SIAM J. SCI. COMPUT.
Vol. 21, No. 2, pp. 405–420

SOLVING HYPERBOLIC PDEs USING INTERPOLATING WAVELETS*

MATS HOLMSTRÖM†

Abstract. A method is presented for adaptively solving hyperbolic PDEs. The method is based on an interpolating wavelet transform using polynomial interpolation on dyadic grids. The adaptability is performed automatically by thresholding the wavelet coefficients. Operations such as differentiation and multiplication are fast and simple due to the one-to-one correspondence between point values and wavelet coefficients in the interpolating basis. Treatment of boundary conditions is simplified in this sparse point representation (SPR). Numerical examples are presented for one- and two-dimensional problems. It is found that the proposed method outperforms a finite difference method on a uniform grid for certain problems in terms of flops.

Key words. interpolating wavelet transform, PDEs, hyperbolic equations, finite difference methods, mesh refinement

AMS subject classifications. 65D05, 76M25, 65N50

PII. S1064827597316278

1. Introduction.

Solutions to PDEs often behave differently in different areas. In acoustics an example is a low-frequency wave, with a localized high-frequency burst. In fluid dynamics we have shocks, boundary layers, and turbulence. For these examples the solution can be smooth in most of the solution domain, with small areas where the solution changes quickly. When solving such problems numerically we would like to adjust the discretization to the solution. In terms of finite difference methods, we want to have many points in areas where the solution has strong variation and few points in areas where the solution is smooth. If we use a Galerkin method, this corresponds to the representation of the solution having fewer basis functions in the smooth areas.

Given a wavelet representation of a function

$$\sum_k s_{J,k}\varphi_{J,k}(x) + \sum_{j,k} d_{j,k}\psi_{j,k}(x),$$

where $\varphi_{J,k}(x)$ are scaling functions and $\psi_{j,k}(x)$ are wavelets, the scaling function coefficients, $s_{J,k}$, essentially encode the smooth part of the function, while the wavelet coefficients, d_{jk}, contain information on the functions behavior on successively finer scales. The most common way of compressing such a representation is thresholding. We delete all wavelet coefficients of magnitude less than some threshold, ϵ. If the total number of coefficients in the original representation was N, we have N_s significant coefficients left after the thresholding. Note that by thresholding a wavelet representation we have a way to automatically find a sparse representation, and we can also use this representation to compute function values at any point.

For the above-mentioned examples of solutions of PDEs, N_s will be much smaller than N. This is due to the locality of the wavelets. The representation will have many wavelets in areas where the solution varies but few in the smooth parts. This sparse wavelet representation has two advantages when solving a PDE numerically.

*Received by the editors, February 10, 1997; accepted for publication (in revised form) July 15, 1998; published electronically September 29, 1999.
http://www.siam.org/journals/sisc/21-2/31627.html
†Swedish Institute of Space Physics, Box 812, S-981 28 Kiruna, Sweden (matsh@irf.se).

First, we need less memory to store the solution since we store N_s coefficients instead of N. Second, we could save computational time if we can exploit this sparsity when differentiating and multiplying functions. Ideally we would like the number of floating point operations needed to perform these operations to be proportional to N_s, with a small constant.

Several different approaches have been considered for solving time-dependent PDEs exploiting the sparsity of a wavelet representation. One way is to approximate the operators of the PDE in a wavelet basis. This can be done with a collocation method, as done by Bertoluzza [2], or with a Galerkin method, as done by Bacry, Mallat, and Papanicolao [1] and by Holmström and Waldén [12]. The operators will be sparse in the wavelet basis but N_s needs to be very small compared to N for the methods to be efficient. This is especially true for nonlinear operations such as multiplication of functions (as was noted, in [12] and others).

