A Quasi-Conservative Version of the Semi-Lagrangian Advection Scheme*

A. PRIESTLEY

Institute of Computational Fluid Dynamics, Department of Mathematics, University of Reading, Whiteknights, Reading, United Kingdom
(Manuscript received 11 March 1992, in final form 18 May 1992)

ABSTRACT

The semi-Lagrangian method is now, perhaps, the most widely researched algorithm in connection with numerical weather prediction (NWP) codes. Monotonicity has been added to the basic method by the use of shape-preserving interpolation, and, more recently, by using ideas from flux corrected transport (FCT). In this paper, the authors describe how to make the scheme quasi-conservative. Although the lack of conservation of the semi-Lagrangian method is not widely regarded as a serious problem, for climate studies, where many tens of thousands of time steps are needed, it could become so. The method proposed here is very cheap, and hence is a viable proposition for addition to existing semi-Lagrangian codes. Making the scheme conservative as well as monotone gives the scheme shock-capturing properties, thus making the method much more useful in application areas outside of meteorology.

1. Introduction

Numerical schemes that follow characteristics backward in time and then interpolate at their feet have a history stretching back to the very early days of computational fluid dynamics (Courant et al. 1952). In the last decade, these methods have been catching favor again. In the finite-element literature, there has been a body of work on the Lagrange-Galerkin method by, among others, Benqué et al. (1982), Recovier and Pironneau (1982), Douglas and Russell (1982), Le-saint (1977), Morton et al. (1988), Russell (1985), and Säli (1988). Meanwhile, the finite-difference version of this idea has progressed in the form of the semi-Lagrangian method. A useful survey of this method can be found in Staniforth and Côté (1991). It is this form of the Lagrangian method that we shall be using here.

Consider the Cauchy problem for the scalar, linear advection equation for \( u(x, t) \)

\[
\begin{align*}
  u_t + a \cdot \nabla u &= 0, \quad x \in \mathbb{R}^d, \quad t > 0, \\
  u(x, 0) &= u_0(x),
\end{align*}
\]

(1.1)

where \( u_0 \) is the initial data and \( d \) is the number of dimensions. We can now define characteristic paths or trajectories, \( X(x, s; t) \), in two ways, either as the solution to an ordinary differential equation,

\[
X(x, s; s) = x,
\]

(1.3)

\[
\frac{dX(x, s; t)}{dt} = a[X(x, s; t), t];
\]

(1.4)

or, if desired, as the solution of the integral equation

\[
X(x, s; t) = x + \int_s^t a[X(x, s; \tau), \tau]d\tau.
\]

In order to simplify the notation, for \( t^{n+1} = t^n + \Delta t \), we will denote the foot of the characteristic path at time \( t^n \) to be at \( x \), usually termed the departure point, and its arrival point at time \( t^{n+1} \) to be at \( y \). In terms of the more general notation, these are

\[
x = X(y, t^n; t^{n+1});
\]

and

\[
y = X(x, t^n; t^{n+1}).
\]

A unique (absolutely continuous) solution to (1.3) and (1.4) can be guaranteed under fairly mild assumptions about the velocity field \( a \) (Mizohata 1973, for example). The solution to the original partial differential equations (1.1) and (1.2) is now given by the relation

\[
u[X(\cdot, t + \tau), t + \tau] = u(\cdot, t),
\]

(1.5)

which is just saying that the solution is constant along a trajectory. Putting \( \tau = \Delta t \) and simplifying the notation as above we have,

\[
u(y, t^{n+1}) = u(x, t^n).
\]

* The work reported here forms part of the research program of the Reading/Oxford Institute for Computational Fluid Dynamics and was funded by a grant from the SERC.

Corresponding author address: Dr. A. Priestley, Department of Mathematics, University of Reading, Whiteknights, P.O. Box 220, Reading, RG 6 2AX United Kingdom.

© 1993 American Meteorological Society
If \( y \), the arrival point, is then put equal to a grid point, \( y \), for example, the foot of the trajectory \( x \), the departure point, will not in general be a grid point, and hence, some sort of interpolation will need to be performed. Cubic interpolation seems to have been favored in the development of the semi-Lagrangian method, giving a reasonable compromise between accuracy and computational effort. The two most popular are, in this context, the somewhat confusingly termed cubic-Lagrangian interpolation and cubic-spline interpolation.

