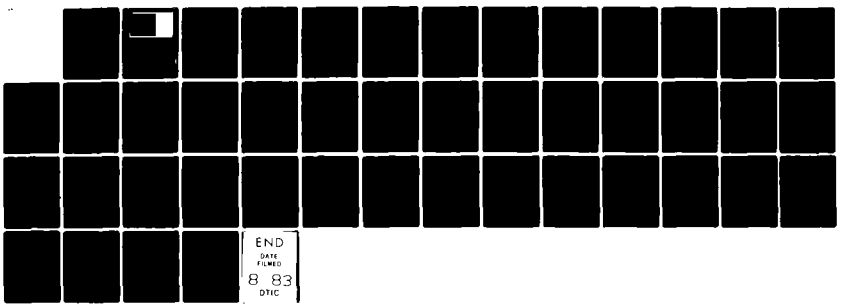
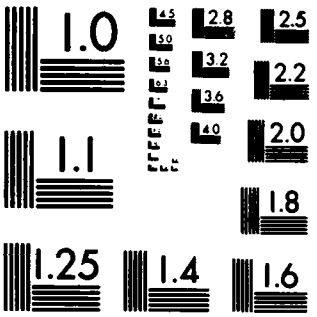
AD-A130 512 A STABLE ADAPTIVE NUMERICAL SCHEME FOR HYPERBOLIC 1/ 1
CONSERVATION LAWS(U) WISCONSIN UNIV-MADISON MATHEMATICS
RESEARCH CENTER B J LUCIER MAY 83 MRC-TSR-2517
UNCLASSIFIED DAAG29-80-C-0041 F/G 12/1 NL



END
DATE FILMED
8 83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

MRC Technical Summary Report #2517

A STABLE ADAPTIVE NUMERICAL SCHEME FOR HYPERBOLIC CONSERVATION LAWS

Bradley J. Lucier

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

May 1983

(Received April 5, 1983)

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

National Science Foundation
Washington, DC 20550

UNIVERSITY OF WISCONSIN - MADISON
MATHEMATICS RESEARCH CENTER

A STABLE ADAPTIVE NUMERICAL SCHEME FOR HYPERBOLIC CONSERVATION LAWS

Bradley J. Lucier*

Technical Summary Report #2517

May 1983

ABSTRACT

A new adaptive finite-difference scheme for scalar hyperbolic conservation laws is introduced. A key aspect of the method is a new automatic mesh selection algorithm for problems with shocks. We show that the scheme is L^1-stable in the sense of Kuznetsov, and that it generates convergent approximations for linear problems. Numerical evidence is presented that indicates that if an error of size ε is required, our scheme takes at most $O(\varepsilon^{-3})$ operations. Standard monotone difference schemes can take up to $O(\varepsilon^{-4})$ calculations for the same problems.

AMS (MOS) Subject Classification: 65M10, 35L65

Key Words: Conservation laws, finite-difference schemes, adaptive numerical methods.

Work Unit Number 3 - Numerical Analysis and Scientific Computing

* Division of Mathematical Sciences, Purdue University, W. Lafayette, IN 47907.

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041. This material is based upon work partially supported by the National Science Foundation under Grant No. MCS-7927062, Mod. 1.

SIGNIFICANCE AND EXPLANATION

Certain problems in gas dynamics, oil reservoir simulation and other fields can be modeled by hyperbolic conservation laws, a class of partial differential equations. The solutions of such problems are typically made up of smooth surfaces separated by discontinuities, or shocks. Usually, less information is needed to specify the solution in the smooth regions than in the shock regions.

In this paper we introduce a stable finite-difference scheme for conservation laws that incorporates a time-varying, nonuniform computational mesh. At any given time, ~~our~~ mesh selection algorithm chooses a mesh based on the approximation calculated up to the time. The algorithm uses knowledge of a solution's structure to reduce the number of meshpoints in the regions where the solution is smooth. This reduces the method's computational complexity while maintaining full accuracy. ~~We~~ prove that ~~our~~ method is stable for the complete nonlinear problem, and that it converges for linear problems. ~~We give~~ examples where ~~our~~ method is asymptotically faster than previous ones.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.

A Stable Adaptive Numerical Scheme for Hyperbolic Conservation Laws

Bradley J. Lucier *

1. Introduction.

Our focus in this paper is the efficient solution of the hyperbolic conservation law

$$u_t + f(u)_x = 0, \quad x \in \mathbb{R},\ t > 0, \qquad (C)$$
$$u(x,0) = u_0(x).$$

We use an adaptive finite-difference scheme that takes advantage of the structure of the solution of (C) to reduce its computational complexity. We consider the scalar case $u(x,t) \in \mathbb{R}$. For the general nonlinear problem, we offer numerical evidence that there is asymptotic improvement in the rate of decrease in the error as a function of computational complexity. For linear problems we prove that a version of our method converges if the initial data is sufficiently smooth.

Our method is, generally speaking, a *viscosity* method. The class of monotone finite-difference schemes for conservation laws are also viscosity methods. Monotone schemes have been analyzed by Harten et al [12], Crandall and Majda [5], Kuznetsov [15], Sanders [19], and Lucier [16]. These schemes converge to the entropy weak solution of the conservation law (C), as formulated by Kruzkov [14]. Kuznetsov provided a general theory of approximation for approximate solutions of (C). He used this theory to provide error estimates for monotone

* Division of Mathematical Sciences, Purdue University, W. Lafayette, IN 47907.

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041. This material is based upon work partially supported by the National Science Foundation under Grant No. MCS-7927062, Mod. 1.

difference methods for problems in arbitrary space dimensions using uniform grids and local difference operators. His techniques were used by Sanders and by Lucier to provide error estimates for difference schemes with nonuniform meshes and nonlocal difference operators respectively. Sanders' paper provides a lucid treatment of Kuznetsov's theory.

One of the considerations in developing our algorithm was that it must exhibit nonlinear stability properties similar to those of monotone finite-difference schemes. Our approach may be conceptualized as follows. We take a uniform mesh in $[a,b]\times[0,T]$ with a mesh spacing of Δt. Meshpoints are removed from the mesh where they are not needed to achieve the required accuracy. A standard finite-difference operator is used to advance the approximate solution from one timestep to the next. (Special techniques are used when the meshes differ from one timestep to the next.) Our method differs from previous ones in the algorithm for choosing the mesh, the interpretation of the approximate solution, and our specific choice of finite-difference operator.

Other adaptive methods have been devised for evolution equations. Davis and Flaherty's algorithm [6] for the solution of evolution equations is designed to solve smooth problems without shocks. They use an L^2 analysis to choose their mesh and to provide an error analysis for some problems. Our algorithm is similar to theirs, in that it tries to equidistribute a measure of the error among the meshpoints. However, we use an L^1 analysis, since the solutions of (C) are stable in L^1, but not L^2.

Other general algorithms for evolution equations were proposed by Miller [17] and Dupont [9]. Dupont supplies convergence analyses for his methods. These algorithms are mainly finite element algorithms that converge well for solutions that are stable in L^2. Since the solutions of (C) are stable in L^1 it is not immediately clear whether such algorithms are useful for approximating (C).

Gannon [11] introduced an adaptive finite-element method for parabolic differential equations based on theory for elliptic equations.

Our algorithm was also motivated by the work of Sanders [19], Douglas [7], and Douglas and Wheeler [8] on monotone finite-difference schemes with nonuniform grids. Sanders and Douglas provide convergence results for such methods with a fixed grid, while Sanders gives an error estimate of $O((h + \Delta t)^{1/2})$, where h is the largest meshsize and Δt is the timestep. Douglas and Wheeler introduce an algorithm that uses grids that may change from one timestep to the next, a true adaptive mesh method. They prove that the solutions of their algorithm converge to the entropy solution of the conservation law. They do not provide an error estimate. We compare their method with ours in the final section.

When Sanders, Douglas, and Douglas and Wheeler considered a nonuniform mesh, they interpreted their numerical solutions as piecewise constant in x, and they used a conservative finite-difference operator that is, in general, inconsistent everywhere. As a consequence, $\max(x_i - x_{i-1})/\Delta t$ must be bounded to achieve stability in time in [8]. Since our method can have arbitrarily large spatial increments, depending on the smoothness of the solution and the nonlinearity of f, we made a different choice. Our method interprets the solution of the numerical problem as a piecewise linear function, to take advantage of at least some smoothness in the solution. We also use a consistent, but nonconservative, difference scheme. (Because of the details of the scheme, it is still conservative near shocks.) Section 4 contains partial results that bound the mass error in a reasonable way.

Adaptive numerical methods for hyperbolic conservation laws have previously been considered by Oliger [18] and his students. Hedstrom and Rodrigue [13] is a survey of some of the techniques that are used. Bolstad [2] presents a framework for methods in one space dimension. His schemes incorporate

locally varying, recursively defined, space and time increments. He uses Richardson extrapolation to estimate the local truncation error of the finite-difference scheme. The estimated truncation error determines the local grid size. Berger [1] extends Bolstad's work to two dimensions. Among other things, she deals with the strictly two-dimensional problems of shock capturing, subgrid orientation, and overlapping grids.

The stability analyses in these papers are L^2 analyses, and their motivation seems to be the accurate approximation of smooth solutions of systems of linear conservation laws. Our motivation is the solution of nonlinear problems with shocks, and, in dealing first with the scalar equation, we use a L^1 analysis. Instead of finding a general framework for the such methods, we use a specific finite-difference operator and mesh selection criteria.

Oliger and his students employ locally varying timesteps as well as spatial mesh increments; we do not. We could have used locally timesteps, but we were not able to prove stability and convergence results for these methods. Since asymptotic improvement in convergence rates can be exhibited with fixed (small) timesteps, local timesteps were not used. When locally varying timesteps are used, our algorithm's implementation is very close to Bolstad's.

The rest of the paper is as follows. Notation and preliminary results complete this section. Section 2 briefly describes the finite-difference operator used here. Section 3 presents the mesh selection algorithm, and proves certain useful properties about the resulting mesh. Section 4 contains proofs of the nonlinear stability of the algorithm. Section 5 shows that a variant of our methods converges for solutions of linear problems. Section 6 details our implementation of the algorithm, and Section 7 describes our computational results.

The BV seminorm of a function u is defined as $|u|_{BV(\mathbb{R})} = \int_{\mathbb{R}} |u'(x)|\,dx$, where the integrand is interpreted as a finite measure. The set of all such

functions is denoted by $BV(\mathbf{R})$. If u is in $BV(\mathbf{R})$, then there exist two bounded functions, u^+ and u^- such that $u = u^+ + u^-$ and u^+ is nondecreasing, u^- is nonincreasing. We define $u^t = u^+ - u^-$, the total variation function of u.

Throughout the paper we assume the normalization that $\|f'\|_{L^\infty} \le 1$. This can always be achieved with a change of the time scale, and is used only for convenience in stating stability conditions.

2. The Finite-difference Scheme.

We use a standard upwind-difference scheme to advance the approximate solution from time t^n to t^{n+1}. We are given a suitable mesh, chosen by the rules in the next section, to represent the solution at time t^n. It is characterized by the meshpoints x_i^n and the values of the approximate solution U_i^n at those meshpoints. We interpret these points as determining a continuous piecewise linear approximation to $u(x,t^n)$. A estimate of the solution at time t^{n+1} is calculated by

$$\frac{\overline{U_i^{n+1}} - U_i^n}{\Delta t} + \frac{f^+(U_i^n) - f^+(U_{i-1}^n)}{h_i^n} + \frac{f^-(U_{i+1}^n) - f^-(U_i^n)}{h_{i+1}^n} = 0 \qquad (2.1)$$

for all x_i^n (except the two endpoints of the interval). We have decomposed f into its increasing (f^+) and decreasing (f^-) parts. If f is monotone increasing or decreasing then this method is an upwind difference method. A similar scheme has been used by Engquist and Osher[10].

