1} X)Y}{k_o + k_1 Y} \\ \frac{dY}{dt} &= (k_{-1} + k_2)X + \frac{3k_3 k_4 X Y^2}{k_o + k_5 + k_4 X} - \frac{k_1(A_o k_o + k_{-1} X)Y}{k_o + k_1 Y} - 2k_3 Y^2 + k_o(Y_o - Y)\end{aligned} \tag{13}$$

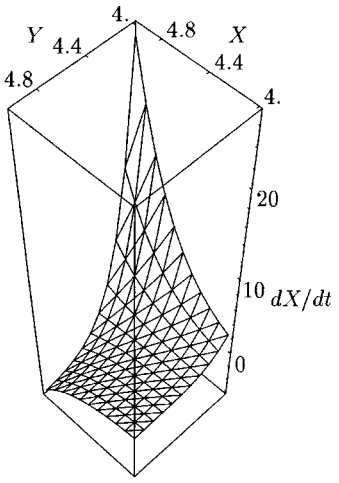
with values of rate constants,

$$\begin{array}{lll} k_1 = 5.0 \times 10^{10}, & k_2 = 6.0 \times 10^{-2}, & k_3 = 7.5 \times 10^4 \\ k_4 = 2.3 \times 10^9, & k_5 = 30, & k_o = 1.5 \times 10^{-3} \\ k_{-1} = 8.1 \times 10^3, & A_o = .09, & Y_o = 3.0 \times 10^{-3} \end{array}$$

and initial conditions $X(0) = Y(0) = 1 \times 10^{-5}$.

IV.A.1. *Results*

Figure 2 shows triangular patches approximating dX/dt and dY/dt on a small region in (X, Y) space, using a $-\log_{10}$ scale. Figure 3 is a schematic of an initial condition on this region and its subsequent time evolution through several triangular patches. Table 1 shows the fractional error and cpu time needed for the two solvers. At these values of the rate constants, studied in [9], $Y(t)$ oscillates between 10^{-10} and 10^{-2}, as shown in Fig. 4. Both CVODE and the method of patches (MOP) returned the correct amplitude of the oscillations. When converged, both codes returned an average period of 857.183 s over 20,000 s of model time. But as the patch size increased, MOP returned solutions with shorter and shorter period. The fractional error shown in Table 1 is the difference between the converged period and that solution's period, divided by the converged period. With sufficiently tight tolerances, CVODE returned solutions with no error, but a minimum CPU time of 1.3 s. For CVODE, tighter tolerances took more CPU time but returned the same solution, while looser tolerances gave either a failure to converge flag, or no error flag but spurious results such as negative values of the variables, not mathematically allowed in these equations for positive initial conditions. CVODE went to a false negative fixed point at a



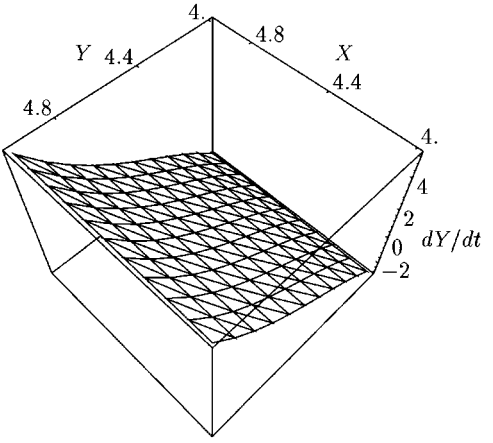


FIG. 2. Triangular patches approximating dX/dt and dY/dt. A $-\log_{10}$ scale is used for all axes.

