JOURNAL OF COMPUTATIONAL PHYSICS 53, 299–318 (1984)

The Use of Nonrational Functions to Represent Steep Front Solutions to Partial Differential Equations

JOHN GREENSTADT

IBM Corporation, Palo Alto Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304

Received January 26, 1983; revised July 6, 1983

It is shown, by calculated examples, that it is possible to follow successfully the development and evolution of steep fronts (almost to but not including actual discontinuities) in the solution of time-dependent partial differential equations by means of global non-rational representations. A characteristic steplike shape forms the basis for the representations; the hyperbolic tangent and a function involving a square root, both having this characteristic shape, gave similar results. The Method of Lines, specifically as formulated by Gelinas, Doss and Miller (*J. Comput. Phys.* **40** (1981), 202–249) was used to form a system of ordinary differential equations governing the time-variation of the parameters in the representation. A fixed-shape moving solution of the wave equation, and developing steep fronts in solutions of Burgers' equation and of the Buckley–Leverett equation were successfully computed. The results are all displayed in the form of plotted profiles, and are compared with the corresponding results in the literature.

1. INTRODUCTION

In the course of solving time-dependent nonlinear partial differential equations, it frequently happens that steep gradients occur in the solution along particular submanifolds in the configuration space (points in one-dimensional problems, curves in 2D, surfaces in 3D, etc.). These submanifolds are generally called "fronts." In many problems, the gradients may become so large that the solution is often treated as if it really is discontinuous across the submanifold.

If the solution is not really discontinuous, we may simply refer to it as a "steep front," partly to distinguish it from a genuine shock, across which the solution has to satisfy special interface conditions. Clearly, if the solution remains continuous, there will be no need for special conditions, since we simply require the solution to satisfy the partial differential equation (PDE) at all points in the domain of interest.

When a PDE is discretized for the purpose of solving it on a digital computer, the problem arises of approximating (or "representing") the solution at the front within the framework of the discretization. There are several ways of doing this:

(a) When a simple finite-difference discretization is used, the solution is assumed to make the transition across the front in a few grid points. When the grid

0021-9991/84 $3.00

spacing is not fine enough to handle the steepness which may be present, the resulting "smoothing" (and often spurious oscillation) of the finite-difference solution is sometimes called "grid dispersion."

(b) To correct this error, the entire grid may be made fine enough to represent whatever steepness is deemed sufficient. This, of course, increases the number of grid points greatly, and increases the calculation time accordingly.

(c) To avoid having too many grid points, the finite-difference grid may be refined locally, so that the fine mesh covers only the vicinity of the front. This procedure may involve considerable logistical problems in the computer program, as well as theoretical problems associated with the interpolation from the coarse to the locally fine grid.

(d) If the front is really a discontinuity, it is necessary to solve a set of *Riemann problems* at each time step. This method eliminates the dispersion problem completely, even with a relatively coarse mesh, but it is somewhat complicated (since every mesh interval at every time slice is a condidate for a shock), and when implemented as the so-called "Random Choice" method, yields solutions with substantial superimposed noise [1].

(e) When the finite element method is used, the grid refinement problem persists, and may be handled in the same ways as for the finite-difference method. The single exception is the so-called moving finite element (MFE) method [2], in which the grid points (or nodes) are caused to move by the direct action of the differential equation itself. The MFE method is a special case of the method of lines, but it has the virtue that the grid really does refine itself automatically wherever there is rapid variation in the solution. This method, which is currently being studied for 2D problems, appears to be one of the most flexible and general for front-tracking.

The reason that the finite-difference and finite element methods require considerable refinement of the grid whenever there is a steep front, is that they both use polynomial representations of the solution, and it is well known that polynomials are ill-suited to represent really rapid variations in function values. For this reason, we have considered the use of alternative classes of functions, since some of them are capable of making the transition from slow to very rapid variation, with only modest changes in the parameters which characterize the solution.

There is ample historical precedent for this approach. Even for one of the simplest ordinary differential equations, the linear one with constant coefficients. it is easy to choose the characteristic frequencies (i.e., the eigenvalues) so that a polynomial representation of the solution (on which almost all the standard numerical solution methods are based), has very great difficulty in following the rapid variations. These are the well-known "stiff" equations. On the other hand, by using the familiar *Ansatz* of representing the solution as a sum of *exponentials* with the parameters appearing *nonlinearly* in the exponents, the solution of the problem is reduced to a more elementary one, that of solving an algebraic equation for its roots.

Another example, from classical mechanics, is the use of a nonlinear representation

of the orbit of a planet, e.g., the well-known polar formula for an unperturbed orbit. The parameters of this representation are then allowed to vary (slowly) with time, and they can then account for the precession of the orbit as well as for secular changes in its eccentricity.

We have adopted this approach in an attempt to represent steep fronts without the need for excessive mesh refinement. What we shall describe is more a "proof of principle" than a practical, working method. As we shall see, we become embroiled in time-consuming adaptive quadrature (which breaks down completely when the front steepness becomes extremely large), the calculation of "pseudopseudoinverses," and other complications. However, we shall indicate at the conclusion how the general idea might be adapted to a more practical computation scheme for solving steep-front problems.

2. General Form of the Problem

We shall model our treatment of the steep-front problem along the lines described in the paper of Gelinas, Doss, and Miller [2]. Their provocative idea of the use of a nonlinear parametrization of the approximate solution served as the inspiration for the method presented here.

The equation to be solved has the form

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \varepsilon \frac{\partial^2 u}{\partial x^2}. \tag{2.1}$$

In all of our examples, we shall have x ranging from 0 to 2 and t ranging from 0 on. The amplitude of the propagating disturbance which may form a steep front is u, and $f(u)$ is a function which defines the nonlinear character of the equation. The second derivative term on the right-hand side represents a possible dissipative effect which we shall sometimes include, although in most of our examples, $\varepsilon = 0$. If $f(u) = u$, we have the first-order wave equation, which waves traveling to the right; if $f(u) = \frac{1}{2}u^2$, we have a version of the Burgers equation., and if $f(u)$ is given by the function

$$f(u) = \frac{u^2}{u^2 + \alpha(1-u)^2} \tag{2.2}$$

then Eq. (2.1) is a simplified version of the Buckley–Leverett equation describing the flow of two miscible fluids through a porous medium. We were primarily interested in solving this last equation using our method.

