Pergamon

Computers Math. Applic. Vol. 35, No. 10, pp. 45–53, 1998
© 1998 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0898-1221/98 $19.00 + 0.00

PII: S0898-1221(98)00071-6

Linear Conservation Laws for ODEs

L. F. SHAMPINE
Mathematics Department
Southern Methodist University
Dallas, TX 75275, U.S.A.
shampine@na-net.ornl.gov

(Received October 1997; accepted November 1997)

Abstract—Some physical systems described by ODEs have quantities that are conserved as the system evolves. Runge-Kutta formulas and linear multistep methods preserve linear conservation laws automatically. A code for the integration of ODEs involves a number of algorithms in addition to the basic formula. It is shown here that popular codes preserve linear conservation laws if they are used properly. An application is made to differential-algebraic equations of index 2 arising in the solution of the Navier-Stokes equation by the method of lines. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords—Conservation laws, Runge-Kutta, Linear multistep method, Adams, BDF.

1. INTRODUCTION

If the differential equation $\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$ is such that $\mathbf{c}^T\mathbf{f}(x, \mathbf{y}) = 0$ for a constant vector $\mathbf{c}$ and all $(x, \mathbf{y})$, then the solution with initial value $\mathbf{y}(a) = \mathbf{A}$ satisfies the linear conservation law $\mathbf{c}^T\mathbf{y}(x) = \mathbf{c}^T\mathbf{A}$, for all x. Such laws express physical properties such as conservation of mass and conservation of charge. A given equation might satisfy a number of such laws. It is natural to monitor how well the numerical solution of an initial value problem satisfies conservation laws as a measure of the accuracy of the solution. However, Robertson and McCann [1] showed that Runge-Kutta and linear multistep formulas preserve linear conservation laws *automatically.* (Nonlinear conservation laws are also important, but the considerations are different because with very few exceptions they are not preserved automatically. For this reason, we consider here only linear conservation laws and refer the reader to [2,3] as entries to the literature on nonlinear laws.) Later Rosenbaum [4] considered other formulas for the solution of initial value problems for ODEs and identified some that do not preserve linear conservation laws. He also considered the practical matter of evaluating implicit formulas. Among the schemes that he studied is the one most commonly used for stiff problems, a simplified Newton (chord) method. He showed that if an analytical expression for the Jacobian $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ is used and if the predicted value used to start the iteration satisfies the conservation laws, then for linear multistep methods, all iterates of a simplified Newton method satisfy the laws and in particular, the value used to advance the integration satisfies the laws. On the other hand, he showed by example that this might not be true for implicit Runge-Kutta methods evaluated by Newton's method.

A production-grade code for the numerical solution of initial value problems involves a good many algorithms in addition to the underlying Runge-Kutta or linear multistep formula. All these algorithms must preserve the conservation laws if the results produced by the code are to satisfy the laws. For instance, often codes do not obtain the results that they report to the user by stepping to an output point with the underlying formula, rather they step past and obtain results by interpolation (continuous extension). Further, the codes vary the step size, and possibly the order of the formula, and it is not obvious that the way this is done will preserve conservation laws. In his analysis of the evaluation of implicit formulas, Rosenbaum analyzed only the case of an analytical Jacobian and it is far more common in practice that Jacobians are approximated by finite differences.

Here, we show that the most widely used codes do preserve linear conservation laws when used properly. Despite the point made by Rosenbaum about implicit Runge-Kutta formulas, we find that popular codes based on such methods do preserve conservation laws when used properly. What does "used properly" mean here? It is often observed that the solution of some stiff problems is much cheaper if the Jacobian is approximated by a diagonal matrix or more generally by a band matrix of narrow width. Popular codes make this easy, but we find that linear conservation laws are *not* preserved when problems are solved this way. These theoretical results are illustrated by numerical experiments with many of the most popular methods as implemented in widely used codes. An extended application of the analysis is made to the approximate solution of the Navier-Stokes equations by the method of lines [5]. The approach results in a differential-algebraic equation (DAE) of index 2: the approximations to the pressure are algebraic variables and the discrete incompressibility condition is a system of algebraic equations. It is suggested in [6] that the system be reduced in standard fashion to a DAE of index 1. In this new system the incompressibility condition becomes a set of linear conservation laws and it is suggested that for many numerical methods, these laws will be satisfied automatically. This is not obvious and it appears that whether it is true depends on how the DAE is solved. Here, we discuss ways of proceeding for which we can show that the laws are satisfied automatically.