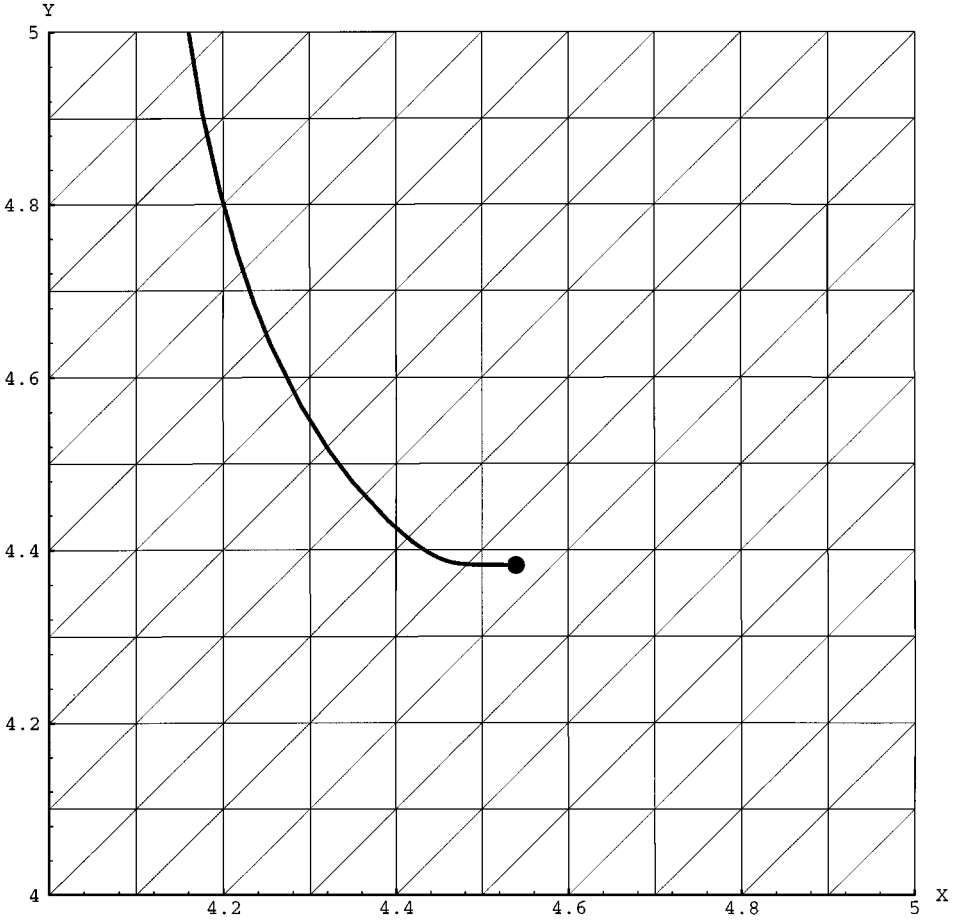


FIG. 3. Solution passing through many triangular patches.

TABLE 1
Solution Accuracy and cpu Time (s) for CVODE and the Method of Patches

Agreement with CVODE period	MOP cpu time	CVODE cpu time
100%	26.4 s	1.3 s
99.98%	5.7 s	Fails
97.4%	0.5 s	Fails
68.4%	0.3 s	Fails
20.8%	0.2 s	Fails

Note. This illustrates how the method of patches can trade accuracy for efficiency by changing the patch size. CVODE can return a converged solution for tight enough tolerances. At looser tolerances, CVODE either cannot converge or gives an erroneous solution.

finite value, or to negative infinity, depending on the tolerance settings. Thus CVODE could give an accurate solution, but it could not give a less accurate but geometrically correct and more efficient solution.

This implementation of MOP, limited by a fixed patch size and by not fitting the approximation to the derivative functions, returned a converged solution only for very high resolution, requiring 26.4 s of CPU time for a totally converged solution, 5.7 s for 99.98% accuracy, but only 0.5 s for 97% accuracy. Solutions exhibiting oscillations of correct amplitude but of much shorter period required just 0.2 s.

IV.A.2. *Technical Details and Discussion*

Both codes were timed using the *time* utility on a Silicon Graphics workstation with an R10K processor. They were both compiled by cc -64 -mips4 -r10000 -O3 using the $-lfastm$

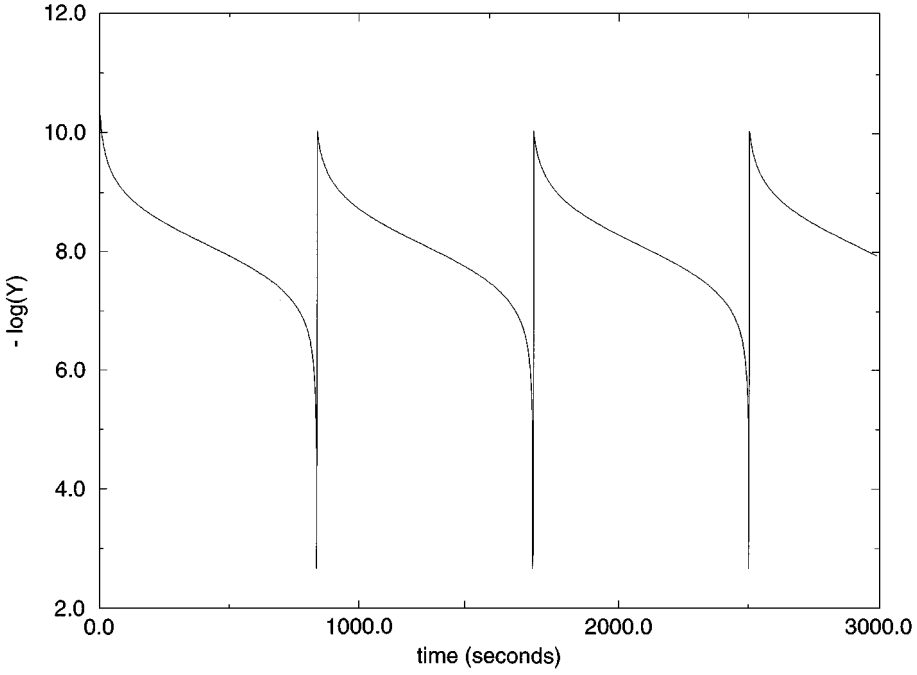


FIG. 4. Method of patches solution for $Y(t)$ in Gaspar–Showalter equations.

library of fast math functions under the *Irix* 6.2 Operating System. CVODE was set for stiff equations, using the BDF method, Newton iteration with the dense linear solver, and automatic Jacobian. The only option turned on was MXSTEP = 200000, because the default limit of 500 was not enough for CVODE to complete some steps. The error tolerances for CVODE were (RTOL $= 1.0 \times 10^{-9}$, ATOL $= \{1.0 \times 10^{-9}, 1.0 \times 10^{-15}\}$) for the most efficient converged solution.