In Williamson and Rasch (1989) and Rasch and Williamson (1990), monotonicity was introduced to the semi-Lagrangian method. Although they experimented with many types of interpolation, perhaps the most pleasing was that using Hermite-cubic interpolation (the monotonicity being obtained by limiting the derivative values). More recently, Bermejo and Staniforth (1992), using ideas from the FCT schemes of Boris and Book (1973) and Zalesak (1979), have created a semi-Lagrangian scheme that achieves monotonicity while still using the cubic-Lagrangian or cubic-spline interpolation, or for that matter any non-specialized interpolation.

In this paper, we will now introduce conservation to the semi-Lagrangian method. This algorithm fits very neatly into the method of Bermejo and Staniforth (1992) and this paper will be referred to frequently in the following. Although the method presented here has more in common with the second method of introducing monotonicity mentioned earlier, there is no reason why it could not be combined with the approach of Williamson and Rasch (1989) to again achieve the aim of a monotonic and conservative scheme.

In the remainder of this section, the basis of the monotonic algorithm described in Bermejo and Staniforth (1992) will be given. In the following section, the algorithm for recovering conservation will be given. Section 3 contains two test problems that will be solved with the currently proposed method, and then finally, in section 4, a summary of the work will be given.

The first stage of any Lagrangian scheme is the solution of the trajectory problem, (1.3) and (1.4). In principle, any ODE solver could be used to do this. In the first test case (to be done later), the exact trajectories are known and used. In the second, nonlinear problem, Euler's forward-difference method is used to determine the departure point of the grid point. Since we are only doing a comparative study, this is adequate. In practice, this \( O(\Delta t) \) time stepping is not, however, regarded as adequate (Staniforth and Côté 1991), and \( O(\Delta t^2) \) time-stepping schemes are generally preferred. Working in spherical coordinates can also raise problems (see Ritchie 1987).

Having found the departure point, we must now interpolate the value of the solution. The most common types used have already been mentioned, and the mechanics of these methods can be found in most books on numerical methods (Johnson and Riess 1977, for example). Here, we will only be using cubic-spline interpolation (bicubic splines in two dimensions). This is purely because this interpolation comes out well in the tests of Bermejo and Staniforth (1992). The results presented here are all compared with like schemes, and so the precise choice of interpolation is somewhat arbitrary and a cheaper interpolation or a more local one, or both, could be used. The only difference would then be a slight increase in the errors presented.

To obtain a monotone scheme via the FCT approach, we need two approximations to the solution at the new time level. These are usually referred to as the high-order solution \( U^H \) and the low-order solution \( U^L \). These terms are slightly misleading, since the only constraint on these solutions is that the low-order solution be monotone, or more precisely in our context, it should introduce no new extrema. (This is a rather cumbersome phrase and so we shall continue to use the word monotone with its extension to higher-order schemes and multidimensions being understood.) We could therefore use the monotone Hermite-cubic interpolation of Williamson and Rasch (1989) as either our high-order scheme or our low-order scheme. As we have mentioned previously, we will use the cubic-spline interpolation for our high-order scheme and linear interpolation for the low-order, monotone scheme. From these two solutions, we then define our monotone solution \( U^M \) at the point \( k \) to be given by

\[
U^M_k = \alpha_k U^H_k + (1 - \alpha_k) U^L_k,
\]

with

\[
0 \leq \alpha_k \leq 1,
\]

where the \( \alpha_k \)'s are yet to be chosen. Clearly, the objective is to find \( \alpha \)'s as large as possible while maintaining monotonicity. A solution is guaranteed to exist, corresponding to \( \alpha_k = 0 \), because of the monotone nature of \( U^L \). Upper and lower bounds are now placed on (1.6) to ensure a monotone solution. Denote by \( U^n \) the monotone solution from the previous time level and by \( \{ U^n, k \} \) the set of solution values at points surrounding the departure point of the grid point \( x_k \). This is \( y(x_k) \) in our previous notation; in the notation of Bermejo and Staniforth (1992), it is \( \{ U^n, k \} = \{ U_{k1}, U_{k2}, U_{k3}, U_{k4} \} \). We now take the highest value of \( \alpha_k \) that satisfies both (1.7) and the condition

\[
\min(\{ U^n, k \}, U^L_k) \leq \alpha_k U^H_k + (1 - \alpha_k) U^L_k \leq \max(\{ U^n, k \}, U^L_k).
\]