2. ANALYSIS

The details required here about the methods and their implementations can be found in general texts with an emphasis on computation like [7–10]; papers and other documentation for the codes cited; and comments in the codes themselves. All the results we need to analyze production codes are easy consequences of the form of the equations that arise and a simple, common argument. To illustrate the argument, suppose that we have approximated the solution $\mathbf{y}(x)$ of an initial value problem at mesh points $\{x_k\}$ by values $\{\mathbf{y}_k\}$ that all satisfy a linear conservation law. We compute a new value $\mathbf{y}_{j+1} \approx \mathbf{y}(x_{j+1})$ at $x_{j+1} = x_j + h$ by a recipe of the form

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h \sum \beta_k \mathbf{f}_k,$$

a form that includes all Runge-Kutta formulas. Here, the $\mathbf{f}_k$ are values of $\mathbf{f}(x, \mathbf{y})$ at arguments that we do not need to specify. The new value satisfies the conservation law because

$$\mathbf{c}^\top \mathbf{y}_{j+1} = \mathbf{c}^\top \mathbf{y}_j + h \sum \beta_k \mathbf{c}^\top \mathbf{f}_k = \mathbf{c}^\top \mathbf{A}$$

follows from the facts that $\mathbf{c}^\top \mathbf{f}_k = 0$ no matter the argument of $\mathbf{f}$ and $\mathbf{c}^\top \mathbf{y}_j = \mathbf{c}^\top \mathbf{A}$ by assumption. This is the classic result of Robertson and McCann. It applies to implicit Runge-Kutta methods when it is assumed that the formula is evaluated exactly. This is not a realistic assumption, so we take up later the practical evaluation of implicit Runge-Kutta formulas. The step size need not be constant here, so this result is enough to demonstrate that linear conservation laws are preserved by codes based on explicit Runge-Kutta methods that obtain output by stepping to the output point. The widely-used code RKSUITE [11] implements three pairs of explicit Runge-Kutta formulas. When its highest order pair, the (7,8) pair, is specified, the code steps to output points. However, at lower orders modern codes select an efficiently large step size and produce output by means of a continuous extension. RKSUITE does this with both its (2,3) and (4,5) pairs. Two approaches are taken to the development of continuous extensions, interpolation, and a family of formulas, but the form of the computational scheme is the same in either approach. After a step to x_{j+1} an approximate solution is obtained at $x_j + \sigma h$ by a recipe of the form

$$\mathbf{y}_{j+\sigma} = \mathbf{y}_j + \sigma h \sum \beta_k^*(\sigma)\mathbf{f}_k.$$

For the sake of efficiency most of the $\mathbf{f}_k$ here are those formed in the step to x_{j+1}, but such matters do not concern us here. The form of the recipe allows us to conclude in the same way as for the basic formula that linear conservation laws are satisfied by these intermediate values, too. With this observation we find that any modern explicit Runge-Kutta code will preserve all linear conservation laws, and in particular that RKSUITE will do so no matter which pair is used.