We can follow the formal development of [2] quite far before introducing special representations. The first step, which is a form of the well-known method of lines, is to represent the dependent variable u as a predetermined function of the configuration variable x, and a set of parameters $\{a_i(t);\ i = 1,..., M\}$ which carry all of the time-dependence of $u(x, t)$. Thus

$$u(x, t) = \psi[x, a(t)]. \tag{2.3}$$

(In [2], the function ψ is a one-dimensional finite element representation, using linear splines and *movable nodes*.) The time derivative $\partial u/\partial t$ can then be expressed solely in terms of the time derivatives of the a's as

$$\frac{\partial u}{\partial t} = \sum_i \frac{\partial u}{\partial a_i} \dot{a}_i. \tag{2.4}$$

For convenience, we shall define the operator L (in x) as

$$Lu \equiv -\frac{\partial f}{\partial u}\frac{\partial u}{\partial x} + \varepsilon \frac{\partial^2 u}{\partial x^2} \tag{2.5}$$

so that (2.1) takes the form

$$\frac{\partial u}{\partial t} = Lu. \tag{2.6}$$

Now, since there are only M a's, the representation (2.3) cannot in general be the exact solution to Eq. (2.6). Therefore, a system of equations must be constructed, M in number, to enable the appropriate values of $\{a_i\}$ to be found.

The simplest method for doing this is to multiply (2.6) by each of the $\{\partial u/\partial a_i\}$ in turn, and then integrate with respect to x over the interval $[0, 2]$. The result is the system

$$\sum_j A_{ij}\dot{a}_j = b_i, \tag{2.7}$$

where

$$A_{ij} \equiv \left\langle \frac{\partial u}{\partial a_i}, \frac{\partial u}{\partial a_j} \right\rangle \tag{2.8a}$$

and

$$b_i \equiv \left\langle \frac{\partial u}{\partial a_i}, Lu \right\rangle, \tag{2.8b}$$

and the "inner product" bracket is defined by

$$\langle A, B \rangle \equiv \int_0^2 A(x)\, B(x)\, dx. \tag{2.9}$$

The same set of Eqs. (2.7) can be derived from a variational principal of sorts. When the approximation for u is used, there will, in general, be a residual, defined by

$$R \equiv \frac{\partial u}{\partial t} - Lu \tag{2.10}$$

and, if we define the squared norm of R in terms of the inner product, namely,

$$\|R\|^2 \equiv \langle R, R \rangle, \tag{2.11}$$

then Eq. (2.7) can be derived by requiring the norm of R to be a minimum with respect to variations in $\{\dot{a}_i\}$.

If Eqs. (2.7) are solved, we should expect the a's to evolve through time in such a way as to maintain the representation $\psi[x, a(t)]$ as a reasonable approximation to the true solution $u(x, t)$. In [2], it is shown by various examples that the MFE method is quite capable of providing good solutions of (2.1) for the three forms of $f(u)$. In the next section, we shall describe the class of representations with which we attempt to accomplish the same result.

3. Steplike Functional Representations

If we accept the inadequacy of polynomial representations for steep-fronts, then we must ask what other functional forms are available. Rational approximations have been studied much, but they too, seem to suffer from the same inability to make a transition from a mildly varying function to a rapidly varying one with only modest changes in the key parameter. On a finite segment, rational approximations can doubtless be constructed which have this desirable property, but they too, have a tendency to vary vigorously where it is not desired that they should.

We shall try to find some guidance in the selection of representations by the heuristic approach of asking what kinds of functions (of a scale-free variable v) can represent a "step" on the *infinite* line $[-\infty, +\infty]$. That is, if $v \to -\infty$, the "steplike" function $S(v)$ should approach a limit A, and if $v \to +\infty$, the function S should approach a *different* limit B. Consider v to range over the complex plane, in which the real line is imbedded, and consider the class of analytic functions whose only singularities in the finite plane are at worst *poles* (i.e., the *meromorphic* functions). In this case, the point at infinity cannot be a regular point, or A and B would be the same; nor can it be a pole, or A and B would be infinite, a condition which would render ψ useless as a step representation. Therefore, if S is to be meromorphic, the point at ∞ must be an essential singularity, since almost any value can be the limit as $v \to \infty$ in various directions. Conversely, if we insist that ∞ be a regular point, then S cannot be a meromorphic function, but must have either essential singularities or branch points in the finite plane [3]. In case S is required to be bounded in the finite plane, we are led to consider either nonmeromorphic functions with ∞ as a regular point, or entire functions with an essential singularity at ∞, so that we can then have A and B finite but different. This rules out the class of rational functions (naturally including the polynomials).

There are many possible candidates for "step" representations $S(v)$, but the two simplest ones we have condidered are the hyperbolic tangent

$$S(v) = \tanh v \tag{3.1}$$

and the irrational algebraic function

$$S(v) = \frac{v}{(1+v^2)^{1/2}}. \tag{3.2}$$

Both of these functions go to -1 at $-\infty$, and $+1$ at $+\infty$. The hyperbolic tangent does so because it is an entire function with an essential singularity at ∞, and goes to different limits when that point is "approached" in different directions, and the square root function because it has branch points at $+i$ and $-i$. If a branch cut is made, starting from $+i$ and going up the imaginary axis, through the point at ∞, and back up to $-i$ from below, then any path that starts out to the right on the real axis, goes through the point at ∞ and then returns from the left, will have passed the cut, and will be on the other sheet of the Riemann surface defined by the cut, thus returning with the other value.