Equation (1.8) only differs from the one given in Bermejo and Staniforth (1992) in that the value of the low-order solution has been included in the bounds. When calculated by linear interpolation from the values \( \{ U^n, k \} \) for pure advection problems it will clearly not affect anything. For more general problems, that is, when source terms are present or we are solving a non-
linear system, it is necessary to include this value. We now proceed to discuss the recovery of conservation.

2. Recovering conservation

We shall refer to the maximum values of $\alpha$ that satisfy constraints (1.7) and (1.8) as $\{\alpha_k^{\text{max}}\}$. Suboptimal values of the $\alpha_k$'s can now be chosen to try and make the scheme conservative. That is we choose $\alpha_k$'s such that

$$0 \leq \alpha_k \leq \alpha_k^{\text{max}}$$

while at the same time trying to enforce, for example,

$$\int U^M(x) \, dx = \int U^0(x) \, dx = C. \quad (2.1)$$

In general, of course, the best we can do is to minimize the difference between the two quantities. In the examples given later, we have precisely the situation stated in (2.1), but in other problems it may be necessary to take into account source terms and boundary conditions. There are many ways this problem can be solved—by linear programming methods, for example—but the following algorithm is found to be a very efficient and direct way of obtaining a solution. The efficiency of the algorithm will be demonstrated in the results section and also the fact that in using suboptimal $\alpha$'s we sacrifice comparatively little accuracy compared with the introduction of the monotonicity.

First, we define $S_i$ to be the area associated with node $i$. With a regular mesh, this is just $h$ in one dimension and $h^2$ in two dimensions. A more general case is illustrated in Fig. 1. Second, let

$$\beta_i = (U_i^M - U_i^0) S_i.$$

The problem now is to maximize the $\alpha$'s subject to the following condition arising from (1.6) and (2.1),

$$\sum_i \alpha_i (U_i^M - U_i^0) S_i = C - \sum_i U_i^f S_i = C^*.$$

Assume that

$$\sum_i \alpha_i^{\text{max}} \beta_i > C^*.$$

If this is not the case, then the definitions are just changed so that

$$\beta_i \rightarrow -\beta_i$$

$$C^* \rightarrow -C^*.$$

Step 1:

If $\beta_i \leq 0$, then $\alpha_i = \alpha_i^{\text{max}}$

iflag($i$) = 1;

otherwise,

$$\alpha_i = 0$$

iflag($i$) = 0.

Step 2: Define a surplus

$$= C^* - \sum_{\text{iflag}(k) = 1} \alpha_k \beta_k.$$

Step 3: Define the average value of $\alpha$

$$\alpha_{\text{avg}} = \frac{\text{surplus}}{\sum_{\text{iflag}(k) = 0} \beta_k}.$$

Step 4: If $\alpha_{\text{avg}} < \alpha_k^{\text{max}}$ everywhere, iflag($k$) = 0;
then for these $k$ set $\alpha_k = \alpha_{\text{avg}}$;
END

Step 5: Else for those $k$ that satisfy both
iflag($k$) = 0 and $\alpha_{\text{avg}} > \alpha_k^{\text{max}}$,
put

$$\alpha_k = \alpha_k^{\text{max}}$$

iflag($k$) = 1.

Step 6: GOTO 2.

If the surplus is negative, then there is no conservative solution and the best solution, in regard to conservation, is given by the initial setup of the $\alpha$'s in step 1.

In step 4, where the successful algorithm finishes, there is a certain amount of freedom in what we do. Here, we have just set the remaining $\alpha$'s to the average value. This is quick and is certainly a reasonable thing to do. If desired, however, these extra degrees of freedom, which correspond to the solution lying on a plane in the linear programming case, could be used to try to achieve some other goal. In the next section, we

Fig. 1. The shaded area shows the value of $S_i$ for the point $P$ on a nonuniform mesh.
shall monitor the second moment of the solution, which ideally should also be conserved. This, of course, is not conserved by this algorithm, but these extra degrees of freedom could be used to try to achieve conservation of the second moment, or indeed, any other conserved quantity.