The linear interpolant of the values $\overline{U_i^{n+1}}$ at the points x_i^n is a function we call $\overline{u_h}$. The mesh selection algorithm of the next section, when applied to $\overline{u_h}$, gives us a new mesh x_i^{n+1} and function values U_i^{n+1} for the approximate solution of (C) at time t^{n+1}.

This process is repeated until $t^{n+1} = T$.

3. The Mesh Selection Algorithm.

This section describes our mesh selection algorithm. In our implementation, the mesh at time t^{n+1} is built from the mesh at time t^n, but the method of approximation is general and applies to any function with reference to a time-stepping procedure. We present the algorithm here in its general form.

Our method of mesh selection is similar to well known ones for adaptive linear approximation[3]. The mesh approximately equidistributes an estimate of part of the error incurred by the finite difference scheme, thus following methods used in static problems [4] and other evolution equations[6]. The method presented here was designed for problems with shocks, while previous methods were designed for possibly nonlinear problems with smooth solutions.

The problems in [6] succumbed to an L^2 analysis, while the conservation laws that are the target of this method are stable only in L^1. The mesh selection algorithm therefore chooses a mesh that is "right" for L^1. Our specific choice of the mesh will allow us to prove the stability results of section 4, an important goal.

Let u be any bounded function defined on $[a,b]$ that is constant outside of $[a,b]$, and let ε be a small parameter. Choose the mesh according to the following algorithm:

ALGORITHM M: This algorithm chooses meshpoints at which to approximate an arbitrary function u defined on $[a,b]$.

1. The meshpoints consist only of the points a and b and the centers of admissible intervals. Admissible intervals are defined by (2) and (3) below.
2. The interval $[a,b]$ is an admissible interval.
3. For any admissible interval I, let $3I=\{x \mid \text{dist}(x,I)=\inf_{y\in I}|x-y| < |I|\}$. If $|I| \geq \varepsilon$ and

$$|I| \int_{3I} |u_{xx}| + |f''(u)| u_x^2 \, dx \geq \varepsilon, \tag{3.1}$$

then the left and right halves of I are admissible intervals. The above integral is finite if u_{xx} is a finite measure; it is to be interpreted as ∞ otherwise. Note that $3I$ is an *open* interval.

When this algorithm is used for our adaptive method, $\Delta t = \varepsilon/4$, so that $\Delta t \leq \min_{i,n}(x_i^n - x_{i-1}^n)$.

A *minimal interval* is an admissible interval that contains no proper admissible subintervals. It is clear that the width of any given admissible interval is $2^{-k}(b-a)$ for some nonnegative k. Also, $|I| \geq \varepsilon/2$. It follows that the mesh is a subset of

$$S_\varepsilon = \{a + m\, 2^{-k}(b-a) \mid m = 0, 1, \cdots, 2^k, \text{and} (b-a)2^{-k} \geq \varepsilon/4 > (b-a)2^{-k-1}\}$$

Some lemmas about the structure of the mesh generated by this algorithm follow.

LEMMA 3.1. *If A and B are two adjacent minimal intervals, then*

$$\frac{1}{2} \leq \frac{|A|}{|B|} \leq 2.$$

Proof. We let $|B| > 2|A|$, and derive a contradiction. Since the width of any admissible interval is $(b-a)2^{-k}$ for some $k \geq 0$, $|B| \geq 4|A|$. Consider now the admissible interval from which A was derived by Step 3, and call it C. Since C was divided into two admissible subintervals, the integral in Step 3 is greater than ε. However, $|B| \geq 2|C|$ and $3C \subset 3B$, so that the corresponding integral for B must also be greater that ε, a contradiction to the minimality of B.

///

Except for the two points a and b, the set of meshpoints has the natural structure of a *tree*. The point $(a+b)/2$ is the *root* of the tree. You can also think of the interval $[a,b]$ as the root of the tree. (Since there is a one-to-one

correspondence between the meshpoints and the set of admissible intervals, we will describe the structure of the mesh equivalently in terms of intervals or meshpoints.) If an interval I is divided into two admissible subintervals by Step 3, then these two intervals are the left and right *children* of I, and I is their *parent*. A meshpoint with no children (the center of a minimal interval) is a *leaf*.

LEMMA 3.2. *Every second meshpoint is a leaf.*

Proof. The statement of the lemma is true after Step 2, and it is left invariant by Step 3.

///

This implies the obvious result that there is a unique covering of $[a,b]$ with minimal intervals.

LEMMA 3.3. *If the set of meshpoints is* $\{x_i\}$ *with* $h_i = x_i - x_{i-1} \geq 0$, *then* $1/2 \leq h_i / h_{i+1} \leq 2$.

Proof. If x_i is a leaf, then $h_i = h_{i+1}$. If x_{i-1} and x_{i+1} are leaves, then the result follows from Lemma 3.1.

///

The linear interpolant u_ε of u is defined by requiring that $u_\varepsilon(x_i) = u(x_i)$ for all x_i, and that u_ε be a linear function between the meshpoints so that u_ε is continuous on $[a,b]$. The function u_ε has the following approximation properties (cf. [3]).

LEMMA 3.4.

$$\|u - u_\varepsilon\|_{L^1([a,b])} \leq \frac{\varepsilon}{32}|b-a| + \sum_{\substack{I \text{ minimal} \\ |I|<\varepsilon}} \|u - u_\varepsilon\|_{L^1(I)} \tag{3.2}$$

Proof. If $I = (x_{i-1}, x_{i+1})$ is minimal and $|I| \geq \varepsilon$ then Step 3 implies that $|I| \int_I |u_{xx}|\, dx \leq \varepsilon$ The $L^1(I)$ error for linear interpolation at the points x_{i-1}, x_i,

and x_{i+1} is bounded by

$$\frac{|I|^2}{32}\int_I |u_{xx}|\,dx \le \frac{|I|}{32}\varepsilon .$$

Summing over all I gives the first term in the expression. The second term contains all the intervals not yet considered.

///

There are two interesting cases when the second term is known to be small.

LEMMA 3.5.

(a) *If u is a continuous, piecewise linear function such that u_x is discontinuous only at the points in S_ε, then the second term in (3.2) is zero.*

(b) *If u is in $BV(\mathbf{R})$ then the second term in (3.2) can be bounded by $1/2\,\|u\|_{BV(\mathbf{R})}\varepsilon$. (Here we assume, without loss of generality, that $u(x) = \lim_{h\to 0}\frac{1}{2h}\int_{x-h}^{x+h} u(t)\,dt$ for all x.)*

Proof.

(a) Since u is linear on each half of the minimal intervals I with $|I| < \varepsilon$, $u_\varepsilon = u$ on these intervals.

(b) It is an exercise to show that $\|u_\varepsilon - u\|_{L^1(I)} \le 1/2\,\varepsilon\|u\|_{BV(I)}$. Since u is in $BV(\mathbf{R})$, we may add these individual bounds to obtain the lemma.

///

A few other stability properties will be used in the sequel.

LEMMA 3.6. $\|u_\varepsilon\|_{BV([a,b])} \le \|u\|_{BV([a,b])}$, *and* $\|u_\varepsilon\|_{L^\infty([a,b])} \le \|u\|_{L^\infty([a,b])}$.

Proof. It is clear that linear interpolation has these properties.

///

4. Stability Properties.

The numerical method presented here has several stability properties that mimic those for conservation laws. In brief, solutions of the numerical method satisfy a maximum principle, are total variation diminishing, and are stable in time. Although the numerical scheme is not conservative, one can bound the mass error for most problems.

Boundary effects will be analyzed in the following way. Outside of $[a,b]$ the mesh will be extended so that all the mesh intervals to the left of a are of the same width as the mesh interval immediately to the right of a, denoted by h_a. A similar extension will be made to the right of b. We assume that the function U_i^n is constant to the right and to the left of $[a,b]$. The mapping from U_i^n to U_i^{n+1} is now divided into three parts: the finite-difference scheme (2.1) transforms U_i^n on the extended mesh to $\overline{U_i^{n+1}}$; the values at the points a and b are reset to their original values; and the remeshing procedure, applied to the mesh values in $[a,b]$, yields U_i^{n+1}.

For the method to work properly, some criteria is needed to decide when a shock or other disturbance is getting too close to the boundary of the interval. Throughout, we will assume that the minimal intervals adjacent to the boundary points a and b have diameters greater than ε. This is a simple and effective criteria. If this criteria is in danger of being violated, the interval $[a,b]$ is to be enlarged.

We will follow the development of [8] for many of our proofs.

The following simple lemma will have important applications.

LEMMA 4.1. *If g is f^+, f^- or f^t, and u is the linear interpolant of the points (x_i^n, U_i^n) then*

$$\left| \frac{g(U_{i+1}^n) - g(U_i^n)}{h_{i+1}^n} - \frac{g(U_i^n) - g(U_{i-1}^n)}{h_i^n} \right| \le \int_{x_{i-1}^n}^{x_{i+1}^n} |u_{xx}| + |f''(u)| u_x^2 \, dx . \quad (4.1)$$

Note that this is multiple of the quantity that we use as a subdivision criteria.

Proof. Since g has a bounded, piecewise continuous second derivative,

$$g(U_{i+1}^n) = g(U_i^n) + h_{i+1}^n g'(U_i^n) \frac{(U_{i+1}^n - U_i^n)}{h_{i+1}^n} + \int_{(x_i^n, x_{i+1}^n)} (x_{i+1}^n - x) g(u(x))_{xx} \, dx .$$

Now, $g'(u)$ is bounded by 1, and for $x \in (x_i^n, x_{i+1}^n)$, $g(u)_{xx} = g''(u) u_x^2$, since $u_{xx} = 0$. Also, $|g''|$ is either $|f''|$ or 0. The previous equation and a similar one for $g(U_{i-1}^n)$ can be rearranged and summed to yield

$$\left| \frac{g(U_{i+1}^n) - g(U_i^n)}{h_{i+1}^n} - \frac{g(U_i^n) - g(U_{i-1}^n)}{h_i^n} \right| \le \int_{(x_{i-1}^n, x_{i+1}^n)} |u_{xx}| + |f''(u)| u_x^2 \, dx .$$

Here we have expressed the difference of the left and right derivatives of u at x_i^n as the integral of the second derivative "delta" measure.

///

The sharper bound

$$\left| \frac{f^+(U_{i+1}^n) - f^+(U_i^n)}{h_{i+1}^n} - \frac{f^+(U_i^n) - f^+(U_{i-1}^n)}{h_i^n} \right| + \left| \frac{f^-(U_{i+1}^n) - f^-(U_i^n)}{h_{i+1}^n} - \frac{f^-(U_i^n) - f^-(U_{i-1}^n)}{h_i^n} \right|$$

$$\le \int_{(x_{i-1}^n, x_{i+1}^n)} |u_{xx}| + |f''(u)| u_x^2 \, dx . \quad (4.2)$$

may be derived by noting that $f = f^+ + f^-$, and if the first or second derivative of $f^+(u(x))$ is nonzero at x, then the first and second derivatives of $f^-(u(x))$ are zero there.