In order to allow for varying steepness and position of our step, we relate the argument v to the original variable x of the problem by

$$v = a_1(x - a_2), \tag{3.2a}$$

where a_2 is obviously the location of the front, and we shall refer to a_1 as the *steepness parameter*. In the particular applications we have considered, it has been more convenient to use a step-function whose values go from 0 to 1, instead of from -1 to 1. We use the new step-function $S^*(v)$, defined by

$$S^*(v) \equiv \tfrac{1}{2}[S(v) + 1]. \tag{3.2b}$$

We can henceforth drop the asterisk, because we shall use the $0 \to 1$ step-function from now on.

With this choice of $S(v)$, we note that $1 - S(v)$ is a step that goes from 1 to 0, as is the function $S(-v)$. If we now imagine that the steepness parameter a_1 is a very large negative number, then the function $S[a_1(x - a_2)]$ is essentially unity from 0 to a_2, and nearly vanishes from a_2 to 2, while its "complementary" function $1 - S$ does just the opposite. Hence, the step-function divides the interval $[0, 2]$ into two "regimes" in the following sense: If we have two "shape" functions, $Q_1(x)$ and $Q_2^*(x)$ defined over $[0, 2]$, then the function

$$\psi(x) \equiv Q_1(x) \cdot S[a_1(x - a_2)] + Q_2^*(x) \cdot (1 - S[a_1(x - a_2)]) \tag{3.3}$$

has essentially the shape of Q_1 from 0 to a_2 and that of Q_2^* from a_2 to 2. We can simplify this form by combining the coefficients of S, and *redefining* $Q_1 - Q_2^*$ to be just Q_1, so that we end up with

$$\psi(x) \equiv Q_1(x) \cdot S[a_1(x - a_2)] + Q_2^*(x). \tag{3.4}$$

In the problems we shall consider, these shapes are fairly smooth and simple, so it suffices to choose low-order polynomials for the Q's, and we shall allocate more of the a's to these functions. For convenience, we shall use the notation a_1^* to denote the set of parameters $\{a_{1,1}, a_{1,2},..., a_{1,M_1}\}$, and a_2^* to denote $\{a_{2,1}, a_{2,2},..., a_{2,M_2}\}$. Thus, we are dealing with a total of $2 + M_1 + M_2$ parameters in the representation of the function ψ. These parameters, $\{a_1, a_2, a_{1,1}, a_{1,2},..., a_{2,1}, a_{2,2},..., a_{2,M_2}\}$, we shall denote simply by a.

At this point, it is useful to incorporate the boundary conditions on ψ into its representation, so that they wil be automatically satisfied. In this way, the total $(2 + M_1 + M_2)$ parameters $\{a\}$ will not be constrained in any way. This is usually not the best way of handling boundary constraints; rather, a more symmetrical a posteriori imposition of these constraints makes for a more flexible algorithm. Unfortunately, there was no opportunity to investigate such an improved boundary constraint procedure, so the method of explicit construction was adhered to for this study. As can be seen from Eq. (3.7), the correction to ψ is very complicated, and obviously depends on the type of b.c. (i.e., Dirichlet or Neumann). For this reason alone, we have limited the scope of this work, and have considered only the following b.c.'s, which were used for the Buckley–Leverett problem in the literature

$$\psi(0, a) = 1, \tag{3.5a}$$

$$\frac{\partial \psi(2, a)}{\partial x} = 0. \tag{3.5b}$$

Accordingly, if we now define

$$Q_1(x, a_1^*) \equiv \sum_{i=1}^{M_1} a_{1,i} x^{i-1} \tag{3.6a}$$

and

$$Q_2^*(x, a_2^*) \equiv w_1 + w_2(x - 2) + Q_2(x, a_2^*), \tag{3.6b}$$

where

$$Q_2(x, a_2^*) \equiv \sum_{i=1}^{M_2} a_{2,i}(x - 2)^{i+1} \tag{3.6c}$$

we can choose w_1 and w_2 so that ψ will automatically satisfy the boundary conditions. w_1 and w_2 are, as indicated above, rather complicated functions of the a's. If we define

$$Q_1' \equiv \frac{\partial Q_1}{\partial x}; \qquad Q_2' \equiv \frac{\partial Q_2}{\partial x}, \tag{3.7}$$

$$v_0 \equiv a_1(0 - a_2); \qquad v_2 \equiv a_1(2 - a_2),$$

$$S_0 \equiv S(v_0); \qquad S_2 \equiv S(v_2),$$

$$P(v) \equiv \frac{\partial S(v)}{\partial v}; \qquad P_0 \equiv P(v_0); \qquad P_2 \equiv P(v_2),$$

w_1 and w_2 have the values

$$w_2 = -(Q_1'(2) \cdot S_2 + Q_1(2) \cdot P_2), \tag{3.8a}$$

$$w_1 = 1 - (Q_1(0) \cdot S_0 + Q_2(0)) + 2w_1. \tag{3.8b}$$

We also need initial conditions in order to start the integration of the system (2.7). We have dealt with this problem by first selecting the appropriate function which has the shape desired for a particular case, and then doing a least-squares fit of the functional form (3.4) to find the best values of the a's for that shape. The minimum set of a's is used which can give a good fit to the true initial function. This approximation is then used as the initial condition for the integration. It would also be possible to generate a more flexible (but undetermined) representation by including more parameters than necessary for the initial fit, but this interesting line of investigation falls outside the scope of the work described here.

It is important to note that the system matrix which results from the application of the Gauss–Newton method of solving the least-squares problem is identical with that which arises in the method of lines, viz., the A-matrix defined by (2.8a). Hence, if we restrict the condition number of the ODE matrix, it is of little use to try to resolve the fit to better than what is feasible with an A-matrix of similar condition. Hence, the least-squares fit is done using the same "pseudoinverse" technique as is used in solving the ODE system.

4. Integration of the ODE System

The first computational problem we face in solving (2.7) is the evaluation of the inner products defined in (2.8) to sufficient accuracy. True, some of these quadratures can be evaluated analytically, but many of the most important ones cannot, either for the hyperbolic tangent or the square-root forms of $S(v)$. For the sake of simplicity in our APL program, we took the position of evaluating all of the inner products by numerical quadrature, using an adaptive Newton–Cotes (6th-order) procedure similar to that described in the book of Forsythe *et al.* [4, Chap. 5].