In order to cover the range 10^{-10} to 10^{-2}, the method of patches uses a regular grid in logarithmic (base 10) coordinates. We used a grid of continuous triangular patches created by sampling the derivative functions on a rectangular grid in $-\log_{10}(X, Y)$ centered at (2, 2), and bisecting these rectangles along a diagonal. The rectangular grid spacing was (0.9, 0.9) for the solution requiring 0.2 s. A parameter of the method of patches, the tolerance window outside of a patch within which the solution is required to be located (Subsection III.B), was 0.08. If the tolerance was turned up to 0.8, the code went to a false fixed point. For the converged solution, grid spacing was (0.0003, 0.0003) and tolerance 0.0004. Note that the tolerance was larger than the grid spacing. This means that the initial guess of time step to reach the edge of the patch is likely to be accepted, meaning less time spent bisecting in t to find when the solution leaves the patch. It also tends to give more accuracy because using a linearization beyond the edges of the patch effectively averages the original function. An adaptive grid in the original variables would be much more efficient, as roughly 40–50% of this code's CPU time is spent evaluating the C function $pow(10, x)$ required by the logarithmic conversion. Storing the grid on disk is restrictive and slower, as the machine is able to calculate the linearization faster than it can be looked up in memory. This may vary with hardware.

A notable risk in using the method of patches is that one may introduce geometrically incorrect features into the problem. This difficulty is well illustrated by trying to solve the Gaspar–Showalter equations (13).

Upon linearizing the derivative functions on a (0.12, 0.12) grid, the null-clines (curves defined by $\frac{dX}{dt} = 0$ or $\frac{dY}{dt} = 0$) are not captured correctly. Many spurious fixed points appear when the linearized null-clines cross where they should not. Such false fixed points can be detected by comparing, in each patch, fixed points of the linearization with those of the exact system to verify that they are accurate in type and location. We show in Section VI, for differentiable functions, that in the limit where patch size goes to zero, the approximate and exact solutions converge. Upon proceeding to a (0.09, 0.09) uniform grid, the correct null-cline structure is recovered. Clearly a non-uniform grid can bring computational advantages in such a case, but we have not yet proceeded to such refinements of the method. False fixed points are easy to detect and correct. Other features of a dynamical system which determine the qualitative behavior may be more difficult to preserve. Qualitative dynamics is an area of active research which is applicable to constructing linearizations for use in this method [4].

IV.B. Laser Oscillator Model

This model [10, p. 32] represents a ruby laser oscillator where ϕ is photon density and n is dimensionless population inversion, and is specified by

$$\begin{aligned} \frac{dn}{dt} &= -n(\alpha\phi + \beta) + \gamma, \\ \frac{d\phi}{dt} &= \phi(\rho n + \sigma) + \tau(1 + n), \end{aligned} \tag{14}$$

with values of parameters $\alpha = 1.5 \times 10^{-18}$, $\beta = 2.5 \times 10^{-6}$, $\gamma = 2.1 \times 10^{-6}$, $\rho = 0.6$, $\sigma = 0.18$, $\tau = 0.016$ and initial conditions $n(0) = -0.8$, $\phi(0) = 10^{-12}$.

IV.B.1. *Package Solver*

In [10, p. 46], Byrne and Hindmarsh say, "This problem is challenging because it is stiff initially, but mildly damped and oscillatory later.... The catch is that either quite a little analysis to observe this is required or some numerical computation must be done." To solve this problem with discrete methods, one must first discover that it is stiff, pick a method appropriately, and choose good tolerances. To make the solution efficient, one must choose a stiff method in the early part of the run, and then switch to another method to finish it.

IV.B.2. *Method of Patches*

The photon density ϕ varies from 10^{-12} to almost 10^{14} over the course of the solution. To use a regular triangular mesh with sufficient resolution, we converted to a new variable $f : \phi = 10^f$, with $f \in [-12, 14]$ and $n \in [-1, 1]$. We used a uniform grid with 260 divisions along f and 10 along n. This mesh was adequate to return accurate solutions. No analysis was required, nor even the knowledge that the equations are stiff. Solution from our method is shown in Fig. 5. A parameter of the method of patches, the tolerance window outside of a patch within which the solution is required to be located (Subsection III.B), was 0.01. Neither the solution nor the required computer time varied much when this number was changed over a range of reasonable values.