The form stated for Runge-Kutta methods applies to Adams methods, too. The distinction is that the coefficients depend on the mesh and the arguments of $\mathbf{f}$ come from approximate solutions at points prior to x_j rather than points in the span of the current step. Of course this does not affect the argument, so if the integration is started with values that satisfy the laws, then the explicit Adams-Bashforth formulas preserve conservation laws and implicit Adams-Moulton methods do, too, if they are evaluated exactly. One way such formulas are used is as a predictor-corrector pair. An explicit Adams-Bashforth formula is used to form a tentative, "predicted", value $\mathbf{p}_{j+1} \approx \mathbf{y}(x_{j+1})$. This value is substituted for the unknown $\mathbf{y}_{j+1}$ in the right-hand side of an implicit Adams-Moulton formula to obtain a "corrected" value. If this corrected value is used to advance the integration, the procedure amounts to an explicit recipe for $\mathbf{y}_{j+1}$. Because the formula for the accepted value is of the same form, the same argument shows that such schemes preserve linear conservation laws. This is also true for the evaluation of the implicit Adams-Moulton formula by successive substitution. The process just described is repeated with the new value replacing the old in the right-hand side of the formula until it is judged that the implicit formula is satisfied to an acceptable accuracy. The distinction is that in a predictor-corrector implementation, a fixed number of corrections is made and in an implicit implementation, corrections are done until the implicit formula is evaluated to a fixed accuracy.

All production-grade codes based on Adams methods vary the step size, but there are two distinct ways this is done. The fully variable step size implementation of ODE/STEP,INTRP [12] computes the coefficients that correspond to the actual mesh. Because the argument does not require that the same formula be used at all steps, we find that such codes preserve conservation laws. Later we take up the other way of varying the step size. The production-grade codes also vary the order. Again it matters how this is done, but for fully variable step size implementations, this is just a different formula of the same form and the argument applies. Variation of step size and order is used to get the "previously computed" approximations required by Adams methods. The codes start at order 1 with what amounts to an explicit Runge-Kutta formula and rapidly increase the step size and order until values appropriate to the task are found. Accordingly, in a fully variable step size implementation, linear conservation laws are preserved at every step. Adams methods are based on polynomial interpolants, so all the production-grade codes select an efficient step size and obtain output by evaluating an interpolant. This interpolant has the same origin as the formulas themselves, so has the same form, and just as with Runge-Kutta methods, the intermediate values preserve all conservation laws. We now have all the results required to show that ODE/STEP,INTRP preserves all linear conservation laws. The way it varies the step size and order does not interfere with preservation of the laws and in particular, the way the code starts preserves the laws. Output is obtained by interpolation, which also preserves linear conservation laws.

The widely used code VODE [13] is the latest in a family that originated with Gear's DIFSUB [14]. Like all members of the family, both Adams-Moulton methods and BDF, backward differentiation formulas, are implemented. The Adams-Moulton methods are evaluated by repeated corrections in the manner described earlier and other aspects of the implementation are like those of ODE/STEP,INTRP except for the change of step size and order. The Nordsieck (Taylor series) representation of the formulas obscures the matter, but DIFSUB integrates with constant step size formulas. When it is deemed advisable to change the step size and/or order, fictitious approximate solution values are obtained at the new step size by interpolation and they are used to advance the integration with the new step size. The interpolated values are formed as a linear combination of previously computed approximations to the solution and its slope. The general form is

$$\mathbf{y}_\sigma = \sum \alpha_k \mathbf{y}_k + h \sum \beta_k \mathbf{f}_k.$$

The consistency condition $\sum \alpha_k = 1$ leads immediately to the conclusion that all linear conservation laws are preserved. One of the differences between VODE and DIFSUB is that VODE uses a fixed leading coefficient implementation of its formulas. This means that the way step sizes are changed is intermediate between what is done in DIFSUB and what is done in ODE/STEP,INTRP. For our purposes this distinction is unimportant because the step size is changed by interpolation and the form of the interpolant is the same. With these observations we conclude that popular codes based on Adams-Moulton methods with fixed coefficient (constant step size formula) or fixed leading coefficient implementations like DIFSUB, VODE, and other members of the Gear family of codes preserve all linear conservation laws.