With this method, we first specify the maximum allowable estimated error relative to the maximum magnitude attained by the running value of the integral as the quadrature progresses. If the final relative error is too large, we redo the calculation with a more stringent stepsize criterion, and repeat this cycle until the accuracy requirement is met; otherwise, after a fixed number of tries, the quadrature program gives up, and declares the relative error to be zero. The reason for this is that extensive observation and testing have shown that these ultimate failures occurred only when the true value of the integral itself was less in magnitude than the rounding error—in other words, a "zero."

As the matrix A is being computed, each new estimated relative error is compared

in magnitude with the maximum of those that have previously been computed, and the larger of the two is retained. In this way, the maximum relative error in A (MREA) is determined, as a measure of the perturbation of the true A due to quadrature error.

Even with the highest relative accuracy which was specified (a relative error bounded below, in some cases, by 10^{-6}), the eigenvalues of the matrix A, in Eq. (2.7) had a magnitude variation greater than 10^{-6} by several orders of magnitude, indicating that the computed matrix A was effectively *singular*. Hence, it was impossible to solve the system (2.7) for the derivatives $\{\dot{a}_i\}$, and hence, the system could not be cast in the standard form required for the available ODE solvers.

It was therefore necessary to think in terms of using some sort of pseudoinverse of A, instead of trying to solve the singular system (2.7). After much experimentation, a way was found of "doctoring" the computed eigenvalues of A for use in the pseudoinverse A^+. The standard procedure of suppressing (in the pseudoinverse) those eigenvalues whose magnitudes are less than a fixed multiple (PIVTHR) of the largest, did not work in practice. It seemed that the "wrong" set of eigenvalues and eigenvectors was being suppressed. However, when the A-matrix was *prescaled* by suitably multiplying rows and columns so that the resulting "A^*" was still symmetric, but had unit values on its main diagonal, the very same eigenvalue suppression procedure previously used behaved quite differently, and the solutions of the resulting system began to look reasonable.

Initially, an APL version of the Gear–Hindmarsh code [5] (kindly supplied by Dr. Sandy Roberts) was used, and gave creditable reults, but it was ultimately determined that the system (2.7) *is not stiff*, and hence, that the rather elaborate Gear procedures are not necessary. Further, because of the noisy nature of the "right-hand side" (i.e., A^+b) of the ODE system, due to the randomly fluctuating eigenvalues, even the implicit Adams scheme which is included in the Gear–Hindmarsh code for non-stiff problems, gave rise to very considerable difficulty. In fact, the corrector phase of the predictor-corrector procedure often failed to converge, even when the a's were varying smoothly.

To improve efficiency, the Runge–Kutta–Fehlberg method [4, Chap. 6] was tried, with an adaptive step determination similar in nature to that used in the quadrature. This explicit method (described briefly in the Appendix) worked very nicely, with little of the backtracking experienced with the implicit Adams code. Even so, the "roughness" of the ODE system, because of rank changes in A^+ due to variations in the eigenvalues, caused unduly small steps to be selected.

To help overcome the effect of the noisiness of the estimated relative errors, a type of "soft" rank determination was devised. We shall describe this in terms of the ratios $\{\lambda_i/\lambda_1\}$, which we denote by $\{\rho_i\}$. (The corresponding procedures in terms of the λ's are more complicated to explain, but are equivalent.) Our original rank determination was based on setting the reciprocals of all those ρ's less than PIVTHR to zero. This is the same as giving all of these small ρ's the value ∞, and then taking reciprocals as if they were bounded. We can do something analogous to this by first constructing a function $\phi(\rho)$ with the following properties:

(1) $\phi(1) = 1$,

(2) $\partial\phi(1)/\partial\rho = 1$,

(3) $\phi \to \infty$ as $\rho \to 0$,

(4) ϕ has a minimum at ρ_m ($\equiv$PIVTHR).

A simple function with these properties has the form

$$\phi(\rho) = a + b\rho + c_\rho{}^{-k}, \tag{4.1}$$

where, if we define D to be $1 + \rho_m^{-(k+1)}$, a, b, and c are given by:

$$a = \left(1 - \frac{1}{k}\right)\Big/ D \quad , \tag{4.2a}$$

$$b = (\rho_m^{-(k+1)})/D, \tag{4.2b}$$

$$c = 1/(kD). \tag{4.2c}$$

After some experimentation, it was found that a value for k of about 6 gave the right cutoff properties for $\phi(\rho)$. The ρ-values were then evaluated via

$$\rho^* = \phi(\rho). \tag{4.3}$$

Of course, after the application of ϕ, it was only necessary to take the ordinary inverse.

By the use of the adaptive Runge–Kutta–Fehlberg scheme, and the ϕ-mapping of the eigenvalues described above, we were able to integrate (2.7) in a quite satisfactory way. In the next section, we shall show some of the profiles that were generated by solving (2.7) using these means.

5. Numerical Tests

Preliminary experiments indicated that the square-root form of the step-function was handled more easily by the quadrature routine, and the front-shapes were as satisfactory as those obtained with the tanh function. We therefore used the square-root function for almost all of the tests, except as noted explicitly. The specified relative accuracy for the quadratures was chosen to be that power of 10 which was just below the smallest ρ, except that it was constrained to lie between 0.001 and 10^{-7}. The upper bound on the MREA was necessary in order to obtain reasonably accurate and stable values of the A_{ij}'s, so that the solution to (2.7) would at least

resemble the correct one. The lower bound was set because of the rapidly increasing time and difficulty for the adaptive quadrature to satisfy the accuracy requirement for diminishing values of the latter. It was simply not practical to require the MREA to fall below 10^{-7}. The adaptive quadrature also had greater difficulty in meeting the accuracy requirement for very large values of the steepness parameter a_1. In fact, for magnitudes of the latter much above 100, it was not practical to demand that it should. Hence, we did not attempt to push the time beyond this point.