THEOREM 4.1. *For all $n > 0$, $\sup_i U_i^n \le \sup_i U_i^0$.*

Proof. By Equation (2.1),

$$\overline{U_i^{n+1}} = U_i^n - \Delta t \left[\frac{f^+(U_i^n) - f^+(U_{i-1}^n)}{h_i^n} + \frac{f^-(U_{i+1}^n) - f^-(U_i^n)}{h_{i+1}^n} \right]$$

Since f^+ is increasing, f^- is decreasing, $\|f^t\|_{Lip} \le 1$, and $\Delta t / h_i^n \le 1$, the preceding equation yields

$$\begin{aligned}\overline{U_i^{n+1}} &\le U_i^n - \Delta t \left[\frac{f^+(U_i^n) - f^+(U_{i-1}^n \vee U_i^n \vee U_{i+1}^n)}{h_i^n} + \frac{f^-(U_{i-1}^n \vee U_i^n \vee U_{i+1}^n) - f^-(U_i^n)}{h_{i+1}^n} \right] \\ &\le U_i^n + f^t(U_{i-1}^n \vee U_i^n \vee U_{i+1}^n) - f^t(U_i^n) \\ &\le U_{i-1}^n \vee U_i^n \vee U_{i+1}^n.\end{aligned}$$

Therefore, the finite-difference scheme satisfies a maximum principle. Resetting the values of U_i^n at a and b does not violate a maximum principle. Lemma 3.6 states that the subsequent transformation to U^{n+1} satisfies a maximum principle.

///

THEOREM 4.2. *For all $n>0$, $\|U^n\|_{BV(\mathbb{R})} \le \|U^0\|_{BV(\mathbb{R})}$.*

Proof. If we let $\delta f_i^+ = f^+(U_i^n) - f^+(U_{i-1}^n)$, then

$$\overline{U_i^{n+1}} - \overline{U_{i-1}^{n+1}} = U_i^n - U_{i-1}^n - \Delta t \left[\frac{\delta f_i^+}{h_i^n} - \frac{\delta f_{i-1}^+}{h_{i-1}^n} + \frac{\delta f_{i+1}^-}{h_{i+1}^n} - \frac{\delta f_i^-}{h_i^n} \right].$$

Because f^+ and f^- are monotone, $f^t = f^+ - f^-$, $\|f^t\|_{Lip} \le 1$ and $\Delta t / h_i^n \le 1$, we can take absolute values, and sum:

$$\begin{aligned}\sum_i |\, \overline{U_i^{n+1}} - \overline{U_{i-1}^{n+1}} \,| &\le \sum_i |\, U_i^n - U_{i-1}^n - \frac{\Delta t}{h_i^n}[f^t(U_i^n) - f^t(U_{i-1}^n)] \,| + \Delta t \sum_i \left[\left| \frac{\delta f_{i-1}^+}{h_{i-1}^n} \right| + \left| \frac{\delta f_{i+1}^-}{h_{i+1}^n} \right| \right] \\ &= \sum_i |\, U_i^n - U_{i-1}^n \,|.\end{aligned}$$

Thus the mapping $U^n \to \overline{U^n}$ is total variation diminishing. Resetting the values at the endpoints does not increase the total variation. Since the remeshing process is variation diminishing, by Lemma 3.6, the theorem is proved.

///

THEOREM 4.3. $\|U^{n+1} - U^n\|_{L^1([a,b])} \le (3/2\|U^0\|_{BV(\mathbb{R})} + (b-a)/8)\Delta t + 2\Delta t\,\varepsilon$.

Proof. First, since the mesh is graded,

$$\begin{aligned}
\|\overline{U^{n+1}} - U^n\|_{L^1([a,b])} &\le \Delta t \sum_i |\overline{U_i^{n+1}} - U_i^n| \, \frac{(h_i^n + h_{i+1}^n)}{2} \\
&\le \frac{\Delta t}{2} \sum_i \left[|f^+(U_i^n) - f^+(U_{i-1}^n)| \left(1 + \frac{h_{i+1}^n}{h_i^n}\right) + |f^-(U_{i+1}^n) - f^-(U_i^n)| \left(1 + \frac{h_i^n}{h_{i+1}^n}\right) \right] \\
&\le \frac{3\Delta t}{2} \sum_i |f^t(U_i^n) - f^t(U_{i-1}^n)| \\
&\le \frac{3\Delta t}{2} \|U^0\|_{BV(\mathbb{R})}.
\end{aligned}$$

When the boundary values are restored, we commit an error at the left endpoint of at most $\Delta t \, |U_1^n - U_0^n|$. Under the assumption that the width of the minimal interval adjacent to a is bigger than ε, Step 3 of the mesh construction allows us to bound the error by $\Delta t \, \varepsilon$.

Finally, we may apply Lemmas 3.4 and 3.5 to yield

$$\|U^{n+1} - \overline{U^{n+1}}\|_{L^1([a,b])} \le (b-a) \, \Delta t / 8.$$

///

Because a consistent but nonconservative scheme is used for the time stepping, there will, in general, be some mass balance error. This error can be bounded by the following theorem.

THEOREM 4.4. *Assume* $[a,b] = \bigcup_j S_j \cup \bigcup_j R_j$ *where* $m-1$ *disjoint regions* R_j *are made up of minimal intervals of width less that* ε, *and are surrounded by* m *regions* S_j *that are covered by minimal intervals whose width is greater than* ε. *Assume that there are no more than* k *minimal intervals in* $\bigcup_j S_j$. *Then*

$$\left| \int_{\mathbb{R}} \overline{U^{n+1}} - U^n \, dx - \Delta t \, (f(u(a)) - f(u(b))) \right| \le \Delta t \, (2k\varepsilon + (2m\varepsilon \|U^n\|_{BV(\mathbb{R})})^{1/2}) \quad (4.3)$$

Proof. Using the notation of Theorem 4.2, we have

$$\frac{1}{\Delta t} \int_{\mathbb{R}} \overline{U^{n+1}(x)} - U^n(x) \, dx$$

$$= -\sum_i \left(\frac{\delta f_i^+}{h_i^n} + \frac{\delta f_{i+1}^-}{h_{i+1}^n}\right)\left(\frac{h_i^n + h_{i+1}^n}{2}\right).$$

$$= \frac{1}{2}[f^+(u(a)) + f^-(u(a)) - (f^+(u(b)) + f^-(u(b)))] - \sum_i \left(\frac{\delta f_i^+}{h_i^n} h_{i+1}^n + \frac{\delta f_{i+1}^-}{h_{i+1}^n} h_i^n\right)$$

$$= \frac{1}{2}(f(u(a)) - f(u(b))) - \sum_i \frac{\delta f_i^+}{h_i^n} h_{i+1}^n - \sum_i \frac{\delta f_{i+1}^-}{h_{i+1}^n} h_i^n.$$

We will show that the first sum minus $1/2(f^+(u(a)) - f^+(u(b)))$ can be bounded as in the statement of the theorem.

Pick a particular S_j with left and right endpoints x_l and x_r and consider the sum

$$\sum_{x_i^n = x_l}^{x_r} \frac{\delta f_i^+}{h_i^n} h_{i+1}^n. \tag{4.4}$$

Since every minimal interval contained in S_j is more than ε wide, Step 3 of Algorithm M and Lemma 3.5 show that (4.4) is equal to

$$\sum_{x_i^n = x_l}^{x_r} \frac{\delta f_{i+1}^+}{h_{i+1}^n} h_{i+1}^n + M = \sum_{x_{i+1}}^{x_{r+1}} \frac{\delta f_i^+}{h_i^n} h_i^n + M.$$

where $|M|$ is less than $\varepsilon(r - l + 1)$.

Any interval R_j consists of minimal intervals of width less than ε. Perforce, all the intervals must have the same width. Thus, if x_l and x_r are the left and right boundaries of R_j,

$$\sum_{x_i^n = x_{l+1}}^{x_{r-1}} \frac{\delta f_i^+}{h_i^n} h_{i+1}^n = \sum_{x_i^n = x_{l+1}}^{x_{r-1}} \frac{\delta f_i^+}{h_i^n} h_i^n$$

Matching these results gives

$$\sum_i \frac{\delta f_i^+}{h_i^n} h_{i+1}^n = \sum_i \frac{\delta f_i^+}{h_i^n} h_i^n + M + \sum_{S_j = (x_l, x_r)} (\delta f_{r+1}^+ - \delta f_l^+). \tag{4.5}$$

Here, $|M| \le 2k\varepsilon$, since $\bigcup S_j$ is covered by k minimal intervals. The remark following Lemma 4.1 implies that the same bound holds when the terms incorporating f^- are also included. The first sum on the right of (4.5) is equal to

$f^+(u(b)) - f^+(u(a))$.

For the interval S_j, assume $\left|\frac{\delta f^+_{r+1}}{h^n_{r+1}} - \frac{\delta f_l^+}{h_l^n}\right| = K_j$, for some positive K_j. Then, without loss of generality, we may assume that $\frac{\delta f^+_{r+1}}{h^n_{r+1}} \geq \frac{K_j}{2}$. Lemma 4.1 and Step 3 of the mesh construction imply that $\left|\frac{\delta f^+_{i+1}}{h^n_{i+1}} - \frac{\delta f_i^+}{h_i^n}\right| \leq 1$ if $x_i^n \in S_j$. Also, $h_i^n \geq \varepsilon/2$ for $x_i^n \in S_j$. Therefore,

$$\begin{aligned} \|f^+(U^n)\|_{BV(S_j)} &\geq \varepsilon/2 \sum_{i=0}^{K_j/2} (K_j/2 - i) \\ &\geq \varepsilon K_j^2/8. \end{aligned}$$

Thus, the last sum in (4.5) is bounded by

$$\begin{aligned} \varepsilon/2 \sum_j K_j &\leq (2\varepsilon m)^{1/2} (\varepsilon/8 \sum_j K_j^2)^{1/2} \\ &\leq (2\varepsilon m \, \|f^+(U^n)\|_{BV(\mathbb{R})})^{1/2} \end{aligned}$$

Since $\|f^+(U^n)\|_{BV(\mathbb{R})} + \|f^-(U^n)\|_{BV(\mathbb{R})} \leq \|U^n\|_{BV(\mathbb{R})}$, the theorem is proved.

///

It can be shown that, for piecewise smooth functions, Algorithm M generates fewer than $K\varepsilon^{-1/2}$ meshpoints, for some K (cf. de Boor[3]). If this bound is assumed throughout the calculation of u_ε, Theorem 4.4 bounds the mass error at a fixed time T by a multiple of $\varepsilon^{1/2}$ or $\Delta t^{1/2}$. The expected rate of convergence of the method is $O(\Delta t^{1/2})$.

Unfortunately, we have not been able to calculate an a priori bound on the number of meshpoints in a mesh generated by Algorithm M. If the function u_ε is rough enough, ε^{-1} meshpoints may be needed to approximate it. The numbers k and m can be calculated during the course of the algorithm in a negligible amount of computer time, however, and (4.3) may be used as an a posteriori bound.

Theorem 4.4 does not consider the mass error generated by the adjustment of the function values at the points a and b. Theorem 4.3 shows that this error is $O(\Delta t)$ if the minimal intervals next to a and b have width greater than ε.

Estimate (4.3) does not include the mass error caused by the remeshing process. Such an error occurs when admissible intervals in the mesh at time t^n are no longer needed, and do not appear in the mesh at time t^{n+1}. Lemma 3.4 is sharp for any given time step; the mass error for a time step of size Δt can be comparable to Δt. Using this bound over the interval $[0,T]$ gives an estimate of an $O(1)$ mass error, a totally unacceptable result.