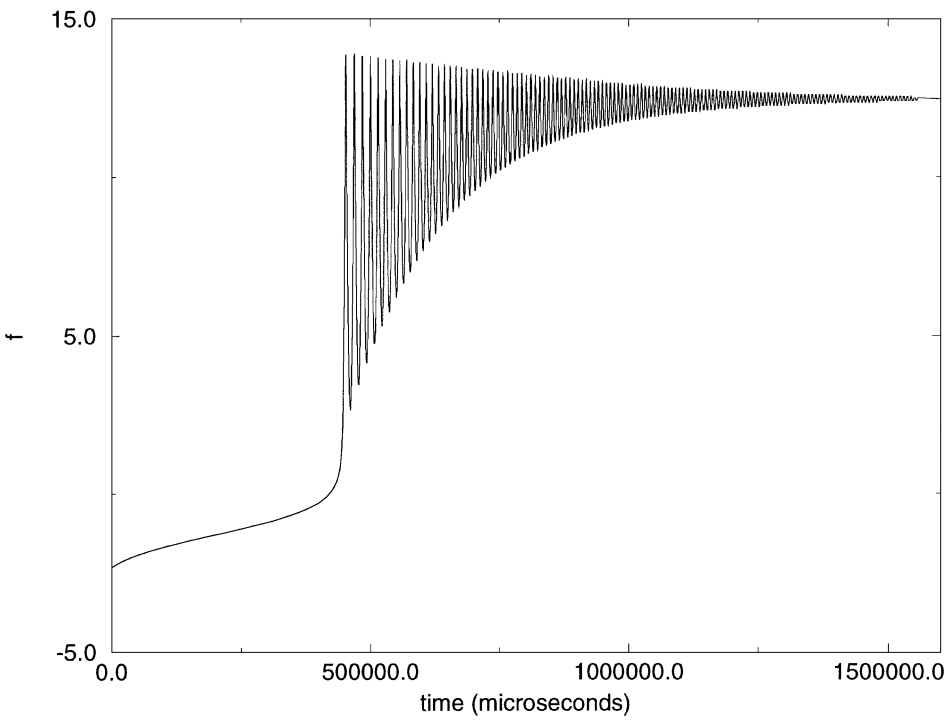


FIG. 5. Method of patches solution for laser equation.

IV.B.3. *Time Needed to Solve*

These equations are less demanding than the previous example. Both CVODE and MOP returned qualitatively accurate solutions in a roughly constant amount of cpu time for a wide range of tolerances. Using the same computer and compiler as reported in Subsection IV.A.2, CVODE with parameters and tolerances set as in [10, p. 46] (RTOL $= 1.0 \times 10^{-6}$, ATOL $= \{1.0 \times 10^{-9}, 1.0 \times 10^{-6}\}$) required 0.8 s to solve 16 s of model time. MOP required 0.1 s on the same machine. A plot of the derivative functions shows that they are very nonlinear only in a small part of the domain, so efficiency could be improved by using a lower resolution in most of the mesh, and a higher resolution for this region.

IV.B.4. *Discussion*

Note that Byrne and Hindmarsh [10, p. 46] used initial conditions $(-1, 0)$ while we used $(-0.8, 10^{-12})$. The initial condition in f is -12 (ϕ is 10^{-12}) because a finite regularly spaced mesh in log space cannot reach minus infinity. The initial condition in n is -0.8 because as n goes to -1, our solver code generates spurious solutions due to round-off error in calculating the matrix exponential. The numerically calculated solution at $t = 0$ disagrees by more than 1% with the initial condition, enough to cause the solver code to fail. This is a hint of problems that lurk in solving the general linear equation $\dot{x} = Ax + b$, where matrices A ill-conditioned for matrix exponentiation can lead to round-off errors. A more sophisticated computer code could better handle these extreme conditions.

V. SEARCHING FOR THE PERFECT FIT

There is considerable freedom available in choosing how to represent a nonlinear function $f(x)$ by a set of linear functions over patches. The simplest choice, which we call the *standard method*, uniquely determines the linearization by requiring it to agree with the original function on the nodes of each patch. However, this choice almost certainly does not minimize overall inaccuracy in the approximate solution of $\dot{x} = f(x)$. Finding the linearization that minimizes the error in the solution is an interesting and open question. In one variable, it is possible to increase the accuracy in the solution after crossing a single patch from $\mathcal{O}(\Delta x^3)$ to $\mathcal{O}(\Delta x^5)$. To do this, one can choose a linear approximation to f by finding a and b to minimize

$$\int_{x_L}^{x_L+\Delta x} (f(x) - l(x))^2 \, dx, \tag{15}$$

where $l(x) = ax + b$ and $\dot{x} = f(x)$. This only requires solving a linear equation. It gives an error in the solution of the differential equation, when $x = x_L + \Delta x$, of

$$\frac{f''(x_L)(6 f'(x_L)^2 - f(x) f''(x_L))}{720 f(x_L)^3} \Delta x^5 + \mathcal{O}(\Delta x^6), \tag{16}$$

where $'$ indicates derivative with respect to x. Note that this expression has only first and second derivatives of f, just as the $\mathcal{O}(\Delta x^3)$ result for standard linearizations, derived in Section VI. Also note that the standard error of $\mathcal{O}(\Delta x^3)$ comes from the maximum difference between the exact and linear functions being $\mathcal{O}(\Delta x^2)$ and the time to cross a patch being