Our experience has shown that if the steepness magnitude gets beyond about 100, it is then necessary, in order to continue the solution, to regard the front as a true discontinuity, and use for $S(v)$ a real step-function. The adaptive quadrature would then have to take cognizance of the real division of the interval $[0, 2]$ into two subintervals in order to get accurate values. The pulse function would be a real δ-function etc., and the evaluation of $\{A_{ij}\}$ would have to be done somewhat differently for these special functions. We have not had the opportunity to attempt an investigation of this variant of the general approach.

Case 1 (*Traveling square wavefront*). The purpose of this test was to determine if the formalism described in Sections 3 and 4 would even allow a simple steep wavefront to propagate as a solution of the first-order wave equation. Therefore, the function $f(u)$ was set equal to u, and the initial condition was described by the function

$$u(x, 0) = a_{1,1} S\{a_1(x - a_2)\} + a_{2,1} \tag{5.1}$$

with $a_1 = -50$, $a_2 = 0.3$, $a_{1,1} = 0.8$, and $a_{2,1} = 0.2$. Because of the influence of the right-hand boundary condition and the inclusion of the correction involving w_1 and w_2, the least-squares approximation $\psi(x, 0; a)$ had somewhat different values of the a's. The actual starting values for a_1, a_2, $a_{1,1}$, $a_{2,1}$ were, respectively, 49.8497, 0.299961, 0.800689, 8.2×10^{-5}, which, except for $a_{2,1}$, are obviously close to the initially chosen values.

The integration of the system (2.7) was done, and the results for $\psi(x, t)$ are plotted in Fig. 1 (where the abscissa is x and the ordinate is ψ. The dotted curve is the initial

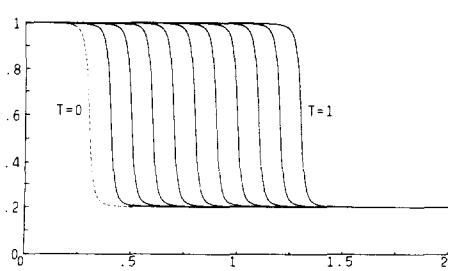

FIG. 1. Traveling steep wavefront. Solution of wave equation, $\varepsilon = 0$. Square-root step-function; $2 + 1 + 1 = 4$ parameters (a's) in representation.

condition.) As can be seen, the wavefront is very stable, and the final values for the a's, at $t = 1$, are -50.2182, 1.29998, 0.800186, 1.7×10^{-4}. This example thus demonstrates a remarkable fidelity of the representation to the actual wave solution, considering that the prescribed relative quadrature accuracy was mostly only .001.

Case 2 (*Gradually forming Shockfront*). For $f(u) = \frac{1}{2}u^2$, Eq. (2.1) becomes a form of Burgers' equation. The initial condition was set equal to the best fit to the function

$$u(x, 0) = 0.6 + 0.4 \cdot \cos(\tfrac{1}{2}\pi x). \tag{5.2}$$

A good fit was obtained using only 1 extra parameter $a_{1,1}$ in Q_1 and likewise in Q_2, for a total of 4 parameters in $\psi(x, 0, a)$. As can be seen in Fig. 2, a front formed in a reasonable way. (In the course of this investigation, these curves gave the first sign that this method was capable of following the formation and propagation of steep fronts.) With no dissipation ($\varepsilon = 0$), the program proved incapable of integrating the ODE system beyond the output point for $t = 1.6$. For that time value, the steepness was over 144 in magnitude, and the values of this parameter were rising into the thousands during the following integration steps. For such large values of a_1, the quadrature routine could not achieve the required relative accuracy.

Case 3 (*Miscible displacement front*). For $f(u)$ given by (2.2), we have the Buckley–Leverett equation. This was solved (for $\alpha = 0.5$) with the initial condition

$$u(x, 0) = \frac{0.1}{0.1 + x}. \tag{5.3}$$

It took 3 parameters in Q_1, and the same number in Q_2 to get a good fit of $\psi(x, 0, a)$ to $u(x, 0)$. The results of the integration of the ODE system are shown in Fig. 3. The same problem of excessive steepness was encountered in extending the calculation beyond $t = 0.4$.

To demonstrate the lack of serious stiffness, the 8 parameters in this representation are plotted against time, using cubic splines to interpolate between the data points, which occur at intervals of 0.05 in the time. In Fig. 4, the first 5 parameters, which

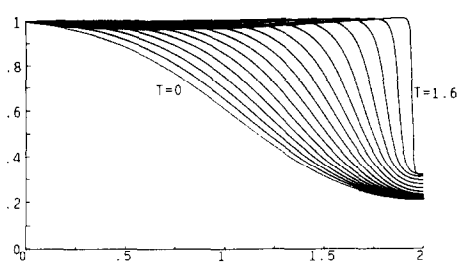

FIG. 2. Developing steep wavefront. Solution of Burgers' equation, $\varepsilon = 0$. Square-root step-function; $2 + 1 + 1 = 4$ parameters.

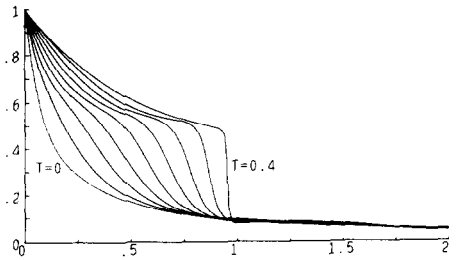

FIG. 3. Developing steep front. Solution of Buckley–Leverett equation, $E = 0$. Square-root step-function; $2 + 3 + 3 = 8$ parameters.

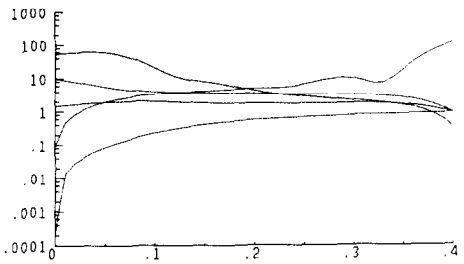

FIG. 4. Changes in parameters with time. Buckley–Leverett case, $\varepsilon = 0$. Square-root step-function; first 5 parameters on semilog plot.