3. Results

The first problem is that of the slotted cylinder. This is example 3 of Bermejo and Staniforth (1992). On the domain $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$, with a uniform grid of size $h = \frac{1}{100}$, we define the slotted cylinder. It is centered at $(-\frac{1}{4}, 0)$ with radius $15h$. Its height is 4. The slot has width $6h$ and length $22h$. The initial data is shown in Fig. 2.

This profile is then advected by a given velocity field $a = \omega(-y, x)$, which causes the cylinder to rotate with angular velocity $\omega$. The value of $\omega$ is taken to be $0.3636 \times 10^{-4}$ s$^{-1}$. This corresponds to one rotation every 20 days. Taking a time step of $\Delta t = 1800$ s, a revolution is completed every 96 time steps. Six revolutions are performed.

Following Bermejo and Staniforth (1992), we define some errors for this problem. The first moment of the approximate solution $U$, normalized by the first moment of the exact solution $u$, is given by

\[
RFM = \frac{\int_{\Omega} U(x, t_{\text{output}}) \, dx}{\int_{\Omega} u(x, 0) \, dx},
\]

and the second moment is similarly defined by

\[
RSM = \frac{\int_{\Omega} U^2(x, t_{\text{output}}) \, dx}{\int_{\Omega} u^2(x, 0) \, dx}.
\]

A fairly obvious discretization of these integrals is used. The maximum and minimum values are also given.
Again following Bermejo and Staniforth (1992) and Takacs (1985), we write the discrete $l_2$ error as

$$\|u - U\|_2^2 = \frac{1}{K} \sum_{k=1}^{K} (u_k - U_k)^2,$$

where $K$ is the total number of points. This is then split into the dissipation error (DISSER) and the dispersion error (DISPER), that is, we can write

$$\|u - U\|_2^2 = \text{DISSER} + \text{DISPER},$$

where

\begin{align}
\text{DISSER} & = [\sigma(u) - \sigma(U)]^2 + (\bar{u} - \bar{U})^2, \\
\text{DISPER} & = 2(1 - \rho)\sigma(u)\sigma(U). 
\end{align} \tag{3.1} \tag{3.2}

In (3.1) and (3.2), the overbar denotes an average value and $\sigma$ is the standard deviation ($\sigma^2$ the variance). The correlation between $u$ and $U$ on the mesh is denoted by $\rho$.

Results are presented for three versions of the semi-Lagrangian method. All use cubic-spline interpolation.

The first is just the basic method, the second is the quasi-monotone version (as in Bermejo and Staniforth 1992), and the third is the new quasi-monotone and conservative method. Plots of the results after six revolutions are shown in Figs. 3, 4, and 5, respectively. The figures indicate that the monotone version of the method lives up to its name. The slot remains well defined in both of the monotonic implementations. Indeed, there is no visible difference between the quasi-monotone solution (Fig. 4) and the quasi-monotone and conservative solution (Fig. 5). More objective comparisons can be made from Tables 1, 2, and 3. Obvious statements that can be made after looking at these tables are that the monotonic implementations are indeed monotone, and that the conservative version is conservative. The results in Table 2 appear to be slightly better than those for the identical problem in Bermejo and Staniforth (1992). This is almost certainly entirely due to the fact that we made use of the known trajectory paths in this problem for the calculations presented here. Looking at the errors DISSER and DISPER, the two quantities making up the discrete $l_2$ error norm, we can see how the modifications to the basic algorithm have changed the accuracy.

First, comparing the results for the basic method and the quasi-monotone method (Tables 1 and 2, respectively), we see that the dissipation error DISSER has increased by a factor of just over 4. The larger dispersion error DISPER, which dominates the $l_2$ error, rises by between about 10% and 20%. Comparing tables 2 and 3 to see how the introduction of the quasi-conservative algorithm affects accuracy, we see an approximate rise in the dissipation error of 10% and in the larger dispersion error of less than 1%. This demonstrates that very little accuracy is sacrificed in achieving conservation compared to what has already been lost due to the introduction of monotonicity.