Computational experience with piecewise smooth solutions of the differential equation has shown that over the interval $[0,T]$ there is a constant C such that fewer than $C2^k$ intervals of width $2^{-k}(b-a)$ are removed from the mesh in $[0,T]$. That is, a minimal interval is not subsumed into a larger minimal interval for a period of time proportial to the its width. If this property holds, an argument similar to that of Lemma 3.4 shows that the mass error due to mesh changes on $[0,T]$ is bounded by $C\Delta t\log(\Delta t^{-1})$ for some C. We therefore have the following theorem.

THEOREM 4.5. *For an interval* $[0,T]$, *assume that there are constants* C_1, C_2, *and* C_3 *such that for any* $\varepsilon>0$

(a) *there are no more than* $C_1\varepsilon^{-1/2}$ *minimal intervals of width bigger than* ε *for any* t^n *in* $[0,T]$,

(b) *there are never more than* C_2 *disjoint regions covered by minimal intervals of width less than* ε, *and*

(c) *at most* $C_3 2^k$ *admissible intervals of width* $(b-a)2^{-k}$ *are removed from the mesh in* $[0,T]$.

Then there exists a constant C, *depending on* C_1, C_2, C_3, *and* $\|u\|_{BV(\mathbb{R})}$, *such that*

the mass error of the numerical approximation (2.1) can be bounded by $C\varepsilon^{\frac{1}{2}}$.

///

The conditions (a), (b), and (c) can easily be checked *a posteriori*.

5. Convergence for Linear Problems.

In this section we prove that solutions of the linear problem

$$\begin{aligned} u_t + \alpha u_x &= 0, \quad x \in \mathbf{R},\ t>0, \\ u(x,0) &= u_0(x), \quad x \in \mathbf{R} \end{aligned} \tag{5.1}$$

are approximated well by variants of our adaptive mesh algorithm. If the initial data is in $BV(\mathbf{R})$, *a posteriori* bounds will be given on the error, and if the initial data is slightly smoother, with $\frac{du_0}{dx} \in BV(\mathbf{R})$, *a priori* bounds are available. Formal justification of the mesh selection criteria used in Algorithm M is also given.

Consider first when $u_0 \in BV(\mathbf{R})$. Our algorithm is as follows:

1. Pick $\varepsilon>0$, and let $u_0^\varepsilon = \psi_\varepsilon * u_0$, where $\psi_\varepsilon(x)$ is $\varepsilon^{-1/2}$ when $|x| \le \varepsilon^{1/2}/2$ and 0 otherwise.
2. Using u_0^ε as the initial data, follow the algorithm in Section 2 to find u^ε. This will be our approximation to u.

It is easily shown that $\|\frac{du_0^\varepsilon}{dx}\|_{BV(\mathbf{R})} \le \varepsilon^{-1/2}\|u_0\|_{BV(\mathbf{R})}$, and $\|u_0 - u_0^\varepsilon\|_{L^1(\mathbf{R})} \le \varepsilon^{1/2}\|u_0\|_{BV(\mathbf{R})}$.

The following theorem applies to such problems.

THEOREM 5.1. *If, at each timestep, there are never more than* $C_1\varepsilon^{-1/2}$ *minimal intervals of width greater than* ε*; and if the total* $L^1([a,b])$ *error due to mesh changes is* $O(\varepsilon^{1/2})$*, then*

$$\|u(t) - u^\varepsilon(t)\|_{L^1([a,b])} \le C_2(t+1)\varepsilon^{1/2}(\|u_0\|_{BV(\mathbf{R})} + 1) \tag{5.2}$$

Furthermore, the computational complexity of the scheme is $O(\varepsilon^{-3/2})$

Theorem 5.1 gives *a posteriori* estimates of the error and computational complexity of the scheme. Our computational experience suggests that the conditions of the theorem are always satisfied. We suspect that the structure of solutions of the conservation laws ensures this. The computational bound is better than for standard monotone methods; for these methods the error is $O(\Delta t^{1/2})$ but the complexity is $O(\Delta t^{-2})$.

The proof of this theorem relies on a number of lemmas.

LEMMA 5.1. *If $u_x \in BV(\mathbb{R})$ and v is the linear interpolant of u on some mesh, $\{x_i\}$, then $\|v_x\|_{BV(\mathbb{R})} \le \|u_x\|_{BV(\mathbb{R})}$.*

Proof. Since $v_x(x) = \dfrac{1}{x_i - x_{i-1}} \displaystyle\int_{x_{i-1}}^{x_i} u_x(t)\,dt$ for $x \in (x_{i-1}, x_i)$, the result follows immediately.

///

LEMMA 5.2. *If U^n is generated by Algorithm M and (2.1), then*

$$\|U_x^n\|_{BV(\mathbb{R})} \le \|U_x^0\|_{BV(\mathbb{R})}.$$

Proof. Since (5.1) is linear, $V_i^n = \dfrac{U_i^n - U_{i-1}^n}{h_i}$ satisfies the same difference equation as U_i^n. Hence $\|\overline{V^{n+1}}\|_{BV(\mathbb{R})} \le \|V^n\|_{BV(\mathbb{R})}$. By the previous lemma, $\|V^{n+1}\|_{BV(\mathbb{R})} \le \|\overline{V^{n+1}}\|_{BV(\mathbb{R})}$. The result then follows by induction.

///

Our analysis views upwind finite-difference schemes for conservation laws in a seemingly new way. In particular, if U_i^n and $\overline{U_i^{n+1}}$ are interpreted as piecewise linear functions, then the equation (2.1) defining $\overline{U_i^{n+1}}$ is equivalent to the following algorithm:

To obtain $\overline{U_i^{n+1}}$, shift U_i^n to the right by $\alpha\Delta t$ (if α is positive) and interpolate the shifted function at the points x_i^n.

Note that error occurs only in the interpolation process; there is no error in the values of the $\overline{U_i^{n+1}}$ themselves if the values of the U_i^n are exact. This calculation is illustrated in Figure 1.

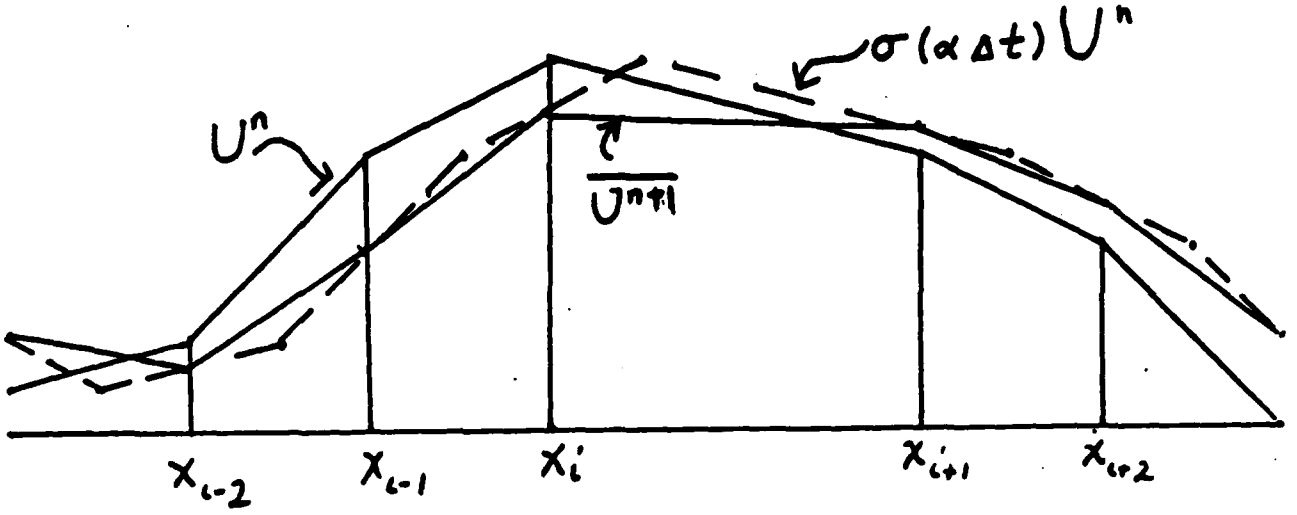


Figure 1. The error caused by time-stepping.

Proof of Theorem 5.1. We first define $\sigma(v)(x) = v(x - a\Delta t)$. If $e_n = \|u(n\Delta t) - U^n\|_{L^1([a,b])}$ then

$$\begin{aligned} e_n &\le \|u(n\Delta t) - \sigma(U^{n-1})\|_{L^1([a,b])} + \|\sigma(U^{n-1}) - U^n\|_{L^1([a,b])} \\ &= \|\sigma(u((n-1)\Delta t)) - \sigma(U^{n-1})\|_{L^1([a,b])} + \|\sigma(U^{n-1}) - U^n\|_{L^1([a,b])} \\ &= e_{n-1} + \delta_n \end{aligned}$$

where δ_n is the local error $\|\sigma(U^{n-1}) - U^n\|_{L^1([a,b])}$. We define $\delta U_i^n = (U_i^n - U_{i-1}^n)/h_i^n$. Now, from Figure 1,

$$\begin{aligned} \|\overline{U^{n+1}} - \sigma(U^n)\|_{L^1([a,b])} &= \sum_{x_i^n} \alpha\Delta t h_i^n \left(1 - \frac{\alpha\Delta t}{h_i^n}\right) |\delta U_i^n - \delta U_{i-1}^n| \qquad (5.3) \\ &\le 2\alpha\Delta t^2 \sum_{h_i^n < 2\Delta t} |\delta U_i^n - \delta U_{i-1}^n| \\ &\quad + \alpha\Delta t \sum_{h_i^n \ge 2\Delta t} h_i^n |\delta U_i^n - \delta U_{i-1}^n| \end{aligned}$$

By using Lemmas 5.1 and 5.2, one can bound the first term by

$$C\Delta t\, \varepsilon \|U_\varepsilon^n\|_{BV(\mathbb{R})} \le C\varepsilon^{1/2} \|u_0\|_{BV(\mathbb{R})} \Delta t$$

Because of Step 3 of Algorithm M, and our assumption about the number of intervals of width greater than ε, the second term can be bounded by

$$\alpha \Delta t \, \varepsilon \times C\varepsilon^{-1/2} = C\varepsilon^{1/2}\Delta t$$

Thus $\|\overline{U^{n+1}} - \sigma(U^n)\|_{L^1([a,b])} \le C\Delta t\, \varepsilon^{1/2}(\|u_0\|_{BV(\mathbb{R})} + 1)$.

Summing this expression over $n \le t/\Delta t$ gives one term of (5.2). We have assumed that $\sum_n \|\overline{U^{n+1}} - U^{n+1}\|_{L^1([a,b])} \le C\varepsilon^{1/2}$. As stated previously, $\|u_0 - u_0^\delta\| \le \varepsilon^{1/2}\|u_0\|_{BV(\mathbb{R})}$. Lemmas 3.4 and 3.5 show that $\|U^0 - u_0^\delta\|_{L^1([a,b])} \le \varepsilon(b-a)/32 + 1/2\varepsilon\|u_0\|_{BV(\mathbb{R})}$. The error bound is proved.