$\mathcal{O}(\Delta x)$. But that analysis does not explain this fitted $\mathcal{O}(\Delta x^5)$ result, since the maximum difference between the exact and linear functions is still $\mathcal{O}(\Delta x^2)$ for the fitted case, and the time to cross the patch is still $\mathcal{O}(\Delta x)$. We get an extra two orders when the differential equation is solved. However, this fifth-order result holds only at the midpoint and end of the time interval used to cross the patch, as we explain below.

The time when the solution leaves the patch is t_{out}. For $x_L < x < x_L + \Delta x$, when t is t_{out}/n, $n \geq 1$, a time when the solution is on the interval, the error is usually third-order. For the standard case, the error at time t_{out}/n is

$$\frac{(2-3n)\, f''(x_L)}{12n^3 f(x_L)} \Delta x^3 + \mathcal{O}(\Delta x^4) \tag{17}$$

and for the fitted case, the error at time t_{out}/n is

$$\frac{(2-3n+n^2)\, f''(x_L)}{12n^3 f(x_L)} \Delta x^3 + \mathcal{O}(\Delta x^4), \tag{18}$$

except for $n = 1, 2$, when it is fifth-order.

Note that the slope and intercept of the linearization, a and b, can be expanded as power series in Δx. We can solve for the coefficients so as to zero out higher and higher order terms in the solution error after crossing one patch. A hierarchy of equations results, so that if we pick the first coefficient, the others are determined. For the logistic equation (1), this procedure to zero out the 0–4th order terms in the error with two coefficients each in the series for a and b (the 0th order term is free) returns two values of slope and intercept that zero all orders in the error. In other words, the exact and approximate solutions agree exactly at the boundary. This is not too surprising, since the logistic solution is known analytically and contains exponentials as does the linear solution. But it is an interesting example of the cancellation from the linear derivative being first too fast, then too slow, such that the two solutions agree exactly at the end of the interval.

One meaningful way to determine the hierarchy of equations mentioned in the previous paragraph is to demand that the slope and intercept of the approximation converge to the standard case in the limit $\Delta x \to 0$. This also implies that the approximation goes to the exact function in that limit. Since this hierarchy of equations is generally very complex, an interesting question is: Can we avoid it by finding some measure $\mu(x)$ and some functional $G(f, l)$ such that an equation involving $\int G(f(x), l(x))\mu\, dx$ returns an $l(x)$ that gives small error is the solution? We have shown that minimizing the distance gives fifth-order error for one variable, but that did not take into account the fact that the error in the function contributes to the solution error in a way that depends on x.

For multi-variable systems, this fifth-order result cannot hold in general for a fixed grid spacing Δx. This is because the solution must cross the entire interval in each variable to get the fifth-order result (Even more, we must yet prove the result for N variables if the solution *does* cross the entire interval.) However, one can imagine constructing a grid such that the solution always crosses the entire interval in each variable. If our fifth-order result holds for N coupled variables, then this hypothetical grid would give fifth-order solution error. Of course, our code only finds the edge approximately, so the numerical error would in fact be third-order. While our experience shows that the actual solution error is greatly reduced by fitting, we do not know how the error varies with the fraction of the cell crossed in each variable.

VI. ERROR ANALYSIS

We set a bound on the error in the solution, $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{x}_L(t)$, where $\mathbf{x}(t)$ is a vector of dependent variables that solve the ordinary differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$, and where $\mathbf{x}_L(t)$ is the time integrated solution of the piecewise linearization of $f(\mathbf{x})$, $\dot{\mathbf{x}}_L(t) = f_L(\mathbf{x})$.

The main results from this section are:

(1) The global error (error after crossing many patches) is $\mathcal{O}(h^2)$.

(2) The local error (error after crossing one patch) is $\mathcal{O}(h^3)$, unless there is a stable fixed point that traps the trajectory in the patch for all time, in which case the error is $\mathcal{O}(h^2)$,

where h is the maximum of the patch sizes in each variable, Δx_i.

VI.A. Maximum Error of Piecewise Linearization

In this section, we assume that the maximum difference between the linearization and the exact function scales as h^2. This is easily proven in one variable for linearizations which intersect the exact function twice on each patch.