As to cost, an increase of 0.9% in CPU time was needed, compared to calculating the quasi-monotone scheme. This was for a problem where the trajectories were calculated once and then stored, and the cubic-spline interpolation was able to take advantage of the interpolation points being fixed. Timings can be found in Table 4 for both the results obtained here on a SUN Sparc 1 and for results obtained by Gravel and Staniforth (1992, personal communication) on a CDC 4680,
Table 2. Errors for slotted-cylinder problem using semi-Lagrangian method with cubic-spline interpolation and monotonicity.

<table>
<thead>
<tr>
<th>Time steps</th>
<th>RFM</th>
<th>RSM</th>
<th>Maximum</th>
<th>Minimum</th>
<th>DISSER</th>
<th>DISPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>1.00387</td>
<td>0.873845</td>
<td>3.99947</td>
<td>0.0</td>
<td>4.1257 × 10⁻³</td>
<td>4.3240 × 10⁻²</td>
</tr>
<tr>
<td>192</td>
<td>1.00452</td>
<td>0.85508</td>
<td>3.99849</td>
<td>0.0</td>
<td>5.5065 × 10⁻³</td>
<td>5.0314 × 10⁻²</td>
</tr>
<tr>
<td>288</td>
<td>1.00563</td>
<td>0.84259</td>
<td>3.99429</td>
<td>0.0</td>
<td>6.5539 × 10⁻³</td>
<td>5.4966 × 10⁻²</td>
</tr>
<tr>
<td>384</td>
<td>1.00689</td>
<td>0.83307</td>
<td>3.99113</td>
<td>0.0</td>
<td>7.4235 × 10⁻³</td>
<td>5.8595 × 10⁻²</td>
</tr>
<tr>
<td>480</td>
<td>1.00801</td>
<td>0.82514</td>
<td>3.99011</td>
<td>0.0</td>
<td>8.1949 × 10⁻³</td>
<td>6.1660 × 10⁻²</td>
</tr>
<tr>
<td>576</td>
<td>1.00911</td>
<td>0.81842</td>
<td>3.98925</td>
<td>0.0</td>
<td>8.8848 × 10⁻³</td>
<td>6.4380 × 10⁻²</td>
</tr>
</tbody>
</table>

with 32-bit precision, and a Cray XMP, with 64-bit precision. Times are for 192 time steps of the aforementioned problem. The cost seems to be very machine dependent, but it must be emphasized that the code used for the Cray had not been specially written, and hence it may not have vectorized as fully as it might. It is believed that this timing could be reduced significantly.

The second test problem is that of the one-dimensional inviscid Burgers' equation. This equation has been solved before using the semi-Lagrangian method in Kuo and Williams (1990), but here we will use the same problem as Bermejo and Staniforth (1992). We are required to find the solution to

\[
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0 \quad x, t \in [-1, 1] \times [0, T], \quad (3.3)
\]

\[
u(t, 0) = \frac{1}{4} + \frac{1}{2} \sin \pi x, \quad (3.4)
\]

\[u(-1, t) = u(1, t) \quad \forall t.\]

The initial data [Eq. (3.4)] are depicted in Fig. 6. We do not take the velocity from (3.3) to be \(u\), but rather define it to be \(a(u)\), where \(a(u) = \frac{\partial f}{\partial u}\) and \(f\) is the flux function defined by \(f(u) = \frac{1}{2}u^2\) for Burgers' equation. This is then approximated by a simple central difference as

\[a(u) = \frac{f(u_{i+1}) - f(u_{i-1})}{u_{i+1} - u_{i-1}}.\]

In smooth regions of the flow, this should be a good approximation to \(u\), whereas in nonsmooth regions it should give a good approximation to the shock speed. The velocity thus generated is then used in Euler's method to determine the departure point, that is,

\[y_i = x_i - a(u_i) \Delta t.\]

For this problem, it was not felt necessary to use a more sophisticated approach. Eighty grid points were used giving a value of \(h = \frac{1}{40}\). Figure 7 shows the result at \(T = 0.65\). All three methods give virtually identical results at this time, before the shock develops, and we only show the result for the quasi-monotone and conservative method. This solution was obtained in 10 time steps, implying a maximum CFL number of just under 2. The next output time in Bermejo and Staniforth (1992) is \(T = 0.9\). We, however, found very little to distinguish the methods at this time; the basic method had a slight oscillation, but was otherwise identical to the other two solutions.