To overcome this problem one can choose an approach similar to what is done when using pseudospectral methods. Then one uses both the physical representation of the solution and the wavelet representation. One can then do multiplication in the physical space and differentiation in the wavelet space, as is done by Beylkin and Keiser [4] with coiflets; it can also be done by using collocation methods by Charton and Perrier [5], Fröhlich and Schneider [9], Ponenti and Liandrat [15], and Vasilyev and Paolucci [16]. With this approach one transforms back and forth between the physical domain and the wavelet domain in each time step, which introduces some overhead, especially if the support of the wavelet is large.

Again, another approach is to do all operations in the physical representation of the solution and use only wavelets to construct and update the representation. This promises low computational overhead, since we are working with point values in the physical representation. Along these lines, Jameson [14] uses wavelets for finding where to refine the grid in a finite difference method and then uses finite difference stencils on an irregular grid. Waldén describes a filter bank method in [18]. Also, Harten [11] and Gerritsen and Olsson [10] have used wavelets to localize where to apply different types of finite difference methods.

In this paper a representation in physical space, the *sparse point representation* (SPR), is introduced. It is based on an interpolating wavelet transform using polynomial interpolation on dyadic grids. The interpolating wavelet transform has also been used in a Galerkin method for solving elliptic problems on the interval by Bertoluzza, Naldi, and Ravel [3]. A feature of the chosen basis is the one-to-one correspondence between point values and wavelet coefficients. Also the treatment of boundary conditions is simplified in the SPR. Centered finite differences are used for approximating space derivatives, and we then use a Runge–Kutta method to advance the solution in time. Numerical examples are presented for the advection equation and Burgers's equation in one and two dimensions, with periodic and nonperiodic boundary conditions.

A problem with many adaptive wavelet methods, as noted in [5, 9], is that although multiplication and differentiation are done in $\mathcal{O}(N_s)$ time, the constants can be large. Using the SPR with finite differences we show, by numerical examples, that these constants are small.

In section 2 we describe the interpolating wavelet transform and introduce the sparse wavelet and point representation, along with operations on these representations, and we also describe the time stepping method. In sections 3 and 4 we present the results of numerical experiments in one and two dimensions for linear and nonlin-

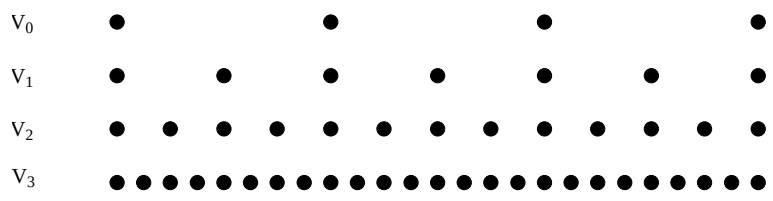


FIG. 2.1. *Examples of point positions on dyadic grids.*

ear problems.

2. Theory. Wavelets are usually introduced by defining scaling functions, $\varphi_{j,k}$, wavelets, $\psi_{j,k}$, and the associated function spaces V_j and W_j. Since we are using an interpolating wavelet transform, it is possible to define instead the transform in terms of interpolation on dyadic grids. First we present the interpolating subdivision scheme by Deslauriers and Dubuc [6] and Dubuc [8].

Assume that we have a set of dyadic grids on the real line,

$$V_j = \{x_{j,k} \in \mathbb{R} : x_{j,k} = 2^{-j}k, k \in \mathbb{Z}\}, \qquad j \in \mathbb{Z}.$$

The locations of points on such dyadic grids are illustrated in Figure 2.1. Given function values on V_j, $\{f_{j,k}\}_{k\in\mathbb{Z}}$, where $f_{j,k} = f(x_{j,k})$, a function defined on the grid points in V_j, we would like to extend them to all points $f_{j+1,k}$ in V_{j+1}. The interpolating subdivision scheme is an algorithm to accomplish this. The even-numbered grid points $x_{j+1,2k}$ already exist in V_j, and the corresponding function values are kept unchanged. Values at the odd-numbered grid points $x_{j+1,2k+1}$ are computed by polynomial interpolation from the values at the even-numbered grid points. The degree of this interpolating polynomial is $p-1$, and we say that the interpolation is of order p. The order is chosen to be even to make the interpolation symmetric. The adaption to boundaries is simple. We use the closest points inside the boundary on the coarser grid to define the interpolating polynomial. In the multidimensional case we use bases constructed by tensor products of the one-dimensional V_j spaces.