The BDFs are the most popular formulas for the solution of stiff problems. For our purposes, as they are implemented in the Gear family of codes, they differ from the Adams-Moulton formulas only in the way that these implicit formulas are evaluated. The general form of the algebraic equation for $\mathbf{y}_{j+1}$ is

$$\mathbf{y} = h\gamma \mathbf{f}(\mathbf{y}) + \psi.$$

The constant γ depends on the formula and h is the step size. The independent variable x_{j+1} is not displayed in $\mathbf{f}$ because it is fixed and other quantities fixed in the computation are combined in ψ. All we need to know about ψ is that

$$\mathbf{c}^\top \mathbf{y} = h\gamma \mathbf{c}^\top \mathbf{f}(\mathbf{y}) + \mathbf{c}^\top \psi = c^\top \psi = c^\top \mathbf{A},$$

because an exact evaluation of the formula preserves linear conservation laws. All the popular codes for solving stiff problems solve such equations by linearizing $\mathbf{f}$ about a current iterate:

$$\mathbf{f}(\mathbf{y}) \approx \mathbf{f}\left(\mathbf{y}^m\right) + J\left(\mathbf{y} - \mathbf{y}^m\right).$$

Here, J is an approximation to the Jacobian $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$. The linearization and a little manipulation leads to a system of linear equations for the change in the current iterate:

$$(I - h\gamma J)\left(\mathbf{y}^{m+1} - \mathbf{y}^m\right) = h\gamma \mathbf{f}\left(\mathbf{y}^m\right) + \psi - \mathbf{y}^m.$$

Rosenbaum studied this iteration when $J = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ and the predicted solution $\mathbf{y}^0$ satisfies the conservation law. Taking the partial derivative of $\mathbf{c}^\top \mathbf{f}(x, \mathbf{y}) = 0$ shows that $\mathbf{c}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \mathbf{0}$. Then

$$\mathbf{c}^\top (I - h\gamma J)\left(\mathbf{y}^{m+1} - \mathbf{y}^m\right) = \mathbf{c}^\top \mathbf{y}^{m+1} - \mathbf{c}^\top \mathbf{y}^m,$$

and

$$\mathbf{c}^\top \left[h\gamma \mathbf{f}\left(\mathbf{y}^m\right) + \psi - \mathbf{y}^m\right] = \mathbf{c}^\top \psi - \mathbf{c}^\top \mathbf{y}^m,$$

so

$$\mathbf{c}^\top \mathbf{y}^{m+1} = \mathbf{c}^\top \psi = c^\top \mathbf{A}.$$

Every iterate satisfies the conservation laws, so the one accepted as $\mathbf{y}_{j+1}$ does, too. We see that Rosenbaum's assumption that the predicted value satisfy the conservation laws is not needed for this class of methods when an analytical Jacobian is available and the formula is evaluated in this way.

Stiff problems are often solved with a finite difference approximation J to the Jacobian because this is so much less trouble for users. The question is whether this approximation satisfies the fundamental relation $\mathbf{c}^{\mathsf{T}} J = \mathbf{0}$ satisfied by the analytical Jacobian. For a general matrix, J is formed a column at a time by a difference quotient, so let us write $J = [J_1, \ldots, J_n]$. There are many practical details, but invariably J_r is the difference of two function values divided by an increment: $J_r = (\mathbf{f}^* - \mathbf{f})/\delta$. When this is appreciated, it is obvious that $\mathbf{c}^{\mathsf{T}} J_r = 0$ for each r, hence that the numerical Jacobian does satisfy the fundamental relation and all iterates of the simplified Newton method satisfy all linear conservation laws.

Curtis *et al.* [15] pointed out that the sparsity pattern of the Jacobian could be exploited to reduce the number of evaluations of $\mathbf{f}$ in forming J. The idea is that if known zero entries in two columns match up properly, it is possible to approximate all the nonzero elements in both columns simultaneously. In particular, it is possible to form an approximation to a banded Jacobian with a number of function evaluations that depends only on the width of the band, not the number of columns. This reduction in cost is so important that all the recent members of the Gear family of codes allow users to specify a banded matrix by means of the number of diagonals below and above the main diagonal. There are codes like **ode15s** [16] that allow users to specify the precise sparsity pattern. Often it is found that some entries in the Jacobian matrix are very small relative to others. Or, it might be found that the elements near the main diagonal dominate strongly. The sole purpose of the matrix J is to make the iteration converge with acceptable speed. With this in mind, it is natural to set small elements to zero or to approximate the Jacobian by a banded matrix in order to reduce the cost of forming, storing, and solving the linear systems that arise in evaluating the formula. This might be done in any code that allows for the numerical approximation of a banded Jacobian simply by giving the code parameters specifying a band thought to contain all the important elements. VODE specifically provides for the extreme case of a diagonal approximation to the Jacobian. In this the Jacobian is approximated using only two function evaluations $\mathbf{f}^*$, $\mathbf{f}$, and an increment δ; for each i, $J_{ii} = (f_i^* - f_i)/\delta$ and all other entries of J are zero. Thinning out the Jacobian in this way may be advantageous numerically, but it obviously destroys the fundamental relation needed for preservation of linear conservation laws and generally they will not be preserved. A person solving a system for which certain physical quantities are conserved needs to appreciate the consequences of economizing in this way.