In Bermejo and Staniforth (1992), a shock-fitting technique was used to obtain their results. It should be stressed that when we later get rather poor results for the quasi-monotone semi-Lagrangian method, the shock fitting has not been used and it is hence not comparable with the scheme presented by Bermejo and Staniforth (1992). We reject the use of shock fitting for two main reasons. First, although shock fitting may work very well for the one-dimensional Burgers' equation (3.3), for a one-dimensional system of equations such as the Euler equations of gas dynamics, we already face difficulties. Numerically, in this situation the Rankine–Hugoniot equations predict three separate shock speeds and the conservation laws three separate shock positions. These difficulties can be overcome (Morton and Sweby 1987), although it is not a very aesthetically

Table 3. Errors for slotted-cylinder problem using semi-Lagrangian method with cubic-spline interpolation, monotonicity, and conservation.

<table>
<thead>
<tr>
<th>Time steps</th>
<th>RFM</th>
<th>RSM</th>
<th>Maximum</th>
<th>Minimum</th>
<th>DISSER</th>
<th>DISPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>1.0</td>
<td>0.86936</td>
<td>3.99990</td>
<td>0.0</td>
<td>4.4037 × 10⁻³</td>
<td>4.3300 × 10⁻²</td>
</tr>
<tr>
<td>192</td>
<td>1.0</td>
<td>0.84970</td>
<td>3.99842</td>
<td>0.0</td>
<td>5.8975 × 10⁻³</td>
<td>5.0384 × 10⁻²</td>
</tr>
<tr>
<td>288</td>
<td>1.0</td>
<td>0.83560</td>
<td>3.99398</td>
<td>0.0</td>
<td>7.1163 × 10⁻³</td>
<td>5.5111 × 10⁻²</td>
</tr>
<tr>
<td>384</td>
<td>1.0</td>
<td>0.82431</td>
<td>3.99062</td>
<td>0.0</td>
<td>8.1843 × 10⁻³</td>
<td>5.8790 × 10⁻²</td>
</tr>
<tr>
<td>480</td>
<td>1.0</td>
<td>0.81484</td>
<td>3.98967</td>
<td>0.0</td>
<td>9.1441 × 10⁻³</td>
<td>6.1885 × 10⁻²</td>
</tr>
<tr>
<td>576</td>
<td>1.0</td>
<td>0.80657</td>
<td>3.98757</td>
<td>0.0</td>
<td>10.031 × 10⁻³</td>
<td>6.4621 × 10⁻²</td>
</tr>
</tbody>
</table>
pleasing approach. In two dimensions, things become even worse in that it is even a problem to detect shocks unless they are lying along, or close to, grid lines. If tolerances are reduced to the extent that most of the shock is recognized, it usually implies that "ghost" shocks will also be caught. A notable exception to this rule would appear to be that of Moretti (1987).

It is known, however, that with conservation we are guaranteed the correct shock position (Lax and Wendroff 1960), while monotonicity ensures convergence (Sanders 1983). An entropy condition is needed to finally complete the picture of convergence to the correct physical solution (Harten et al. 1976). Thus, by adding conservation to the quasi-monotone scheme, we are freed from the need to fit shocks, and hence the scheme becomes much more widely applicable even in higher dimensions. To demonstrate the advantages of having conservation, the previous problem was run to $T = 1.5$. This time was reached in 30 time steps, giving a maximum CFL number of 1.5. A control solution, obtained with the quasi-monotone and conservative method on a grid with 2000 points, is shown in Fig. 8. Figures 9, 10, and 11 show the solution as calculated by the basic method, the quasi-monotone method, and the quasi-monotone and conservative method. At first glance, all three methods have performed well, with the basic semi-Lagrangian method only suffering from a slight oscillation behind the shock. On closer inspection, however, we see that both the basic and quasi-monotone methods have not transported the shock far enough. The quasi-monotone and conservative version has got the shock in the right place. This is confirmed by checking the function values to see where the shock lies. Allowing one intermediate point, in the control solution the shock lies somewhere in the interval $[-0.626, -0.624]$. The basic and quasi-monotone semi-Lagrangian methods both predict a shock position somewhere in the interval $[-0.7, -0.65]$, while the quasi-monotone and conservative algorithm correctly predicts the shock to be somewhere in the interval $[-0.65, -0.6]$. We must stress again that this is not the implementation of the quasi-mono-