The complexity of the scheme is $O(\Delta t^{-1}N)$ where N is the maximum number of meshpoints in any mesh $\{x_i^n\}$, since the amount of work is linear in the number of meshpoints. By hypothesis, there are fewer than $C\varepsilon^{-1/2}$ minimal intervals of width greater than ε. Furthermore, since $\int_{3I} |u_{xx}|\,dx > 1$ if $I < \varepsilon$, there must be fewer than $3\|U_x^n\|_{BV(\mathbb{R})} \le 3\varepsilon^{-1/2}\|u_0\|_{BV(\mathbb{R})}$ minimal intervals of width less than ε. Because every second meshpoint is the center of a minimal interval, the theorem is proved.

///

We now consider the case when u is smoother. Let $L^{1,2}(\mathbf{R}) = \{u \in L^1(\mathbf{R}) \cap BV(\mathbf{R}) \mid u_x \in BV(\mathbf{R})\}$.

We define here a modification of Algorithm M for functions in $L^{1,2}(\mathbf{R})$.

Algorithm M′:

1. The points a and b are meshpoints. The center of every admissible interval (to be defined below) is a meshpoint.
2. The interval $[a,b]$ is an admissible interval.
3. If I is an admissible interval, $|I| \ge 4\Delta t$, $3I = \{x \mid \text{dist}(x,I) < |I|\}$, and

$$|I| \int_{3I} |u_{xx}| \, dx > \frac{\|u'\|_{BV(\mathbb{R})}}{(b-a)} (\Delta t)^2,$$

then the left and right halves of I are admissible intervals. As before, u_{xx} is to be interpreted as a measure.

This new mesh algorithm makes the smallest admissible intervals the same size as the "average" interval, i.e. when u_{xx} is bounded. Since $u \in L^{1,2}(\mathbb{R})$, the very small spatial and temporal mesh increments of the previous method are not needed. The following adaptive algorithm is therefore a true "smooth solution" algorithm.

Our new algorithm is as follows:

(a) Given $u_0(x)$, find x_i^0 using Algorithm M′ and choose U_i^0 using piecewise linear interpolation of u_0.

(b) For each timestep: Given meshpoints $\{x_i^n\}$ and function values $\{U_i^n\}$ defined at those meshpoints, calculate $\overline{U_i^{n+1}}$ using (2.1). (Equation (2.1) reduces to the upwind differencing scheme in this case.) Interpreting $\overline{U_i^n}$ as a continuous, piecewise linear function, use algorithm M′ to find a new mesh $\{x_i^{n+1}\}$ for $\overline{U_i^{n+1}}$, and define U_i^{n+1} on that mesh by linear interpolation of $\overline{U_i^{n+1}}$.

The following result holds.

THEOREM 5.2. *Let $u_0 \in L^{1,2}(\mathbb{R})$ and $u(x,t)$ be the solution of (5.1). If $n\Delta t = T$, and U^n is the solution of the adaptive mesh algorithm above, then*

$$\|u(t) - U^n\|_{L^1([a,b])} \le 3(T+1)\Delta t \, \|u_0'\|_{BV(\mathbb{R})}.$$

Proof As in Theorem 5.1,

$$\|\overline{U^{n+1}} - \sigma(U^n)\|_{L^1([a,b])} \le 2\alpha\Delta t^2 \sum_{h_i^n < 2\Delta t} |\delta U_i^n - \delta U_{i-1}^n| + \alpha\Delta t \sum_{h_i^n \ge 2\Delta t} h_i^n \, |\delta U_i^n - \delta U_{i-1}^n|$$

The first term is bounded by $2\alpha\Delta t^2 \|u_0'\|_{BV(\mathbb{R})}$ because of Lemmas 5.1 and 5.2. By using Step 3 of Algorithm M′, the second term can be bounded by

$$\alpha\Delta t\left(\sum_{h_i^n=2\Delta t} h_i^n\right)^{1/2}\left(\sum_{h_i^n=2\Delta t} h_i^n\,|\,\delta U_i^n-\delta U_{i-1}^n\,|^2\right)^{1/2}$$
$$\le \alpha\Delta t\,(b-a)^{1/2}\Delta t\,\frac{\|U_x^n\|_{BV(\mathbb{R})}}{(b-a)^{1/2}}$$
$$\le \alpha\Delta t^2\|u_0'\|_{BV(\mathbb{R})}.$$

Thus $\|\overline{U^{n+1}}-\sigma(U^n)\|_{L^1([a,b])}\le 3\alpha\Delta t^2\|u_0'\|_{BV(\mathbb{R})}$.

Finally, let S be the set of all minimal intervals in the mesh at time t^{n+1} that are not minimal at time t^n. Then, because of Lemma 3.4 and step 3 of Algorithm M′, the expression for the error in the trapezoid rule yields

$$\|\overline{U^{n+1}}-U^{n+1}\|_{L^1([a,b])}\le \sum_{\substack{I\in S\\ x_i^n\in I}}\frac{|I|^2}{32}\,|\,\delta\overline{U_i^n}-\delta\overline{U_{i-1}^n}\,|$$
$$\le \frac{\Delta t^2\|u_0'\|_{BV(\mathbb{R})}}{32(b-a)}\sum_{I\in S}|I|$$
$$\le \frac{\Delta t^2}{32}\|u_0'\|_{BV(\mathbb{R})}$$

Thus, $\delta_n\le 3\Delta t^2\|u_0'\|_{BV(\mathbb{R})}$. After calculating the error caused by the approximation of u_0, the theorem follows by induction.

///

The same analysis can be used formally for nonlinear problems: estimate the difference, now nonzero, between the values $\overline{U_i^{n+1}}$ and $S_{\Delta t}(U^n)(x_i^n)$, ($S_{\Delta t}$ advances the solution of (C) by time Δt) and then separately estimate the interpolation error as in Figure 1. The error in $\overline{U_i^{n+1}}$ is bounded formally by

$$\Delta t\int_{x_{i-1}^n}^{x_{i+1}^n}|\,f\,(U^n(x,t^n))_{xx}\,|\,dx+O(\Delta t^2). \tag{5.4}$$

The L^1 error on (x_{i-1}^n,x_{i+1}^n) is therefore bounded by $(h_i^n+h_{i+1}^n)/2$ times (5.4), bounded in turn by $C\Delta t$ times the integral in (3.1).

Thus, if $\|U_x^{n+1}\|_{BV(I)}\le C\|U_x^n\|_{BV(3I)}$ for each minimal interval I, the error caused by the nonlinearity is of the same order as the error caused by the

dissipation in the difference scheme. Thus, our mesh selection criteria achieves a balance between errors caused by nonlinearity and dissipation.

6. Implementation Details.

Our scheme was implemented using the programming language Pascal. The resulting programs were run on a VAX 11/780 with a floating point accelerator under the VMS 2.5 operating system and Pascal 1.2 compiler, and under the Berkeley Unix 4.1 operating system with its Pascal compiler. In this section, we describe the algorithms and data structures used in our implementation. We show that the integrals in (3.1) can be evaluated exactly, and hence that the stability results of the previous section hold without a separate theory describing the effects of numerical quadrature. We also estimate the computational complexity of the scheme.

The following steps advance the approximate solution from one timestep to the next. First, the finite-difference formula advances u_h from t^n to t^{n+1} on the mesh $\{x_i^n\}$. Secondly, the integrals in (3.1) are calculated for the mesh $\{x_i^n\}$. In our implementation, the integrals are first calculated over the (open) interval I instead of $3I$, and the integral over $3I$ is constructed when it is needed. Finally, the mesh $\{x_i^{n+1}\}$ is constructed. To do this, the union of the meshes $\{x_i^n\}$ and $\{x_i^{n+1}\}$ is built up by adding the appropriate meshpoints to $\{x_i^n\}$. The values of the integrals (3.1) are derived for the new meshpoints as they are introduced. A second pass is made to remove points in $\{x_i^n\}$ that are not needed in $\{x_i^{n+1}\}$.

Except for the two meshpoints a and b, the mesh is naturally organized as a ***binary tree***. This is because it is defined recursively by subdividing admissible intervals into two subintervals at each step. Each admissible interval (or, equivalently, the meshpoint at its center) is a *node* in the tree. The interval $[a,b]$ is the *root* of the tree. If an admissible interval is subdivided into two admissible subintervals, then these subintervals are the left and right ***children***

of the interval. A node without children is a ***leaf***, and corresponds to a minimal interval. Nodes that are not leaves are interior nodes. The ***parent*** of an interval is the admissible interval from which it was derived.

We first describe the information stored in a node corresponding to a meshpoint x_i^n at the center of an interval $I = (x_l, x_r)$. This information consists mainly of ***pointer variables*** and ***numeric variables***. A pointer variable either points to the beginning of a block of computer store allocated to a node, or it is ***nil***. A pointer has the value nil when it points to "nothing", i.e. it does not point to the memory location of a node. For example, the pointer "parent" would point to the beginning of the information for the parent node of x_i^n. (The parent pointer of the root would be nil.) The pointer "left.child" would point to its left child, etc. The numeric variables contain such information as the value U_i^n, the value of the integral (3.1) on the interval I, etc. Figure 2 contains the description of a node.

The function $\bar{f}$ is an auxiliary function introduced to calculate the integral (3.1) For all $\xi \in \mathbf{R}$, $\bar{f}'(\xi) = |f''(\xi)|$. it follows that

$$\int_{x_i^n}^{x_{i+1}^n} |U_{xx}^n| + |f''(U^n)|(U_x^n)^2\,dx = \frac{U_{i+1}^n - U_i^n}{h_i^n}(\bar{f}(U_{i+1}^n) - \bar{f}(U_i^n)), \qquad (6.1)$$

since U_{xx}^n is 0 for all $x \in (x_i^n, x_{i+1}^n)$. For each minimal interval, use (6.1) to calculate right.integral and left.integral; for each interior node, left.integral and right.integral are the values of this.integral for the left and right children of the node. For any node, calculate this.integral by adding the difference of the left and right derivatives of u at the meshpoint x_i^n to the sum of left.integral and right.integral.

In each node we have included pointers and numerical variables whose values can be calculated from already available information. For example, in each leaf node x_i^n we save h_i^n and $\Delta t / h_i^n$. These values are needed often in the

Node :

x { x_i^n }

u { U_i^n }

isaleaf { true if this node is a leaf }

left.child, right.child { pointers to $\frac{x_i^n+x_l}{2}$ and $\frac{x_i^n+x_r}{2}$ if this node is not a leaf }

parent { pointer to parent of this node }

isaleftchild { true if this node is a left child }

depth { distance, in nodes, from this node to the root node }

left.neighbor, right.neighbor { pointers to x_{i-1}^n and x_{i+1}^n }

left.boundary, right.boundary { pointers to x_l and x_r }

left.sibling, right.sibling { pointers to the nearest nodes at the same depth as x_i^n, to the left and right, respectively, of x_i^n }

this.integral, left.integral, right.integral { the integral (3.1) over (x_l,x_r), (x_l,x_i^n), and (x_i^n,x_r), respectively }

uxx { $\left|\frac{U_{i+1}^n-U_i^n}{h_{i+1}^n}-\frac{U_i^n-U_{i-1}^n}{h_i^n}\right|$ }

left.ux, right.ux { if this node is a leaf, $\frac{U_i^n-U_{i-1}^n}{h_i^n}$ and $\frac{U_{i+1}^n-U_i^n}{h_{i+1}^n}$ }

fluxplus, fluxminus, fluxbar { $f^+(U_i^n)$, $f^-(U_i^n)$, $\bar{f}(U_i^n)$ }

deltatoverh, hinverse, criticalvaluc { $\Delta t/(x_i^n-x_l)$, $1/(x_i^n-x_l)$, $\varepsilon/(x_r-x_l)$ }

Figure 2. Definition of a node corresponding to x_i^n in the interval (x_l,x_r).

scheme, and change only when the mesh is changed. Our experience with the scheme has shown that, on average, less than one node is added to or subtracted from the mesh at each time step. Thus, using the saved values reduces execution time significantly. Because a new mesh is constructed at each time step, we also require that the structural information necessary to derive a new mesh be readily available at each node. Again, extra storage reduces execution time. Since, for many problems, many fewer nodes are necessary to obtain a satisfactory error with this method than with standard first order methods, the extra storage requirements are not deemed critical.