Section IV.8 of [9] discusses the "Implementation of Implicit Runge-Kutta Methods" in general and documents what is done in the important code RADAU5 in particular. A step from an approximation y_0 at x_0 to an approximation y_1 at $x_1 = x_0 + h$ with an implicit Runge-Kutta method of s stages involves the solution of a system of nonlinear algebraic equations of the form

$$Z = \begin{pmatrix} z_1 \\ \vdots \\ z_s \end{pmatrix} = A \begin{pmatrix} hf(x_0 + c_1 h, z_1) \\ \vdots \\ hf(x_0 + c_s h, z_s) \end{pmatrix}.$$

Here, the coefficients $A = (a_{ij})$ and the c_i are characteristic of the method. Then

$$y_1 = y_0 + \sum_{i=1}^{s} d_i z_i.$$

The equations for Z are linearized just as with the BDFs and all the Jacobians are approximated by the same matrix J. This results in an iteration matrix

$$\begin{pmatrix} I - ha_{11}J & \cdots & -ha_{1s}J \\ \vdots & & \vdots \\ -ha_{s1}J & \cdots & I - ha_{ss}J \end{pmatrix}$$

for computing a correction to the current approximation Z^k. The actual computation of the correction makes use of transformations due to Bickart and Butcher that reduce the cost substantially and there are a number of important practical details, but this does not matter for our purposes because the form used is mathematically equivalent to this one. As with the BDFs, it is seen easily from the form and $\mathbf{c}^T J = 0$ that the intermediate values and y_1 satisfy linear conservation laws if the predicted values do. Hairer and Wanner suggest using a continuous extension from the preceding step (interpolation) to predict these values and as we have seen before, these values satisfy linear conservation laws if the values of the preceding step do. Accordingly, implicit Runge-Kutta methods implemented in the manner of RADAU5 preserve linear conservation laws. Like VODE, this code provides for the efficient computation of banded Jacobians with differences. And, just as with VODE, setting entries to zero by specifying a smaller band width to reduce the cost will generally lead to results that do not satisfy linear conservation laws.

3. NUMERICAL EXPERIMENTS

Both the equations that we solve here are such that the sum of the solution components is constant and with the initial conditions specified, the constant is one. We test preservation of this law by computing the difference between the sum of the components and the proper value and finding the maximum of this difference over all the specified output points. By specifying output points, we test the effect of interpolation in those codes which use it to get output. The computations were done using double precision versions of the codes and a PC with IEEE arithmetic. If the law is preserved, the maximum error should be no more than a small multiple of the unit roundoff, which is about 2.2×10^{-16}.

The collection of nonstiff test problems [17] includes a small system describing a nonlinear chemical reaction

$$\begin{aligned} y_1' &= -y_1, \\ y_2' &= y_1 - y_2^2, \\ y_3' &= y_2^2, \end{aligned}$$