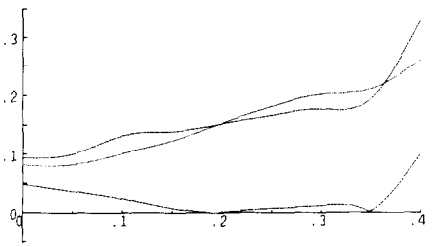

FIG. 5. Changes in parameters with time. Buckley–Leverett case, $\varepsilon = 0$. Square-root step-function; last 3 parameters on linear plot.

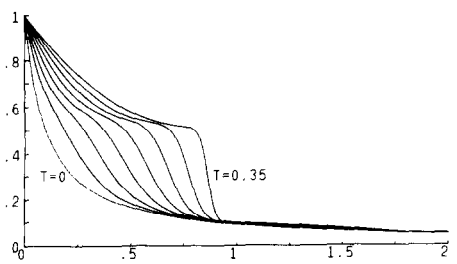

FIG. 6. Developing steep front. Solution of Buckley–Leverett equation, $\varepsilon = 0$. Tanh step-function; $2 + 3 + 3 = 8$ parameters.

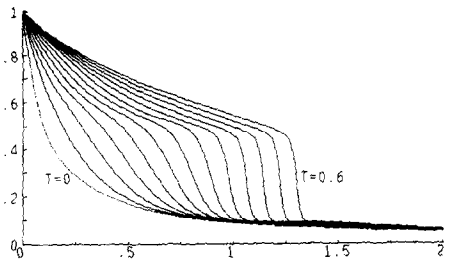

FIG. 7. Developing steep front. Solution of Buckley–Leverett equation, $\varepsilon = 0.01$. Square-root step-function; $2 + 3 + 3 = 8$ parameters.

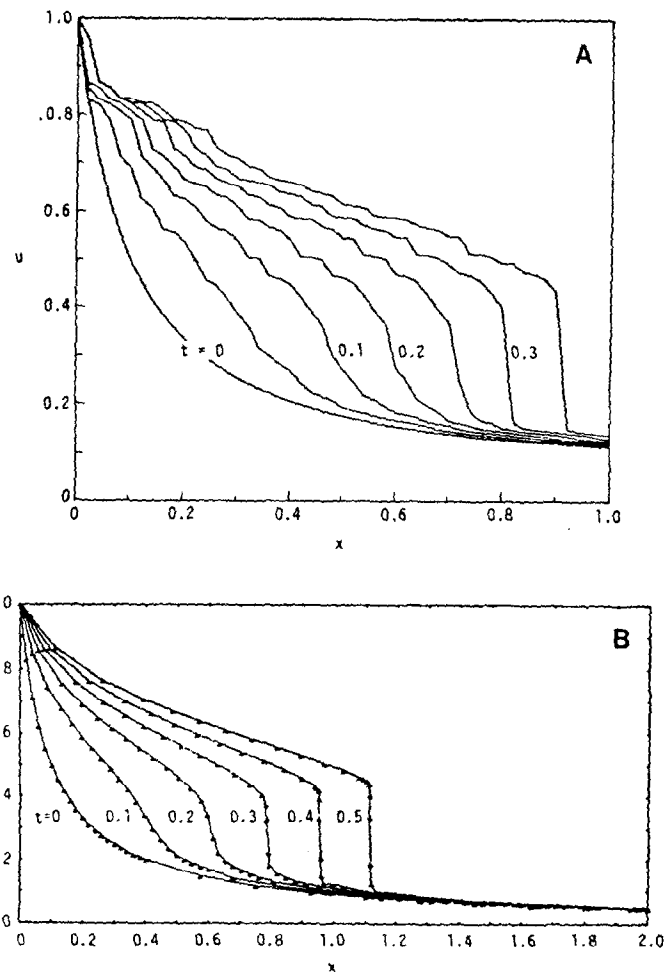

FIG. 8. Results of Buckley–Leverett solution, with $\varepsilon = 0$, from literature for comparison. (a) Concus and Proskurowsky [1], (b) Gelinas, Doss, and Miller [2].

are sizable and do not change sign, are shown plotted on a semilog scale. The last 3 parameters are much smaller, and the 8th one changes sign. These are shown on an ordinary linear plot in Fig. 5. It is fair to say that the variation of the parameters, even when the steep-front is forming, is not as drastic as one would expect for the solution of a stiff problem. The initial steepness of a two of the parameters on the semilog plot is due mainly to the fact that their starting values are close to zero. The jog in one of the curves (which happens to be for a_1) is associated with the transition to a jump discontinuity which occurs (somewhat earlier) in the true solution.

For purposes of comparison, the same problem has been done using the transcendental tanh step-function. The results are shown in Fig. 6. The profiles are almost identical with those of Fig. 3, although the tanh step-function seems less able to continue when the steepness exceeds about 50 in magnitude.

All of the preceding calculations were done with $\varepsilon = 0$. The fact that the representation contains only 8 degrees of freedom introduces a certain "rigidity" which tends to act somewhat like an artificial dissipation, in that it "softens the corners." However, it is still of interest to try the Buckley–Leverett example with a nonvanishing ε. Figure 7 shows the result of setting $\varepsilon = 0.01$. The effect of the dissipative part is to soften the front (i.e., to limit its steepness), to the extent that it can be followed much farther. We have stopped the calculation at $t = 0.6$.

Proskurowski, and by Gelinas, Doss, and Miller, so that our results can be compared with theirs.

6. Discussion

What we have shown by these relatively simple examples, is that steep-front problems can be solved without the *necessity* of using refined meshes or grids, although we are not claiming that the latter is not a *good* approach, but only that it is not the *only* approach. By using representations more general than rational functions alone, it appears to be feasible to maintain a coarse grid, and still describe a rapidly varying function. Nor are we recomending the method of lines, in conjunction with the functional (2.11), for the solution of time-dependent problems of this type; in point of fact, a functional more strongly based on physical principles would seem preferable. We are also not recommending or suggesting that real problems involving steep fronts should be solved using only a single domain and a single representation, as was done here. For real problems, it is almost certainly necessary to subdivide the domain in which the solution is to be found, if only to allow for more complicated combinations of steps, ramps, etc.