The special nodes a and b are the left and right boundaries of $[a,b]$, the root node. In addition, supplemental meshpoints are added to the left and to

the right of the interval $[a,b]$ so that the pointers left.sibling and right.sibling will not be nil for any node in the tree headed by $[a,b]$. The complete data structure is shown in Figure 3. The lines in Figure 3 illustrate only the children and sibling relationships of the nodes.

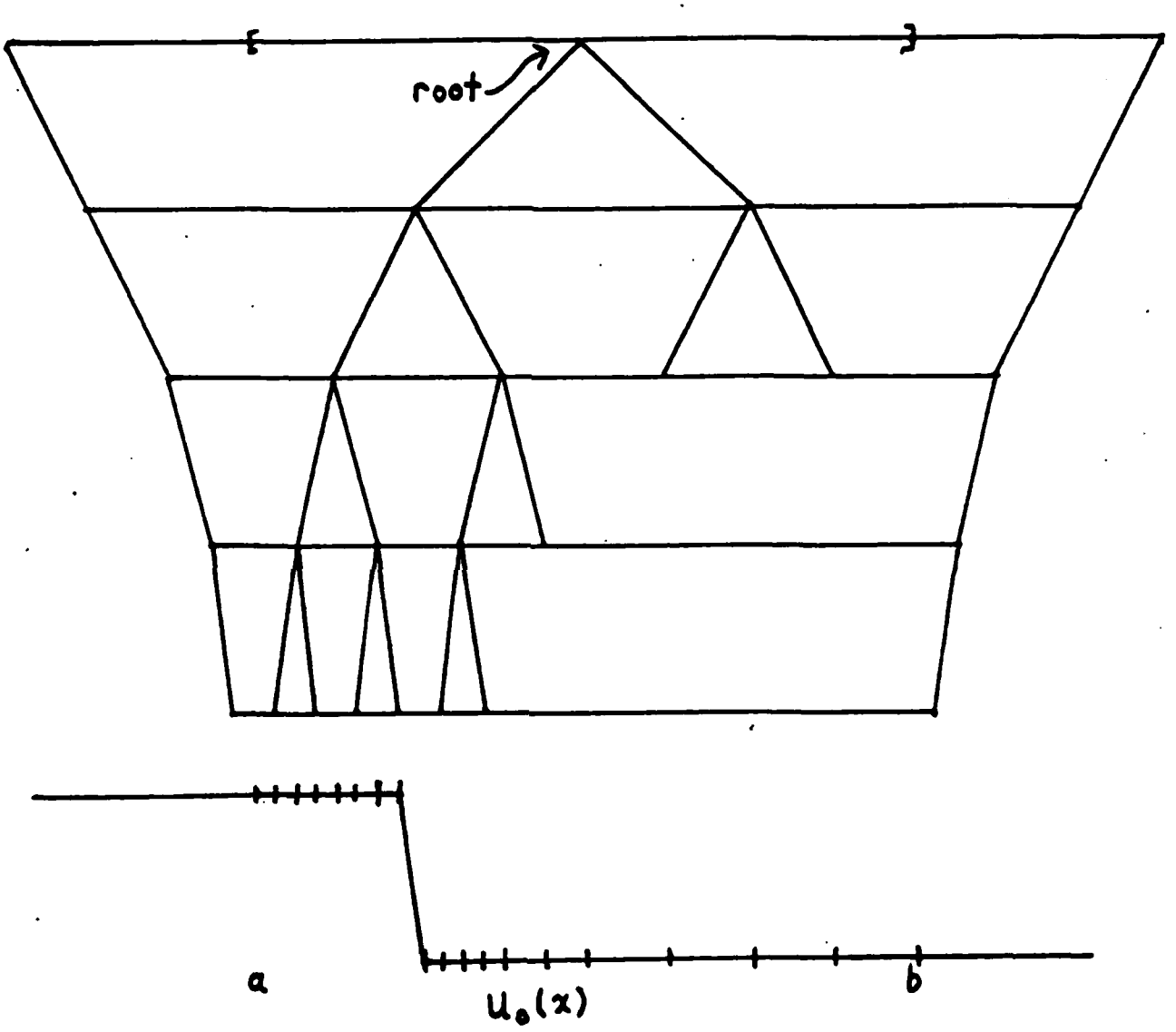


Figure 3. Sibling and child relationships for the mesh points.

Figure 4 presents the procedure *criteria* that calculates the nodal values needed to build the new mesh. On each invocation, criteria calculates the nodal

parameters for all nodes in the subtree headed by p. Indirection is indicated by a caret (^). The **with** statement means that all unqualified nodal variables belong implicitly to the node to which the pointer "p" points. For example, "right.neighbor^.left.ux" refers to p's right.neighbor's left.ux. Each time step, the nodal values for a and b are calculated first, and then criteria(root) is called. The following lemma outlines a proof that the procedure is correct.

```
procedure criteria(p: nodepointer);
begin
    with p^ do begin
        fluxbar := fbar(u);
        if isaleaf then begin
            left.ux        := (u - left.neighbor^.u) * hinverse;
            left.integral  := abs(left.ux * (left.neighbor^.fluxbar - fluxbar));
            right.ux       := (right.neighbor^.u - u) * hinverse;
            right.integral:= abs(right.ux * (right.neighbor^.fluxbar - fluxbar));
            uxx            := abs(left.ux - right.ux);
            this.integral  := left.integral + right.integral + uxx
        end else begin { p does not point to a leaf }
            criteria(left.child);
            criteria(right.child);
            right.integral:= right.child^.this.integral;
            left.integral  := left.child^.this.integral;
            uxx            := abs(right.neighbor^.left.ux - left.neighbor^.right.ux);
            this.integral  := left.integral + right.integral + uxx
        end { if isaleaf then ... else ... }
    end { with p^ do ... }
end; { criteria }
```

Figure 4. Procedure for calculating the integral (3.1).

LEMMA 6.1. ***Criteria(p) calculates the values of right.integral, left.integral, uxx, and this.integral for all nodes in the subtree headed by p. Furthermore, criteria(p) calculates left.ux and right.ux for all leaves in the subtree headed by p.***

Proof. First one shows that $\bar{f}(U_i^n)$ is calculated for the left and right boundary points of p before criteria(p) is called. This is easily proved by noting that it is true initially for the root node, and that if it is true for p's parent, then it is

true for p. The proof of the lemma then follows by induction on the height of the subtree headed by p.

///

For each interval I, the integral (3.1) over $3I$ will be used to test for two things:

1 If I is a leaf and $I \geq \varepsilon$, to see whether left and right children of I should be added to the set of meshpoints, according to the criteria of Step 3 of Algorithm M.

2 If I is not a leaf, to see whether the subtrees headed by the left and right children of I are still needed in the mesh.

These tests are done on separate passes of the tree. In the first pass, points are added to $\{x_i^n\}$ to obtain $\{x_i^n\} \cup \{x_i^{n+1}\}$. In the second pass, points are removed from the union to leave only $\{x_i^{n+1}\}$. Since the set of points that are tested on each pass are for the most part disjoint (only nodes with newly added children are tested twice), this two pass algorithm does not add too much to the arithmetic complexity of the scheme. A one pass algorithm could surely be devised.

An argument similar to to that in Lemma 3.1 shows that, in any mesh constructed using Algorithm M, a node's parent will always have adjacent left and right siblings. If $\{x_i^n\} \cup \{x_i^{n+1}\}$ is formed from $\{x_i^n\}$ by adding meshpoints in order of increasing depth, then this property also holds for all meshes intermediate to $\{x_i^n\}$ and $\{x_i^n\} \cup \{x_i^{n+1}\}$. Thus, we add nodes to the mesh in a depth first ordering, and calculate (3.1) for an interval that is a left child, for example, by adding p^.parent^.this.integral, p^.parent^.left.sibling^.right.integral, and p^.left.boundary^.uxx. Figure 5 illustrates the calculation. A similar expression holds for nodes that are right children.

The above calculation is the reason that each admissible interval has a

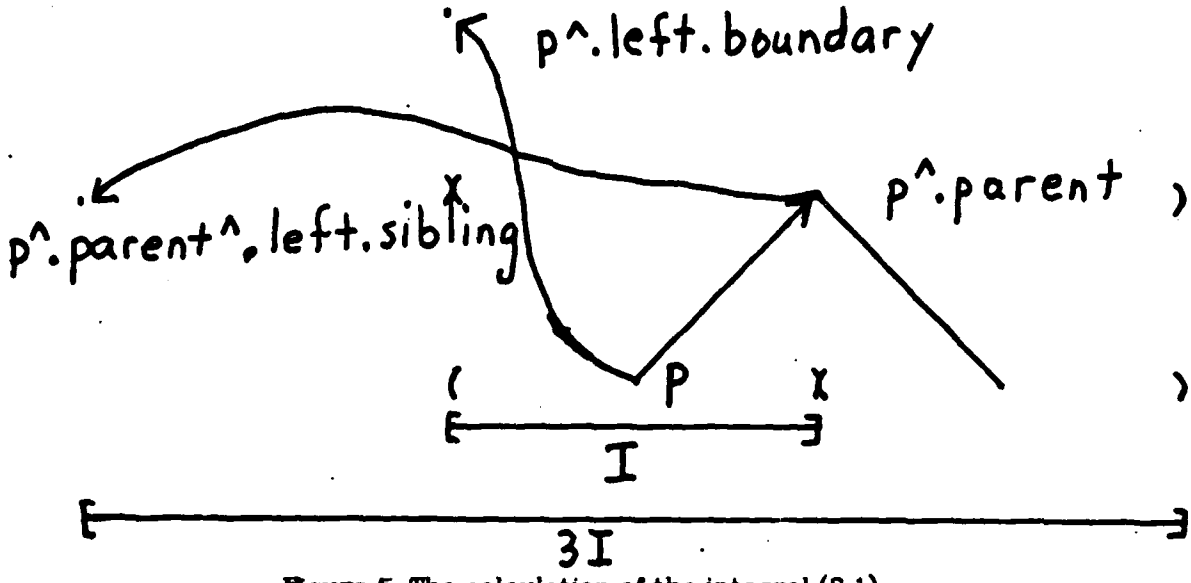


Figure 5. The calculation of the integral (3.1).

meshpoint at its center, and that the integral (3.1) is calculated over the left and right half of each admissible interval.

The second pass of the algorithm checks each node with children to see if the subtrees headed by that node's children are needed to approximate the function U_i^{n+1} well. If not, they subtrees are removed. The nodes are checked for subtree removal in a depth first order.

Note that our algorithm removes unnecessary subtrees at once, instead of removing one meshpoint at a time. In practice, however, we have not not observed the removal of any but the trivial subtrees consisting of only one node.

We use a simple and efficient memory management scheme. When a node is no longer needed and is removed from the tree, it is appended to a *free list* of nodes. When a node is needed for addition to the tree, it is taken from the free list if the list is not empty. If the free list is empty, a call to the operating system allocates enough storage for the new node.