that is to be solved on $[0, 20]$ with initial values $\mathbf{y}(0) = (1, 0, 0)^T$. Although the tolerances and other computational details are not relevant to testing preservation of the conservation law, we comment that we used a relative error tolerance of 10^{-4} and an absolute error tolerance (or threshold, depending on the code) on each component of 10^{-8}. First, we solved the problem with the $(7, 8)$ pair of RKSUITE. This is an explicit Runge-Kutta method that produces output by stepping to the output point. The maximum error was only 1.1×10^{-16}; this is smaller than a unit roundoff because the PC has a number of guard digits, even in double precision. This illustrates the classic Robertson-McCann result for explicit Runge-Kutta formulas. When the problem is solved with the $(4, 5)$ pair in RKSUITE, output is obtained by interpolation. This run resulted in a maximum difference of about 2.2×10^{-16}. Evidently, interpolation is preserving the conservation laws. VODE makes available two kinds of methods. For the solution of a nonstiff problem like this one, VODE is appropriately used as a variable step size, variable order implementation of the Adams-Moulton formulas. Step size and order are varied by interpolation and output is obtained by interpolation. The implicit formulas are evaluated by successive substitution. The maximum error in satisfying the conservation law was about 5.6×10^{-16}, confirming our analysis.

The Robertson problem,

$$\begin{aligned} y_1' &= -0.04 y_1 + 10^4 y_2 y_3, \\ y_2' &= +0.04 y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2, \\ y_3' &= 3 \times 10^7 y_2^2, \end{aligned}$$

is a classic test problem for codes that solve stiff ODEs. We integrate over $[0, 1000]$ with initial values $\mathbf{y}(0) = (1, 0, 0)^\mathsf{T}$. The problem appears in the prolog to VODE as an example of the use of the BDFs for the solution of a stiff problem. As in the example, we used a relative error tolerance of 10^{-4} and absolute error tolerances of 10^{-8}, 10^{-14}, and 10^{-6}, respectively. In the experiments reported here, we used numerical Jacobians and specified output at the integers. When instructed to use an approximation to the full Jacobian, VODE produced results for which the maximum error was about 4.4×10^{-16}. This confirms our theoretical results that a variable step size, variable order BDF code with interpolatory change of step size and order, output by interpolation, and numerical Jacobians formed by differences preserves linear conservation laws. On the other hand, when instructed to use a diagonal approximation to the Jacobian, the maximum error was about 2.0×10^{-2}. Clearly, economizing in this way has resulted in a solution that does not satisfy the conservation law, as predicted.

We solved the Robertson problem in the same way with the implicit Runge-Kutta code RADAU5. The $(3, 1)$ entry in the Jacobian is zero, so instructing the code that the Jacobian is a banded matrix with one diagonal below the main diagonal and two above accounts properly for the structure of the Jacobian. When instructed to use numerical Jacobians, the maximum error was about 4.4×10^{-16}. This confirms our theoretical results that an implicit Runge-Kutta method evaluated properly with numerical Jacobians and with output obtained by a continuous extension does preserve linear conservation laws. On the other hand, when instructed to use a tridiagonal approximation to the Jacobian, the maximum discrepancy was about 9.8×10^{-7}. This is far greater than a unit roundoff and shows once again that setting nonzero entries in the Jacobian to zero in the approximation J results in a solution that does not satisfy the conservation law.

4. AN APPLICATION

Gresho *et al.* [5] discuss solution by the method of lines of the equations modelling the flow of an incompressible, viscous fluid. Semidiscretization of the Navier-Stokes equation gives rise to a DAE of index 2. We follow the discussion of this DAE by Brenan *et al.* [6, p. 181]. The equations are

$$M\mathbf{U}' + (K + N(\mathbf{U}))\mathbf{U} + C\mathbf{P} = \mathbf{f}(\mathbf{U}, \mathbf{P}), \tag{1}$$
$$C^\mathsf{T}\mathbf{U} = \mathbf{0}. \tag{2}$$