---

<table>
<thead>
<tr>
<th>Method</th>
<th>SUN Sparc 1</th>
<th>CDC 4680</th>
<th>Cray XMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spline only</td>
<td>597.5</td>
<td>88.4</td>
<td>17.3</td>
</tr>
<tr>
<td>Spline + monotonicity</td>
<td>736.1</td>
<td>148.3</td>
<td>18.3</td>
</tr>
<tr>
<td>Spline + monotonicity + conservation</td>
<td>742.6</td>
<td>155.8</td>
<td>23.5</td>
</tr>
<tr>
<td>Percentage overhead</td>
<td>0.9</td>
<td>5.1</td>
<td>29</td>
</tr>
</tbody>
</table>

**Table 4.** The CPU times for the various schemes on different machines.

---

**Fig. 6.** Initial data for Burgers' equation.

**Fig. 7.** Solution at $T = 0.65$ using quasi-monotone and conservative semi-Lagrangian method.

---

**Fig. 8.** Control solution at $T = 1.5$. 

Unauthenticated | Downloaded 12/08/23 11:27 AM UTC
tone semi-Lagrangian scheme advocated for this problem by Bermejo and Staniforth (1992). We, however, believe that the introduction of monotonicity used here to be a much more general and robust procedure for obtaining correct shock positions than the shock-fitting technique.

The extra cost involved in calculating the conservative solution, over and above that of calculating the monotone solution, represents an extra 0.75% of CPU time. The reduction over the previous case is because the trajectories have to be calculated at each time step now, and the interpolation routine cannot take advantage of the interpolation points being fixed. Clearly, if a more expensive trajectory solver were used, as is likely to be the case in practice, then the relative cost will go down slightly, just as it will go up a little if a cheaper interpolation is used. In either event, conservation is achieved at very low cost.

4. Treatment of source and boundary terms

In both test cases performed here, the equations were homogeneous and the domains had periodic boundary conditions. This meant that there was a constant, denoted by $C$ in (2.1), which equaled the amount of the conserved quantity $u$ for all time. We now consider the nonhomogeneous problem corresponding to (1.1),

$$u_t + a \cdot \nabla u = b(x, t),$$

with the initial conditions still supplied by (1.2). The constant $C$ now becomes a function of time, $C(t)$. For our purposes, we now need to replace the $C$ used in the algorithm described previously by its value at $t = t^{n+1}$. That is, we take

$$C = C^{n+1} = C^n + \int_\Omega b(x, t) dxdt,$$  \quad (4.1)

where $\Omega$ represents the spatial domain and $C^0$ is just given by the spatial integral of the initial data $u_0(x)$.

In some situations, $b(x, t)$ is a known function, where a pollutant is released at a known rate (Pudykiewicz 1989), and it may be possible to evaluate the integral in (4.1) analytically. More generally, the function $b$ will also depend upon the solution $u$, and the integral will have to be approximated. A simple and obvious choice would be to put

$$C^{n+1} = C^n + \Delta t \int_\Omega b(x, t^n) dx.$$

If this is a part of some iteration, that is, there is already a guess for $u^{n+1}$, then clearly a more accurate treatment of the integral may be used.

Boundary terms affecting conservation can be treated in an entirely analogous manner, a surface integral of the flux function now arising in (4.1).
5. Summary

In this paper we have shown how the semi-Lagrangian method can be made conservative. Although the procedure presented here falls in most naturally with the monotone scheme of Bermejo and Staniforth (1992), there is no reason why it should not be used with the monotone schemes of Rasch and Williamson (1990). Conservation is achieved at very little expense above that needed for the calculation of the monotone scheme.

This makes it a practicable addition to semi-Lagrangian codes for climate modeling, or other applications, where loss (or gain) in conserved quantities can become an issue.

In addition, the shock-capturing properties endowed upon the semi-Lagrangian method due to the addition of conservation and monotonicity makes the scheme much more widely applicable.

REFERENCES