The complexity of the scheme can be measured by counting the number of floating point operations that are done, on average, for each meshpoint x_i^n at

each time t^n. More operations are performed for leaf nodes than for interior nodes. We assume that an evaluation of f^+, f^-, or $\bar{f}$ requires two additions (or subtractions), two multiplications, and two comparisons. This would be the case, for example, if these functions were defined as piecewise quadratic functions over four intervals. Our assumptions yield

Complexity per meshpoint = 17 *Additions* + 9.5 *Multiplications* + 7 *Comparisons*.

This may be compared with a complexity estimate of 8 Additions, 4 Multiplications, and 4 Comparisons for a careful implementation of a uniform mesh algorithm with the same spatial difference operator and $\Delta t = h$. If an arbitrary variable spaced mesh is chosen every timestep, the standard algorithm will require 10 Additions, 5 Multiplications, 2 Divisions, and 4 Comparisons per meshpoint. This does not include whatever calculations are necessary to choose the mesh. Thus, the special placement of the meshpoints in our algorithm no more than doubles the work per meshpoint.

Nonarithmetic operations must be included in any complexity estimate of these algorithms. It is more difficult to quantify this nonarithmetic complexity, what may be considered "overhead." We give empirical results in the next section that show that the nonarithmetic overhead is the same for the fixed and adaptive mesh algorithms.

A more serious difficulty in comparing the efficiency of these algorithms is that the fixed mesh algorithm converges at different rates for differing fluxes f. We will find in the next section that for the problems for which the fixed mesh algorithm performs relatively well, the new algorithm compares poorly. The problems for which the fixed mesh algorithm performs poorly, however, are solved with surprising success by our algorithm.

7. Computational Results.

The Pascal implementation of our algorithm was run on two Digital Equipment Corporation VAX 11/780 computers with floating point accelerators under the VMS 2.5 and Berkeley Unix 4.1 BSD operating systems. Both programs use double precision (64 bit) floating point numbers. We only needed to change real constants from double precision to single precision notation to move the program from VMS to Unix; no other changes were needed.

We chose computational examples to highlight the strengths of our method as well as point out directions for improvements. The biggest omission was of problems with smooth solutions. (We did not implement the algorithm for smooth solutions given in section 5.) It is easily shown that for monotone uniform grid methods, the work expended to achieve an accuracy of δ at time T is proportional to δ^{-2}. When given a problem with a smooth solution, the present implementation of our method will achieve the same accuracy, but will take timesteps that are much smaller than the average mesh spacing. In fact, the work for our method will be proportional to δ^{-3}. This shortcoming will be taken up in a broader context later. We have compared the methods for problems with shocks, contact discontinuities, and expansion waves following shocks. These comparisons lead us to believe that our method is superior to fixed mesh monotone methods when the discontinuities in the solutions of (C) are smeared across more than a fixed number of mesh intervals by the monotone schemes.

In these examples, our method is compared with a finite difference scheme with a fixed, uniform, spatial grid. This scheme's difference operator is

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + \frac{f^+(U_i^n) - f^+(U_{i-1}^n)}{h} + \frac{f^-(U_{i+1}^n) - f^-(U_i^n)}{h} = 0.$$

Always $\Delta t = h$ to avoid any divisions or multiplications in this part of the algorithm. A careful implementation of the comparison scheme uses only two func-

tion evaluations, three subtractions, and one addition per meshpoint per timestep.

For all examples but the sixth, $f^{-}=0$, so that each difference operator reduces to the standard upwind-difference scheme. The initial values U_i^0 were chosen to interpolate the initial data u_0 at the points ih. The solution of the fixed mesh method was interpreted as a piecewise linear function taking on the values U_i^n at the points $(ih, n\Delta t)$. The difference in $L^1[a,b]$ between the approximate solution and the piecewise linear interpolant of the true solution $u(x,t)$ was recorded as the error of the both schemes.

Table 1 presents the fluxes f and the initial and final values of u for the computational experiments. Table 2 contains the errors and execution times, respectively, corresponding to various values of Δt. For the purposes of comparing the methods, the parameter δ, a measure of the average mesh spacing where the solution is smooth, is introduced. The parameter δ has the value $\sqrt{\Delta t}$ for the adaptive mesh method, and Δt for the fixed mesh method. Table 3 tabulates an expression of the error as $error = C\,time^{\alpha}$ and $error = C\delta^{\alpha}$ for each test.

TABLE 3
ERROR DECAY RATES

Test	Adaptive Mesh		Fixed Mesh	
1	$e = .027\,t^{-.826}$	$e = .508\,\delta^{1.995}$	$e = .033\,t^{-.493}$	$e = .472\,\delta^{.975}$
2	$e = .121\,t^{-.326}$	$e = .419\,\delta^{.915}$	$e = .057\,t^{-.487}$	$e = .770\,\delta^{.961}$
3	$e = .377\,t^{-.418}$	$e = 1.864\,\delta^{1.200}$	$e = .506\,t^{-.499}$ *	$e = .246\,\delta^{.995}$ *
4	$e = .179\,t^{-.319}$	$e = .684\,\delta^{.915}$	$e = .183\,t^{-.253}$	$e = .643\,\delta^{.457}$
5	$e = .082\,t^{-.373}$	$e = .338\,\delta^{.909}$	$e = .081\,t^{-.371}$	$e = .593\,\delta^{.736}$
6	$e = .091\,t^{-.381}$	$e = .397\,\delta^{1.034}$	$e = .087\,t^{-.376}$	$e = .652\,\delta^{.749}$

* See text.

The CPU times in Table 2 are the "user" times reported by the Unix operating system. The "system" time, which measures the time spent by the operating system to service the program, was not included. The system time varied greatly, depending on the system usage, so it was not deemed a reliable measure of the methods' resource needs. The user times corresponded well with the

TABLE 1
FLUX, INITIAL AND FINAL VALUES OF $u(x,t)$.

	$f(u)$	$u_0(x)$	$u(x,4)$
1	$1/4(u+u^2)$	$\begin{cases}1 \text{ if } x \le 1\\ 0 \text{ if } x > 1\end{cases}$	$\begin{cases}1 \text{ if } x \le 3\\ 0 \text{ if } x > 3\end{cases}$
2	$1/4(u+u^2)$	$\begin{cases}x-1 & \text{if } 1 \le x \le 2\\ 3-x & \text{if } 2 \le x \le 3\\ 0 & \text{otherwise}\end{cases}$	$\begin{cases}(x-2)/3 & \text{if } 2 \le x < 2+\sqrt{6}\\ 0 & \text{otherwise}\end{cases}$
3	$1/2u$	$u(x+2,4)$	$\begin{cases}2-4\lvert x-3\rvert & \text{if } \lvert x-3\rvert \le 1/2\\ 0 & \text{otherwise}\end{cases}$
4	$1/2u$	$\begin{cases}1 \text{ if } x \le 1\\ 0 \text{ if } x > 1\end{cases}$	$\begin{cases}1 \text{ if } x \le 3\\ 0 \text{ if } x > 3\end{cases}$
5	$\begin{cases}1-(1-u)^2 & \text{if } u \ge 1/2\\ u^2 & \text{if } u \le 1/2\end{cases}$	$\begin{cases}1 & \text{if } x \le 1\\ 0 & \text{otherwise}\end{cases}$	$\begin{cases}1 & \text{if } x \le 1\\ 1-(x-1)/8 & \text{if } 1 \le x \le 9-4\sqrt{2}\\ 0 & \text{otherwise}\end{cases}$
6	$\begin{cases}1-(1-u)^2 & \text{if } u \ge 1/2\\ -2u^3+3u^2-u/2 & \text{if } u \le 1/2\end{cases}$	$\begin{cases}1 & \text{if } x \le 1\\ 0 & \text{otherwise}\end{cases}$	$\begin{cases}1 & \text{if } x \le 1\\ 1-(x-1)/8 & \text{if } 1 \le x \le 9-4\sqrt{2}\\ 0 & \text{otherwise}\end{cases}$

CPU times measured on the VMS operating system on a lightly loaded machine. The error decay rates in Table 3 are based on a log-log least squares fitting of the data. They were calculated from the data in Table 2, but with the full accuracy of the data, which is truncated in Table 2.

Test 1 presents our method in the best light. This is a Riemann problem with a strong nonlinearity keeping the shock width to within a few of the small ($O(\Delta t)$) mesh intervals. At the same time, the solution is constant outside the shock region, so only $O(-\log(\Delta t))$ meshpoints are necessary to accurately represent the solution. The result is an error of $O(\Delta t)$ with a computational complexity of $O(-\Delta t \log(\Delta t))$.

Test 2 is perhaps the worst comparison between the fixed mesh and adaptive mesh methods. The adaptive method gives no more than $O(\varepsilon^{1/2})$ accuracy in the regions where the solution is smooth. To do so, it uses $O(\varepsilon^{-1/2})$ meshpoints.

TABLE 2
COMPUTATIONAL RESULTS

δ^{-1}	Test 1.				Test 2.			
	Adaptive mesh		Fixed mesh		Adaptive mesh		Fixed mesh	
2	1.252-1	0.12	2.258-1	0.02	2.468-1	0.13	3.419-1	0.02
4	3.237-2	0.88	1.252-1	0.07	1.174-1	1.18	2.079-1	0.07
8	8.093-3	5.25	6.456-2	0.27	5.449-2	8.88	1.119-1	0.28
16	2.023-3	26.33	3.237-2	1.07	3.169-2	58.85	6.276-2	1.07
32	5.058-4	132.65	1.618-2	4.37	1.771-2	383.15	2.852-2	4.48
64	1.264-4	625.68	8.093-3	17.65	9.547-3	2560.93	1.308-2	17.22
128	3.161-5	2988.28	4.046-3	71.25	5.129-3	17678.77	6.782-3	69.95
δ^{-1}	Test 3.				Test 4.			
	Adaptive mesh		Fixed mesh		Adaptive mesh		Fixed mesh	
2	8.727-1	0.12	1.116-0	0.02	3.611-1	0.10	4.560-1	0.02
4	4.010-1	1.22	8.627-1	0.05	1.904-1	0.88	3.425-1	0.07
8	1.374-1	10.62	6.139-1	0.25	1.022-1	6.45	2.530-1	0.23
16	5.532-2	73.72	3.954-1	1.00	5.429-2	42.57	1.846-1	1.00
32	2.747-2	493.65	2.302-1	4.02	2.942-2	299.60	1.335-1	3.98
64	1.348-2	3168.97	1.229-1	16.07	1.532-2	2235.78	9.592-2	16.00
128	6.091-3	20316.55	6.243-2	64.45	7.898-3	17124.27	6.860-2	64.40
δ^{-1}	Test 5.				Test 6.			
	Adaptive mesh		Fixed mesh		Adaptive mesh		Fixed mesh	
2	1.800-1	0.13	3.924-1	0.02	2.018-1	0.12	4.234-1	0.02
4	8.160-2	0.93	1.795-1	0.07	9.276-2	0.92	1.969-1	0.07
8	4.133-2	6.92	1.393-1	0.27	4.633-2	6.45	1.477-1	0.30
16	2.063-2	38.98	7.100-2	1.12	2.174-2	41.32	7.532-2	1.18
32	1.042-2	238.97	5.131-2	4.45	1.069-2	253.40	5.345-2	4.68
64	5.390-3	1472.12	2.875-2	17.63	5.440-3	1594.37	2.983-2	18.87
128	2.718-3	9701.87	1.590-2	71.07	2.716-3	10267.05	1.644-2	75.70

Coupled with a time step of $O(\varepsilon)$, we achieve a complexity bound of $O(\varepsilon^{-3/2})$, a decidedly inferior result when compared with the fixed mesh. This predicament strongly suggests a method where the local timesteps are proportional to the local meshsize. Such methods have been used by Oliger [18], Bolstad [2], Berger [1], and others (see references in [13]). With such a scheme, the error will still be $O(\varepsilon^{1/2})$, but the complexity of the scheme will be reduced to $O(\varepsilon)$. This is the same relationship between error size and complexity that applies for the fixed mesh method, which does surprisingly well for this problem.