Here, $\mathbf{U}(t)$ is a vector approximating the velocity on a fixed spatial grid and $\mathbf{P}(t)$ is a similar approximation to the pressure. Using finite differences to discretize the spatial derivatives it is possible to obtain a mass matrix M that is the identity and using finite elements it is possible to obtain a mass matrix that is symmetric, positive definite. Here, we consider the case $M = I$ for the sake of simplicity only. The matrix C represents a discrete approximation to the gradient operator. The algebraic equations (2) represent the incompressibility condition because they require that a discrete divergence of the velocity be zero. The function $\mathbf{f}(\mathbf{U}, \mathbf{P})$ arises from the boundary conditions. We assume that it does not depend on $\mathbf{P}$, an important assumption that we discuss further below. With this assumption and the assumption that the columns of C are linearly independent, the DAE is of index 2. Indeed this follows easily from the reduction used for its solution in [6]. Differentiation of (2), substitution of $\mathbf{U}'$ from (1) into the result, and a little manipulation yields

$$\mathbf{U}' = \mathbf{f}(\mathbf{U}) - (K + N(\mathbf{U}))\mathbf{U} - C\mathbf{P}, \tag{3}$$
$$C^\mathsf{T}C\mathbf{P} = C^\mathsf{T}(\mathbf{f}(\mathbf{U}) - (K + N(\mathbf{U}))\mathbf{U}). \tag{4}$$

This is a DAE of index 1. To see this, we observe that the assumption on C implies that $C^\mathsf{T}C$ is nonsingular, so we can solve (4) for $\mathbf{P}$ and eliminate it from (3). The resulting ODE for $\mathbf{U}$ is called the underlying ODE and we write it as $\mathbf{U}' = \mathbf{F}(t, \mathbf{U})$. The incompressibility condition (2) represents a set of conservation laws for the underlying ODE because

$$C^\top \mathbf{F}(t, \mathbf{U}) = C^\top (\mathbf{f}(\mathbf{U}) - (K + N(\mathbf{U}))\mathbf{U}) - C^\top C\mathbf{P} = \mathbf{0}.$$

Generally numerical methods for DAEs do not preserve hidden constraints like (2) when solving (3,4). It is suggested by Brenan *et al.* that the present situation is special because the constraints are linear conservation laws and many methods preserve them automatically. Here, we investigate the matter briefly as an application of our theoretical results. Because the algebraic equations (4) are linear and the matrix $C^\top C$ is constant, the reduction used to show the DAE is of index 2 is a practical way to solve it. Any of the popular numerical methods might be applied to the underlying ODE, including explicit ones. Each time that it is necessary to evaluate $\mathbf{F}(t, \mathbf{U})$, the linear system (4) is first solved for the $\mathbf{P}$ corresponding to $(t, \mathbf{U})$. This $\mathbf{P}$ is then used in evaluating $\mathbf{F}(t, \mathbf{U}) = \mathbf{f}(\mathbf{U}) - (K + N(\mathbf{U}))\mathbf{U} - C\mathbf{P}$. Solution of the DAE in this manner is just the solution of an ODE with a function $\mathbf{F}$ that is a little complicated to evaluate and the results established earlier show that all the popular numerical methods preserve the linear conservation laws (2), provided that the initial values satisfy the laws.

LSODI [18] is a BDF code in the Gear family that can solve ODEs of the form $M\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$ with a constant mass matrix M. This is a convenience for ODEs, but more importantly, the code allows M to be singular so that it can solve DAEs. The same is true of the implicit Runge-Kutta code RADAU5 and an extended version of `ode15s`, a code based on some formulas closely related to the BDFs. The methods of these codes can be derived formally for nonsingular M by writing out the formula for the function $M^{-1}\mathbf{f}(t, \mathbf{y})$ and then multiplying by M. In the iteration for evaluating the implicit formulas, an additional Jacobian is approximated by a constant matrix J. In all cases, the resulting schemes reduce to the usual ones when $M = I$. It is found that with the usual assumptions about DAEs of index 1, the iteration matrices are nonsingular, even when M is singular, and the solution process is essentially the same as for an ODE. However, starting is much more difficult because "consistent" initial values are required when solving a DAE. LSODI and RADAU5 expect the user to supply consistent initial values; `ode15s` accepts consistent initial values but computes them automatically by default. All these codes would accept the DAE (3,4) in the form

$$\begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{U} \\ \mathbf{P} \end{pmatrix}' = \begin{pmatrix} \mathbf{f}(\mathbf{U}) - (K + N(\mathbf{U}))\mathbf{U} - C\mathbf{P} \\ C^\top C\mathbf{P} - C^\top (\mathbf{f}(\mathbf{U}) - (K + N(\mathbf{U}))\mathbf{U}) \end{pmatrix}.$$