We conclude that a good generalized discretization method should have a more natural variational functional, handle time-stepping along more conventional lines (e.g., as in finite-difference methods), but also have the capability of handling unusual functions (i.e., not just polynomials or rational functions), such as those discussed here. Naturally, we have in mind cell discretization as an example of such an

algorithm, insofar as we have already used nonpolynomial representations within its framework, to solve a (simple) eigenvalue problem in [6].

We have so far said nothing about the extension to higher dimensions of the use of nonrational representations. The first problem would be that of matching these approximations across interfaces and at boundaries. We believe that the method of interface moment collocation described in [6] would serve to handle this problem in a natural way, by allowing for interface discontinuities of the representation, in a properly controlled manner.

The second problem would be that of a suitable treatment of the "geometrical" aspects of the steep-front representations in higher dimensions. Instead of a simple steplike transition at one point, as we have in 1D, we must consider a front which is in the shape of a *curve* (in 2D; a surface in 3D, etc.). Instead of the equation for a *point*, as we obtain by setting $v = 0$ in Eq. (3.2), we must construct a function $W(x, y; a)$ such that setting $W = 0$ defines the curve on which the front lies. (We are using the symbol "a" to represent a generic set of parameters; that is, whatever is needed in the problem). For example, if W is a quadratic function of x and y, we can have any conic section be the curve of the front. The next step is to multiply W by a steepness parameter a_1, and define v, the argument of the step-function $S(v)$ by

$$v \equiv (a_1 W(x, y; a). \tag{6.1}$$

Again assuming only two "regimes" in the subdomain we are working n, we form the complete representation of the solution exactly as in Eq. (3.4), except that Q_1 and Q_2 are functions of x and y, as well as the additional parameters needed. This is a reasonable framework for extending this method to higher dimensions, but certainly there are formidable new problems that will arise.

One of these is that of formulating the right variational functional from which to derive the equations which will determine the evolution of the a-values through time. The semi-discretization described in Section 2 leads a set of ODEs that is very sensitive to errors in quadrature, the condition of the A-matrix, etc. This is the main reason why it seems preferable to solve for the a's, time-slice by time-slice, even though this would involve the solution of a highly nonlinear system of equations at every time step.

There also remains the very difficult problem of finding higher dimensional quadrature algorithms that would be appropriate for this formulation of the steep-front problem. There is a serious question as to whether an efficient adaptive quadrature method is in fact feasible; considering the formidable integrands in $\{A_{ij}\}$, this is indeed doubtful.

Finally, we still have the problem in higher dimensions of somehow knowing ahead of time what sort of shapes, how many, and where, the special representation will have to approximate within a given subdomain. This might be more of a logistical problem than a serious theoretical one, but it is certainly non-trivial. Obviously, as in 1D, if a pulselike function is called for, a steplike one will not do, nor will a single step suffice if more than one actually occurs in the true solution. Within the limited

time allocated to us for this investigation, we were not able to address these rather daunting problems.

APPENDIX: BRIEF DESCRIPTION OF THE RUNGE-KUTTA–FEHLBERG METHOD

Let y be a vector of N variables $\{y_i(t)\}$, each depending on the time t, and let us seek to find a numerical solution to the system of ordinary differential equations (ODEs)

$$\dot{y} = f(y, t), \tag{A1}$$

where $f(y, t)$ is the vector of given functions $\{f_i\}$ which define the problem. We shall consider the initial value problem only, with the initial value $y(0)$ given.

The classical Runge–Kutta method is one of the class called *single step* methods because, given the value of y_{n-1} $(\equiv y(t_{n-1}))$ and a time increment h_n, such a method will yield the approximate value y_n $(\equiv y(t_n) = y(t_{n-1} + h_n))$. The well-known system of formulas for this vector is in terms of certain intermediate vectors $\{k_0, k_1, k_2, k_3\}$,

$$y_n = y_{n-1} + \tfrac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3), \tag{A2}$$

where

$$\begin{aligned} k_0 &= h_n f(y_{n-1}, t_{n-1}), \\ k_1 &= h_n f\left(y_{n-1} + \frac{k_0}{2}, t_{n-1} + \frac{h_n}{2}\right), \\ k_2 &= h_n f\left(y_{n-1} + \frac{k_1}{2}, t_{n-1} + \frac{h_n}{2}\right), \\ k_3 &= h_n f(y_{n-1} + k_2, t_{n-1} + h_n). \end{aligned} \tag{A4}$$

The truncation error E_n of this step, defined as the difference $y_n - \bar{y}_n$ (where $\bar{y}$ signifies the *exact* value of y), is known to be of the form

$$E_n = C_n h_n^5, \tag{A4}$$

where the vector C_n depends on the values of the 4th derivatives of the components of f *in the interval* $[t_{n-1}, t_{n-1} + h_n]$. This vector is not known, but it can be estimated by repeating the calculation twice; once over the first half of the interval and once over the second half. We then obtain

$$E_{1n} = C_{1n}(\tfrac{1}{2}h_n)^5, \qquad E_{2n} = C_{2n}(\tfrac{1}{2}h_n)5 \tag{A5}$$

so that the estimated truncation error of y_n^*, the value calculated with two half-intervals, is

$$E_n^* \equiv y_n^* - \bar{y}_n = E_{1n} + E_{2n} = (C_{1n} + C_{2n}) \cdot (\tfrac{1}{2}h_n)^5. \tag{A6}$$

For the purpose of estimating the actual error in terms of h_n, we must make some assumptions about the behavior of f. We assume that the vector C "does not vary too much" in the interval h_n, i.e., that

$$C_n = \bar{C}_n + O(h_n), \tag{A7}$$

where $\bar{C}_n$ is some mean value in h_n. The same is assumed for C_{1n} and C_{2n}. Combining (A4), (A6), and (A7), we have, for the difference

$$\Delta_n \equiv y_n^* - y_n = E_n^* - E_n = (\tfrac{2}{32} - 1) \cdot \bar{C}_n h_n^5 + O(h_n^6) \tag{A8}$$

so that $\bar{C}_n$ can be estimated from Δ_n (which is known). We have

$$\bar{C}_n \sim -\tfrac{16}{15} h_n^{-5} \Delta_n \tag{A9}$$

from which we obtain

$$E_n \sim -\tfrac{16}{15} \Delta_n, \qquad E_n^* \sim -\tfrac{1}{15} \Delta_n. \tag{A10}$$