In one variable, we linearize the exact derivative function $f(x)$ by specifying a uniform grid spacing h, which is the patch size, sampling the nonlinear functions on the grid points, and connecting these points to form a continuous, piecewise linear approximation to the nonlinear function. Following this procedure, we find that the error between the exact function $f(x)$ and the approximating function $f_L(x)$ is $\mathcal{O}(h^2)$, or more precisely

$$f(x) - f_L(x) \leq \frac{1}{2} f''(x) h^2 + \mathcal{O}(h^3), \tag{19}$$

where f'' is the second derivative with respect to x. Note that Eq. (19) implies that our approximate solution $\mathbf{x}_L(t)$ goes to the exact solution $\mathbf{x}(t)$ as h goes to zero.

VI.B. Global Error

We mean by the "global error" the difference between the exact and approximate solutions after crossing many patches. Using the bound on $f(\mathbf{x}) - f_L(\mathbf{x})$ that is appropriate for the N-variable linearization used, similar to Eq. (19), we can bound the error in the solution $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{x}_L(t)$ for a given time $t = T$, where $\mathbf{x}(t)$ is the exact solution of $\dot{\mathbf{x}} = f(\mathbf{x})$, and $\mathbf{x}_L(t)$ is the solution to the linearization $\dot{\mathbf{x}}_L = f_L(\mathbf{x}_L)$. In the following, we assume this bound is $\mathcal{O}(h^2)$. Expanding $f(\mathbf{x})$ in a power series in $\mathbf{x}_L(t)$,

$$f(\mathbf{x}(t)) = f(\mathbf{x}_L(t)) + \frac{\partial f(\mathbf{x}_L)}{\partial \mathbf{x}_L}(\mathbf{x}(t) - \mathbf{x}_L(t)) + \ldots, \tag{20}$$

and defining $\Delta f = f(\mathbf{x}_L) - f_L(\mathbf{x}_L)$, we have that

$$\frac{de_j}{dt} = [f(\mathbf{x}(t)) - f_L(\mathbf{x}_L(t))]_j = \Delta f_j + J_{jk} e_k + H, \tag{21}$$

where e_j is the jth component of $\mathbf{e}$, J is the Jacobian $\partial f_j(\mathbf{x}_L)/\partial x_{L_k}$, and H represents the higher order terms in the expansion. Then we have that

$$\frac{1}{2}\frac{d}{dt}|\mathbf{e}|^2 = \frac{1}{2} e_j \frac{de_j}{dt} = \frac{1}{2} e_j \Delta f_j + \frac{1}{2} e_j J_{jk} e_k + \frac{1}{2} e_j H_j, \tag{22}$$

where summations over j and k, respectively, are indicated. Taking the time derivative of $|\mathbf{e}|^2$,

$$\frac{1}{2}\frac{d}{dt}|\mathbf{e}|^2 = |\mathbf{e}|\frac{d}{dt}|\mathbf{e}| \leq \frac{1}{2}(|\mathbf{e}|\,|\Delta f| + |\mathbf{e}|^2|J| + |\mathbf{e}|\,|H|) \tag{23}$$

by the previous equation and the triangle inequality. Then dividing by $|\mathbf{e}|$,

$$\frac{d}{dt}|\mathbf{e}| \leq \frac{1}{2}(|\Delta f| + |J|\,|\mathbf{e}| + |H|). \tag{24}$$

Equation (19) shows that $|\Delta f| \leq Ch^2$, where C is a constant proportional to the maximum of the second derivatives, and h is the maximum of the Δx_j's. If f is Lipschitz on the domain of the solution $\mathbf{x}_L(t)$, *i.e.*, $|f(\mathbf{x})| \leq K|\mathbf{x}|$, then the triangle inequality on the Taylor expansion of f gives $|H(\mathbf{x}, t)| \leq K|\mathbf{x}|$ on this domain. Then since H is a function of $(\mathbf{e}, t)$,

$$\frac{d}{dt}|\mathbf{e}| \leq Ch^2 + (|J| + K)|\mathbf{e}|, \tag{25}$$

and by Gronwall's inequality,

$$|\mathbf{e}(t)| \leq Ch^2 \int_0^t e^{G(t-s)}\,ds, \qquad G(t) = \int_0^t (K + |J(r)|)\,dr. \tag{26}$$

This shows the error in the solution, $\mathbf{e}(t) = \mathcal{O}(h^2)$. $|J|$ is the matrix norm defined by $\|J\|_p = \sup_{\mathbf{x}\neq 0} \frac{\|J\mathbf{x}\|_p}{\|\mathbf{x}\|_p}$ [11].

VI.C. Local Error

We mean by the "local error" the difference between the exact and approximate solutions after crossing one patch. This error consists of two parts: the first comes from the way in which the difference between the exact and approximate derivative functions scales with patch size, and the second from the way in which the time to cross the patch varies with patch size.