It is not hard to see that the key issue in showing that linear conservation laws are preserved is the computation of $\mathbf{P}$. Because it appears linearly in the last set of equations, the linearization used in the iterative evaluation of the implicit formula is exact for these variables and the computed $\mathbf{P}$ should satisfy the algebraic equations exactly. Accordingly, straightforward solution of the DAE (3,4) of index 1 by any one of these three codes should yield numerical solutions that satisfy the discrete incompressibility condition (2).

We have considered only the case of boundary conditions for which $\mathbf{f}(\mathbf{U}, \mathbf{P})$ does not depend on $\mathbf{P}$. If it did depend on the pressure, the algebraic equations would be nonlinear and we would not expect the pressure approximation to satisfy (4) exactly, suggesting that the laws would not be preserved automatically. We have not pursued this matter because the more restricted class of problems studied here already satisfies our goal of a nontrivial application and because it appears that the analysis will depend on the particular method used and details of its implementation.

REFERENCES

1. H.H. Robertson and M.J. McCann, A note on the numerical integration of conservative systems of first-order ordinary differential equations, *Computer J.* **12**, 81, (1969).
2. C.W. Gear, Maintaining solution invariants in the numerical solution of ODEs, *SIAM J. Sci. Stat. Comput.* **7**, 734–743, (1986).

3. L.F. Shampine, Conservation laws and the numerical solution of ODEs, *Computers Math. Applic.* **12B** (5/6), 1287–1296, (1986).
4. J.R. Rosenbaum, Conservation properties of numerical integration methods for systems of ordinary differential equations, *J. Comp. Phys.* **20**, 259–267, (1976).
5. P.M. Gresho, S.T. Chan, R.L. Lee and C.D. Upson, A modified finite element method for solving the time-dependent incompressible Navier-Stokes equations. Part 1: Theory, *Internat. J. Numer. Methods in Fluids,* **4**, 557–598, (1984).
6. K.E. Brenan, S.L. Campbell and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, SIAM, Philadelphia, (1996).
7. J.R. Dormand, *Numerical Methods for Differential Equations*, CRC Press, Boca Raton, (1996).
8. E. Hairer, S.P. Norsett and G. Wanner, *Solving Ordinary Differential Equations I*, Springer, New York, (1987).
9. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*, Springer, New York, (1991).
10. L.F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, (1994).
11. R.W. Brankin, I. Gladwell and L.F. Shampine, RKSUITE: a suite of Runge-Kutta codes, In *Contributions to Numerical Analysis* Series in Applicable Analysis, (Edited by R.P. Agarwal), Volume 2, World Scientific, Singapore, pp. 85–98, (1993).
12. L.F. Shampine and M.K. Gordon, *Computer Solution of Ordinary Differential Equations*, W.H. Freeman, San Francisco, CA, (1975).
13. P.N. Brown, G.D. Byrne and A.C. Hindmarsh, VODE: a variable-coefficient ODE solver, *SIAM J. Sci. Stat. Comput.* **10**, 1038–1051, (1989).
14. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, (1971).
15. A.R. Curtis, M.J.D. Powell and J.K. Reid, On the estimation of sparse Jacobian matrices, *J. Inst. Math. Appl.* **13**, 117–119, (1974).
16. L.F. Shampine and M.W. Reichelt, The MATLAB ODE suite, *SIAM J. Sci. Comput.* **18**, 1–22, (1997).
17. T.E. Hull, W.H. Enright, B.M. Fellen and A.E. Sedgwick, Comparing numerical methods for ordinary differential equations, *SIAM J. Numer. Anal.* **9**, 603–637, (1972).
18. A.C. Hindmarsh, LSODE and LSODI, two new initial value ordinary differential equation solvers, *SIGNUM Newsletter* **15**, 10–11, (1980).