Used recursively, the interpolating subdivision scheme generates function values on a fine grid, given values on a coarse grid. If we wanted to do the opposite, go from a fine to a coarse grid, we could just throw away half of the grid points at each level, but we would then lose information. Instead we can, at each level, for odd-numbered grid points, compute the difference between the known function value and the function value predicted by the interpolation from the coarser grid. We call these differences in function values wavelet coefficients, $d_{j,k}$. The computation of a wavelet coefficient is illustrated in Figure 2.2 for the cubic case, $p = 4$. Repeating this recursively we have an algorithm for computing the full wavelet representation from function values on a fine grid. In Figure 2.3 we have an illustration of such a wavelet representation. This interpolating wavelet transform was introduced, independently, by Donoho [7] and Harten [11].

How do we reverse the process? The inverse transformation simply takes the levels in the opposite order and adds the correction, $d_{j,k}$, to the interpolated prediction. This transform is a special case of the well-known fast wavelet transform.

The reason that we are interested in wavelet methods in the first place is the possibility of compressing representations of functions in a wavelet basis. This is usually done by the thresholding of the wavelet coefficients; i.e., we remove all wavelet

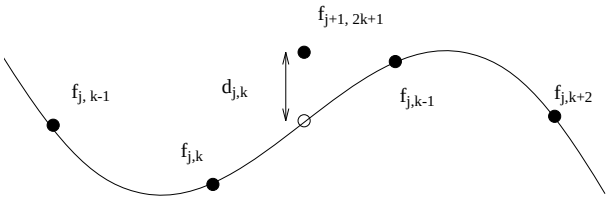


FIG. 2.2. *Generating a wavelet coefficient $d_{j,k}$ in W_j by cubic interpolation from function values on V_j.*

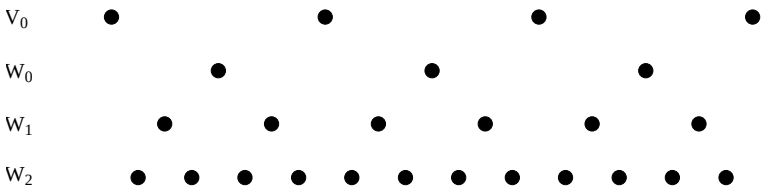


FIG. 2.3. *Example of a wavelet representation where each point, except on the coarsest grid, corresponds to a wavelet coefficient.*

coefficients whose magnitude is smaller than some threshold, ϵ. If we started with N coefficients on a fine grid we have N_s coefficients after thresholding.

Operations such as differentiation and, especially, multiplication can be costly when done in a wavelet basis. On the other hand, the wavelet basis is needed for thresholding the representation into a sparse representation. Ideally we would like to transform our N_s wavelet coefficients to N_s point values, instead of the N points we get if we do an ordinary full inverse transform on our wavelet representation. Such a transform exists for the interpolating wavelets that we have described, due to the one-to-one correspondence between wavelet coefficients and point values. Consider a sparse wavelet representation. We do the inverse transform, though *only* for those points that correspond to the N_s significant wavelet coefficients. If any point value is needed that does not exist, we interpolate the value from a coarser scale, recursively. The algorithm will terminate since we have all function values on the coarsest grid. We call this the inverse sparse wavelet transform that leads to a SPR. The SPR is the set of function values retained after thresholding, i.e., all the function values on the coarsest scale and the function values whose corresponding wavelet coefficients' magnitude is greater than the threshold, ϵ.