VI.C.1. *Relation between Time to Cross Patch and Patch Size*

For each component j of the solution of the linearization, the distance travelled in variable j while crossing the patch is greater than or equal to the time spent in the patch, multiplied by the minimum of the absolute value of the j derivative on the patch. This is expressed by $|\Delta x_j| \geq \Delta t|(\dot{x}_L)_j(t)|_{\min}$, and gives for each variable j an expression $\Delta t_j \leq |\Delta x_j|/|(\dot{x}_L)_j(t)|_{\min}$. Since $\Delta x_j \leq h$, where h is the maximum of the patch sizes in each variable, we can write that

$$\Delta t_j \leq h/|(\dot{x}_L)_j(t)|_{\min}. \tag{27}$$

The largest of these Δt_j bounds Δt, the time spent in the patch. This seems problematic if the derivative changes sign or is everywhere zero, since then Δt_j is infinite. But if some of the Δt_j are infinite, then the largest finite one is the upper bound on Δt. If *all* of the Δt_j are infinite, then this expression cannot bound Δt. A fixed point is in the patch, which may be

attracting for all points in the patch. In that case Δt is infinite, because the solution never leaves the patch. Then Δt cannot be shown to be proportional to h. However, at most one patch, which would be the last, in a given solution, has this property.

To summarize, we conclude that

$$\Delta t = \mathcal{O}(h), \tag{28}$$

where h is the maximum of the patch sizes in each variable Δx_j, unless there is a stable fixed point that captures the solution in the patch.

VI.C.2. *Error between Exact and Approximate Solutions after Crossing One Patch*

The error between the exact and approximate solution is $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{x}_L(t)$. Expanding $f(\mathbf{x})$ in a power series in $\mathbf{x}_L$, we have that

$$\frac{de_j}{dt} = f(\mathbf{x}_L) - f_L(\mathbf{x}_L) + J_{jk}e_k + H_j(\mathbf{e}, t), \tag{29}$$

where e_j is the jth component of $\mathbf{e}$, J is the Jacobian, and H represents the higher order terms in the expansion. Assuming $\mathbf{e}(0) = 0$, the error after crossing one patch is

$$e_j(\Delta t) = \int_0^{\Delta t} (f(\mathbf{x}_L) - f_L(\mathbf{x}_L))_j + J_{jk}e_k + H_j(\mathbf{e}, t))\, dt \approx (f(\mathbf{x}_L) - f_L(\mathbf{x}_L))_j \Delta t \tag{30}$$

as $\Delta t \to 0$. We showed in the previous section that $\Delta t = \mathcal{O}(h)$, so

$$\lim_{\Delta x \to 0} \mathbf{e}(\Delta t) = \lim_{\Delta x \to 0} (f(x_L) - f_L(x_L))\Delta t = \mathcal{O}(h^2)\mathcal{O}(h) = \mathcal{O}(h^3), \tag{31}$$

since we showed above that $f(\mathbf{x}_L) - f_L(\mathbf{x}_L) = \mathcal{O}(h^2)$ and that $\Delta t = \mathcal{O}(h)$ as $\Delta x \to 0$. So the error across one patch is $\mathcal{O}(h^3)$ unless there is a stable fixed point in the patch that captures the solution for all time. In that case the above analysis gives that the error is $\mathcal{O}(h^2)$, since $f(\mathbf{x}_L) - f_L(\mathbf{x}_L)$ is still $\mathcal{O}(h^2)$ but Δt cannot be related to h in the patch that contains the fixed point.

VII. DISCUSSION

The method of patches can be applied to many types of equations, including non-autonomous, higher-order, and partial differential equations, as well as boundary-value problems using shooting methods (see [12] for exact solutions to nonlinear problems). Mathematica [13] has been very useful in generating solutions to nonlinear ODEs, in code development, and in analysis.

In the two variable examples in Section IV, our method yielded faster solutions to ordinary differential equations than several well-known solvers. However, this work was originally motivated by our desire to integrate the partial differential equation $\dot{\mathbf{u}} = D\nabla^2\mathbf{u} + f(\mathbf{u})$ with $f(\mathbf{u})$ given by the Gaspar–Showalter model, Eq. (13), and D a diagonal matrix of diffusion coefficients. If we use a finite-difference approximation to represent the spatial derivative operator, a straightforward approach is then a split-step method, solving the diffusion equation exactly and using our ODE code for the reaction part. Another approach is to incorporate directly into the method the terms arising from the finite-difference diffusion

operator. This can be done by noticing that, in the ODEs that result from finite-differencing, the derivative terms are linear and so can be absorbed into our linearization. This gives a set of coupled linear ordinary differential equations that can in principle be solved with the method. However, this set of equations is now an N variable system where N is the number of spatial points times the number of dependent variables. In practice, we will solve these equations by breaking the matrix into more manageable pieces. Our working reaction-diffusion code approximates spatial neighbors by their value at the beginning of a time step. Some other strategies to approximate derivative terms are mentioned in the next section.