Let us define the measure of relative error ρ_n to be the norm of the vector E_n/y_n^* (each of whose components is the ratio of corresponding components of E_n and y_n^*). Usually the best norm is the maximum-magnitude element of this vector of ratios and, in order to be conservative, we use the "coarse" value E_n, but the more accurate, "fine" value y_n^*. If ε_n is the relative error bound on y_n, we require that

$$\rho_n \leqslant \varepsilon_n. \tag{A11}$$

Clearly, this requirement will be too stringent if some component of y_n^* should happen to be too close to zero; in this case, an alternative criterion must be used. However, we shall not go into these "parasitic" complications, which arise in any implementation of any algorithm.

Rather, we shall look next at the amount of work that must be done to evaluate y_n and y_n^*. Three new evaluations of f are needed for the first half of y_n^* (since k_{01} is already known, being equal to k_0), and an additional four for the second half of y_n^*. This makes a total of $4 + 3 + 4 = 11$ evaluations for one (compound) step. In an effort to decrease the work necessary to obtain estimates of the truncation error of an integration step, i.e., with fewer evaluations of f, Fehlberg [7] succeeded in finding the right special case of the *generalized* Runge–Kutta algorithm, involving only 6 evaluations of f. The general formulas read

$$y_n = y_{n-1} + \sum_{i=1}^{6} \gamma_i k_i, \qquad y_n^* = y_{n-1} + \sum_{i=1}^{6} \gamma_i^* k_i, \tag{A12}$$

where

$$k_i = h_n f\left(y_{n-1} + \sum_{j=1}^{i-1} \beta_{ij} k_j, t_{n-1} + \alpha_i h_n\right). \tag{A13}$$

The α's and γ's are 6-vectors, and the β's may be regarded as forming a 6×5 matrix. These quantities must satisfy the constraints

$$\sum_i \gamma_i = \sum_i \gamma_i^* = 1,$$
$$\sum_j \beta_{ij} = \alpha_i, \tag{A14}$$
$$\beta_{ij} = 0 \qquad \text{for} \quad i \leqslant j,$$

but are otherwise arbitrary. Fehlberg managed to find a set of values of $\{\alpha_i\}$, $\{\gamma_i\}$, $\{\gamma_i^*\}$, and $\{\beta_{ij}\}$ (these are displayed in [4]) such that y_n is correct to the *fourth* orer in h_n, while y_n^* is correct to the *fifth* order (our notation differs at this point from that used in [4]). This enables us to compute a "fine" y_n^*, and also an error estimate (which we shall again denote by E_n), using in all only 6 evaluations of f instead of 11.

Given these estimates for the nth interval, our next objective is to use them to predict h_{n+1}, such that E_{n+1}, when calculated, will in all likelihood satisfy the error criterion for the $(n+1)$th interval. We still have, for E_{n+1},

$$E_{n+1} = C_{n+1} h_{n+1}^5, \tag{A15}$$

and we assume that C and y do not vary too drastically from one interval to the next; in fact, we assume that the value $\| C/y^* \|$ has the property

$$\| C_{n+1}/y_{n+1}^* \| \leqslant M \| C_n/y_n^* \|, \tag{A16}$$

where M is some positive number of modest size. We then have the relations

$$\begin{aligned} \rho_{n+1} &\sim \| C_{n+1}/y_{n+1}^* \| \, h_{n+1}^5 \leqslant M \| C_n/y_n^* \| \, h_{n+1}^5 \\ &\sim M \| E_n/u_n^* \| \left(\frac{h_{n+1}}{h_n} \right)^5 \equiv M \rho_n \left(\frac{h_{n+1}}{h_n} \right)^5. \end{aligned} \tag{A17}$$

Now, if we replace ρ_{n+1} by the error bound ε_{n+1}, and solve for h_{n+1}, we obtain

$$h_{n+1} = h_n \left(\frac{1}{M} \frac{\varepsilon_{n+1}}{\rho_n} \right)^{1/5} \tag{A18}$$

from which h_{n+1} may be estimated. In practice, a value for M of 2^5 was found to work well, which means that the otherwise "bare" estimate for h_{n+1} was divided in half.

As remarked previously, there are many special cases and arbitrary "safety" procedures in any adaptive code, which we shall not go into here in detail.

ACKNOWLEDGMENTS

I am indebted, for useful discussions and advice, to Joe Oliger, Ray Chin, Baxter Armstrong, Sandy Roberts, Dick Stillman, Piero Sguazzero, and, especially, to Alan Karp. Thanks are also due to the referees, whose criticisms were very helpful with the clarification and enhancement of the presentation.

REFERENCES

1. P. CONCUS AND W. PROSKUROWSKI, *J. Comput. Phys.* **30**, (1979), 153–166.
2. R. J. GELINAS, S. K. DOSS, AND K. MILLER, *J. Comput. Phys.* **40** (1981), 202–249.
3. A. T. MARKUSHEVICH, "Theory of functions of a Complex Variable," Vol. II. p. 298, Prentice–Hall, Englewood Cliffs, N.J., 1965.
4. G. E. FORSYTHE, M. A. MALCOLM, AND C. E. MOLER, "Computer Methods for Mathematical Computations," Prentice–Hall, Englewood Cliffs, N.J., 1977.
5. A. C. HINDMARSH, "GEAR: Ordinary Differential Equation Solver," Lawrence Livermore Laboratory Report, No. UCID-30001, Rev. 3, December 1974.
6. J. GREENSTADT, *SIAM J. Sci. Stat. Comput.* **3** (1982), 261–288.
7. E. FEHLBERG, *Computing* **6** (1970), 61–71.