Note that the SPR is *not* a representation in a basis; it is just a collection of point values. Given an SPR, we can also do the reverse—reconstruct the corresponding sparse wavelet representation. This sparse wavelet transform only requires $\mathcal{O}(N_s)$ flops, with a small coefficient. This is the heart of our method and the reason for our choice of scaling function and wavelet. We also note that the SPR can be computed without explicitly forming a sparse wavelet representation. This is done by storing the point values in the SPR instead of the wavelet coefficients when computing the wavelet transform. The wavelet coefficients are then computed only to decide whether the corresponding point value is to be included in the SPR.

When forming the SPR we can start with function values on a coarse grid and add points on the next finer grid that corresponds to points with wavelet coefficients

whose magnitude is larger than ϵ. This is done recursively. The advantage is that only N_s function values are involved, instead of the N function values needed in a full transform. The disadvantage is that we might miss fine scale features. We might also get fictitious large scales if the function only has high-frequency components. This can be avoided by choosing the coarsest grid fine enough to capture all the features of the function.

If we want to multiply two SPRs we simply multiply the point values. If a point value is missing, it is again interpolated from a coarser scale in the SPR.

For differentiation we use a finite difference stencil of order p (the same order as that of the representation). For each point where the derivative is needed, we locate the closest point in the SPR and choose the distance to that point as the step length, h. We then apply a centered finite difference stencil of order p. Again, if any point is missing we interpolate the value from a coarser scale. If any point in the stencil is located outside the boundary, we use a one-sided stencil, of the same order as the centered one, such that all points are inside the boundary, just as we did for the wavelet transform on an interval.

We now put all the pieces together and describe how to solve the initial boundary value problem

$$\begin{cases} u_t = \mathcal{P}u, & u = u(x,t) \in \mathbb{R}^m, \quad t > 0, \quad x \in \Omega \subset \mathbb{R}^n, \\ u(x,0) = u_0(x), & \text{with boundary conditions on } \partial\Omega, \end{cases}$$

by the SPR method. Here $\partial\Omega$ is the boundary of Ω, a box in $\mathbb{R}^n$. We assume that $\mathcal{P}$ can be evaluated in terms of repeated differentiation, multiplication, and summation. We choose to separate the time and space discretizations by using the method of lines, thus we will only discuss the space-discretization. The resulting system of ODEs can then be solved by any standard ODE solver. We transform, threshold, and inverse transform the initial function $u_0(x)$ and have an SPR at $t = 0$. The set of retained coefficients will now need to change over time so we update the SPR after every time step. This means that we make all computations in the SPR, except after every time step, when we threshold. To allow the basis to change we have to extend the SPR after the thresholding. First, in space, we add points corresponding to neighbor wavelets in the SPR. The number depends on the PDEs wave propagation speed. Then the basis is able to adjust when the solution is moving. Second, in frequency, we add points corresponding to wavelets on the next finer scale to the SPR. This refinement allows the development of, e.g., shocks in the solution. The boundary conditions are applied after each time step.

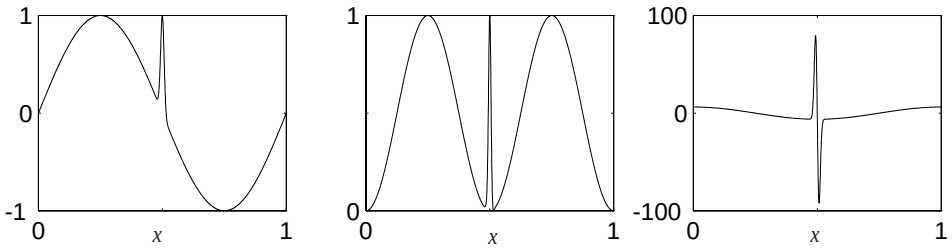
We note that for systems of PDEs we can choose how to represent the different components of solution. We could use the union of the components' SPRs, but maybe more interesting is to let each component have a different SPR. We can then interpolate the needed point values from the SPRs. This could lead to savings in the memory needed to store the solution.