When applied to the third test problem, the fixed mesh method behaves in a different way when the meshsize is small than when it is large. This is because of the large second derivatives in the solution. The fixed mesh method exhibits an error of order Δt^{α}, with α decidedly less than one, when the meshsize is large

(greater than 1/32). When the mesh size is small, however, the mesh can adequately refine the large gradients and the error is of order Δt. Because we are mainly interested in exhibiting "asymptotic" error rates, we have computed the error for the fixed mesh method for Δt as small as 1/1024, and our error decay rates in Table 3 are derived from the smaller timesteps.

Test 4 was the problem that motivated the design of the adaptive method. It corresponds to a contact discontinuity in gas dynamics. The discontinuity is smeared over $O(\Delta t^{-1/2})$ intervals, and the fixed mesh method has an accuracy of order $\Delta t^{1/2}$. The time complexity of the fixed mesh method is $O(\Delta t^{-2})$, so the error of the method is of order $t^{-1/4}$. The adaptive method puts $O(\Delta t^{-1/2})$ intervals of size Δt near the discontinuity, and the same number of size $\Delta t^{1/2}$ away from the it. Thus, the complexity of this scheme is of order $\Delta t^{-3/2}$, and the error is of order $t^{-1/3}$. In other words, if one wants an error of order ε, it takes order ε^{-4} work for the fixed method, and order ε^{-3} work for the adaptive method.

The solutions of Tests 5 and 6 exhibit behavior similar to the solutions of the Buckley-Leverett equation used as a model in petroleum reservoir simulation (see Douglas and Wheeler [8]). These solutions consist of a shock followed by a smooth expansion wave. The shock is not as strong as in the first example. The fixed mesh solution reflects this, as the shock is smeared over several mesh intervals, and an accuracy of order $\Delta t^{3/4}$, instead of Δt, is observed. The tests indicate that the relationship between complexity and error is the same for the adaptive method and the fixed mesh method. Changing the adaptive method to use locally varying timesteps as well as mesh spacing would decrease the CPU time greatly for these problems.

The results of Tests 1 through 6 suggest that our new method is effective when discontinuities in the solutions are smeared across many mesh intervals

by the fixed mesh method. This happens in particular for problems that have discontinuous solutions but weak nonlinear fluxes.

An alternate approach to developing an adaptive mesh method is presented in [8]. There, the motivation is to use an implicit scheme with a large timestep and to refine the mesh near roughness in the solution. Unfortunately, this strategy does not asymptotically decrease the error. Heuristically, this is because one cannot drag a shock or discontinuity across many meshpoints in one timestep without smearing the discontinuity. This effect is illustrated in Table 4 for two problems. Each problem has the solution $u(x) = 1/2\,(1-\mathrm{sgn}(x-t/2))$. The fluxes are $f(u) = 1/2\,u$ and $f(u) = 1/4\,(u+u^2)$, respectively. Each problem was run with h, the mesh spacing, equal to Δt and Δt^2. Thus, our choice is not to adapt the mesh, but to use a uniformly fine mesh. By using a fine mesh everywhere, our results do not depend on any particular mesh refinement algorithm. For the first problem the error is of order $\Delta t^{1/2}$ for both choices of the mesh. The error for the second method is of order Δt. Thus, it appears that spatial mesh refinement, without a corresponding refinement of the temporal increments, is not effective in solving these problems.

TABLE 4
ERRORS USING IMPLICIT METHOD

Δt	Test 1.		Test 2.	
	$h = \Delta t^2$	$h = \Delta t$	$h = \Delta t^2$	$h = \Delta t$
0.500000	7.8550e-01	9.5360e-01	5.1312e-01	6.7678e-01
0.250000	4.8564e-01	6.8264e-01	2.3904e-01	4.1525e-01
0.125000	3.1453e-01	4.8564e-01	1.0772e-01	2.3833e-01
0.062500	2.1129e-01	3.4444e-01	4.9128e-02	1.3741e-01
0.031250	1.4529e-01	2.4393e-01	2.3225e-02	6.4799e-02
0.015625	1.0124e-01	1.7261e-01	1.1274e-02	3.2443e-02

In the previous section, we showed that the arithmetic complexity of the adaptive mesh algorithm was no more than twice that of the fixed point algorithm per meshpoint per timestep. The algorithms differ greatly in their implementations, however. The fixed mesh algorithm is almost trivial to implement,

while the adaptive method uses sophisticated data structures and pointer manipulation. We therefore set out to quantify the amount of nonarithmetic overhead in each method.

We proposed to measure the proportion of the CPU time spent in arithmetic computations as compared with nonarithmetic computations for both implementations. We had available two VAX's running identical software, one of which did not have a floating point accelerator (FPA). A simple program consisting mainly of an equal number of nontrivial memory to register floating point additions and multiplications was run and timed on both machines. A speedup of a factor of five was observed on the machine with the FPA. (The effects of cache memory hits was not considered.) We then compared the CPU times for the same implementations of the two algorithms on both machines. The ratios of the CPU times are given in Table 5. Assuming that only floating point operations were speeded up by the FPA (a faulty assumption, for some integer arithmetic is also faster), we calculated the fraction of time spent in nonarithmetic operations by both programs on both machines. For Test 6, with nontrivial f^+, f^-, and $\bar{f}$, around 90% of the time was spent on nonarithmetic operations on the machine with the FPA for ***both*** problems. The figure is similar for Test 4, with trivial f^+, f^-, and $\bar{f}$.

To discover the effects of the Pascal compiler, we implemented the fixed mesh algorithm in FORTRAN and ran this program on the machine with the FPA. For some reason, the FORTRAN compiler generated much superior code for index calculations. Function calls were slightly cheaper because the FORTRAN program, not being block structured, did not maintain a "display" of the currently accessible data areas. The computation time of the FORTRAN program was 60% of its Pascal counterpart. This still leaves us with an overhead figure of about 67%. These tests indicate that the overhead is similar, and large, for each

algorithm.

TABLE 5
NONARITHMETIC OVERHEAD

Test	Adaptive mesh			Fixed mesh		
	CPU time ratio	Overhead FPA	Overhead No FPA	CPU time ratio	Overhead FPA	Overhead No FPA
4	0.728	91%	66%	0.678	88%	60%
6	0.699	89%	63%	0.715	90%	64%

8. REFERENCES.

1. M. Berger, "Adaptive mesh refinement for hyperbolic partial differential equations," Stanford Computer Science Report STAN-CS-82-924 (dissertation).

2. J. H. Bolstad, "An adaptive finite difference method for hyperbolic systems in one space dimension," Lawrence Berkeley Lab. LBL-13287 (STAN-CS-82-899) (dissertation).

3. C. de Boor, "Good approximation by splines with variable knots," in *Spline functions and approximation theory*, A. Meir and A. Sharma ed., ISNM v. 21, Birkhauser Verlag, 1973, pp. 57-72.

4. C. de Boor, "Good approximation by splines with variable knots II," in Lecture Notes in Mathematics 363, Springer Verlag, 1974, pp. 12-20.

5. M. G. Crandall & A. Majda, "Monotone difference approximations for scalar conservation laws," *Math. Comp.* v. 34, 1980, pp. 1-21.

6. S. F. Davis & J. E. Flaherty, "An adaptive finite element method for initial-value problems for partial differential equations," *SIAM J. Sci. Stat. Comput.*, v. 3, 1982, pp. 6-28.

7. J. Douglas Jr., "Simulation of a linear waterflood," in *Free Boundary Problems*, Proceedings of a seminar held in Pavia, Sept.-Oct. 1979, Vol. II, Insti-

tuto Nazionale di Alta Matematica "Francesco Severi," Roma, 1980.

8. J. Douglas Jr. & M. F. Wheeler, "Implicit, time-dependent variable grid finite difference methods for the approximation of a linear waterflood," *Math. Comp.*, v. 40, 1983, pp. 107-122.

9. Todd Dupont, "Mesh modification for evolution equations," *Math. Comp.*, v. 39, 1982, pp. 85-107.

10. B. Enquist & S. Osher, "Stable and entropy satisfying approximations for transonic flow calculations," *Math. Comp.*, v. 34, 1980, pp. 45-75.

11. D. B. Gannon, "Self adaptive methods for parabolic partial differential equations," U. of Illinois CS Dept. report UIUCDCS-R-80-1020.

12. A. Harten, J. M. Hyman & P. D. Lax, "On finite difference approximations and entropy conditions for shocks," *Comm. Pure Appl. Math.*, v. 29, 1976, pp. 297-322.

13. G. W. Hedstrom & G. H. Rodrigue, "Adaptive-grid methods for time-dependent partial differential equations," in *Multigrid Methods*, W. Hackbusch and U. Trottenberg, ed., Springer Verlag, 1982, pp. 474-484.

14. S. N. Kruzkov, "First order quasilinear equations with several space variables," *Math. USSR Sb.*, v. 10, 1970, pp. 217-243.

15. N. N. Kuznetsov, "On stable methods for solving non-linear first order partial differential equations in the class of discontinuous functions," in *Topics in Numerical Analysis*, John J. H. Miller, ed., Academic Press, New York, 1977, pp. 183-197.

16. B. J. Lucier, "On nonlocal monotone difference methods for scalar conservation laws," to appear.

17. K. Miller, "Moving finite elements, parts II," *SIAM J. Numer. Anal.*, v. 18, 1981, pp. 1019-1057.

18. J. Oliger, "Approximate Methods for Atmospheric and Oceanographic Circulation Problems," in *Lecture Notes in Physics 91*, R. Glowinski and J. Lions, ed., Springer Verlag, 1979, pp. 171-184.

19. R. Sanders, "On convergence of monotone finite difference schemes with variable spatial differencing," *Math. Comp.*, v. 40, 1983, pp. 91-106.

BJL/jvs

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER # 2517	2. GOVT ACCESSION NO. AD-A130 512	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Stable Adaptive Numerical Scheme for Hyperbolic Conservation Laws		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Bradley J. Lucier		8. CONTRACT OR GRANT NUMBER(s) MCS-7927062, Mod. 1 DAAG29-80-C-0041
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work Unit Number 3 - Numerical Analysis and Scientific Computing
11. CONTROLLING OFFICE NAME AND ADDRESS See Item 18 below		12. REPORT DATE May 1983 13. NUMBER OF PAGES 40
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

National Science Foundation
Washington, DC 20550

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Conservation laws, finite-difference schemes, adaptive numerical methods.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A new adaptive finite-difference scheme for scalar byperbolic conservation laws is introduced. A key aspect of the method is a new automatic mesh selection algorithm for problems with shocks. We show that the scheme is L^1-stable in the sense of Kuznetsov, and that it generates convergent approximations for linear problems. Numerical evidence is presented that indicates that if an error of size ε is required, our scheme takes at most $O(\varepsilon^{-3})$ operations. Standard monotone difference schemes can take up to $O(\varepsilon^{-4})$ calculations for the same problems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

N D

TE

MED

83

TIC