VIII. REVIEW OF THE LITERATURE

A careful search has not revealed precisely this method published elsewhere. However, we have found many similar ideas, and a large literature on calculating the matrix exponential, which remains a challenging problem due to round-off error and matrices ill-conditioned for exponentiation. Using the matrix exponential to solve initial value differential equations. Carroll [14] exponentiates the Jacobian. A long-standing work by Pavlov *et al.* [15, 16] retains nonlinear terms of the original equations. Applying the matrix exponential to solving PDEs, Graziani [17] solves linear and radiation diffusion problems using matrix decomposition, furthering the work of Richardson *et al.* [18]. Zhong *et al.* [19] solve PDEs by breaking the spatial domain into sub-domains and approximating terms outside the sub-domain as constant. The method of patches can be combined with these last two schemes to integrate nonlinear PDEs.

IX. CONCLUSION

This paper is a proof-of-principle demonstration of the basic ideas of the method of patches. A few rudimentary improvements to make it faster and more accurate than the code discussed in this paper have yielded solutions on the order of one hundred times faster for the same accuracy. Much more work is needed on accurate and efficient solution of the N-variable problem, generating accurate and valid approximations and solutions, and better methods for partial differential equations, to name a few areas. To become more efficient, the method must change patch size adaptively. Manipulating the linearization to preserve geometric features such as fixed points and null-clines may allow even larger patches while still returning qualitatively correct solutions. Since the method can solve stiff equations without resorting to implicit techniques, it is especially promising for solving stiff partial differential equations on parallel computers.

ACKNOWLEDGMENTS

We thank many of our colleagues for useful discussion and help, especially Ade Lee for help with initial coding efforts and suggestions, John Cogdell for editing, and also James Hyman and Aric Hagberg for their input.

REFERENCES

1. H. C. Yee, P. K. Sweby, and D. F. Griffiths, *J. Comput. Phys.* **97**, 249 (1991).
2. M. H. Schultz, *Spline Analysis* (Prentice Hall, New Jersey, 1973).
3. H. C. Yee and P. K. Sweby, *Nonlinear Dynamics and Numerical Uncertainties in CFD*, Technical Report TM-110398, NASA, 1996.

4. E. Sacks, Automatic qualitative analysis of dynamic systems using piecewise linear approximations, *Artificial Intelligence* **41**, 313 (1990).
5. M. Hirsch and S. Smale, *Differential Equations, Dynamical Systems, and Linear Algebra* (Academic Press, Orlando, FL, 1974).
6. C. B. Moler and C. Van Loan, Nineteen dubious ways to calculate the matrix exponential, *SIAM Rev.* **20**, 801 (1978).
7. E. J. Putzer, Avoiding the Jordan canonical form in the discussion of linear systems with constant coefficients, *Amer. Math. Monthly* **73**, 2 (1996).
8. S. D. Cohen and A. C. Hindmarsh, CVODE, A stiff/nonstiff ODE solver in C, *Comput. Phys.* **10**, 138 (1996).
9. V. Gaspar and K. Showalter, A simple model for the oscillatory iodate oxidation of sulfite and ferrocyanide, *J. Phys. Chem.* **94**, 4973 (1990).
10. G. Byrne and A. Hindmarsh, Stiff ODE solvers: A review of current and coming attractions, *J. Comput. Phys.* **70**, 1 (1987).
11. G. H. Golub and C. F. V. Loan, *Matrix Computations* (Johns Hopkins Press, Baltimore, 1989).
12. A. D. Polyanin and V. F. Zaitsev, *Handbook of Exact Solutions for Ordinary Differential Equations* (CRC Press, Boca Raton, FL, 1995).
13. S. Wolfram, *Mathematica* (Addison–Wesley, Redwood City, CA, 1991).
14. J. Carroll, A matricial exponentially fitted scheme for the numerical solution of stiff initial-value problems, *Comput. Math. Appl.* **26**, 57 (1993).
15. B. V. Pavlov and O. E. Rodionova, The method of local linearization in the numerical solution of stiff systems of ordinary differential equations, *Comput. Math. Math. Phys.* **27**, 30 (1987).
16. B. V. Pavlov and O. E. Rodionova, Numerical solution of systems of linear ordinary differential equations with constant coefficients, *Comput. Math. Math. Phys.* **34**, 535 (1994).
17. F. R. Graziani, The product formula algorithm applied to linear and radiation diffusion, *J. Comput. Phys.* **118**, 9 (1995).
18. J. L. Richardson, R. C. Ferrell, and L. N. Long, Unconditionally stable explicit algorithms for nonlinear fluid dynamics problems, *J. Comput. Phys.* **104**, 69 (1993).
19. W. Zhong, J. Zhu, and Z. Xiang-Xiang, On a new time integration method for solving time dependent partial differential equations, *Comput. Methods Appl. Mech. Eng.* **130**, 163 (1996).