An overview of the algorithm follows:

1. Transform and threshold the initial function u_0 to an SPR and set $t \leftarrow 0$.

2. Extend the SPR in space and frequency by adding points corresponding to neighbor wavelets and wavelets on the next finer scale. This is to allow for features of the solution that move or develop in time.

3. The discretization of the initial boundary value problem is a system of ODEs that is advanced to time $t \leftarrow t + \Delta t$ by any standard method, e.g., a Runge–Kutta method. Here Δt is related to the smallest distance between points in the SPR by a



(a) (b) (c)

FIG. 3.1. (a) *The test function* $f(x) = \sin(2\pi x) + \exp(-\alpha(x - 1/2)^2)$ *when* $\alpha = 10^4$, (b) *the square of the test function, and* (c) *the derivative of the test function.*

CFL-condition.

4. The SPR is thresholded.
5. Goto 2.

3. One-dimensional numerical experiments. We now turn to some numerical examples in one dimension. In section 3.1 we study some properties of the SPR on the unit interval. We examine the error of the representation itself and the error arising when multiplying and differentiating functions. Then we apply the SPR method for solving PDEs to a linear, constant coefficient, advection equation in section 3.2. In section 3.3 we then solve a nonlinear problem: Burgers's equation.

3.1. Representation, multiplication, and differentiation. First we choose a test function that should be well compressed in a wavelet basis, in this case a function on the unit interval that is smooth in most of the domain with a small interval of sharp variation. The function is a superposition of a sine function and a Gaussian and is displayed in Figure 3.1 along with its square and derivative.

We will now examine the error arising when approximating this function by its SPR. Since the SPR consists of point values, it is natural to measure the error in the maximum norm. We introduce the norm

$$|g|_\infty = \max_{0 \le x \le 1} |g(x)|.$$

Denote by $\mathcal{P}_j^\epsilon f$ the thresholded interpolant of $f(x)$ in V_j. Let us define the interpolation $\mathcal{P}^\epsilon f = \mathcal{P}_j^\epsilon f$, where j is chosen large enough such that the SPR is the same for $\mathcal{P}_j^\epsilon f$ and $\mathcal{P}_{j+1}^\epsilon f$. We are interested in the error's dependence on our threshold parameter ϵ. In what follows, c_i denote constants for a given f. Donoho [7] has shown that, for smooth enough f,

$$|f - \mathcal{P}^\epsilon f|_\infty \le c_1 \epsilon \tag{3.1}$$

(actually, part of the constant c_1 depends on -$\log_2 \epsilon$), and the number of significant coefficients, N_s, dependent on ϵ is

$$N_s \le c_2 \epsilon^{-1/p} \tag{3.2}$$

or, equivalently,

$$\epsilon \le c_2^p N_s^{-p}. \tag{3.3}$$

Combining (3.1) and (3.2) we have a bound on the error in terms of N_s,

$$|f - \mathcal{P}^\epsilon f|_\infty \leq c_3 N_s^{-p}. \tag{3.4}$$

This means that our sparse interpolating wavelet representation is of order p *in the number of significant coefficients*, N_s. We have verified these relations numerically for the test function shown in Figure 3.1.

When thresholding an SPR we need to interpolate values at all points that correspond to wavelet coefficients. What amount of work will be associated with this interpolation? Interpolating a function value requires 2 and 6 flops when $p = 2$ and $p = 4$, respectively, *if* the points from which we interpolate are included in the SPR. Otherwise we have to interpolate the needed function values recursively. This will increase the total flop count. On the other hand, we have function values at the coarsest grid, corresponding to scaling function coefficients, that need not be interpolated at all. This will decrease the total flop count. Numerical experiments suggest that we never need more than 2 and 6 flops per coefficient—thus thresholding is not expensive, even if done recursively.

Let us now turn to multiplication of functions. To avoid, for now, the question of how to choose the sparse representation of the result, we consider taking the square of our test function by squaring the point values in the SPR. The square of the test function was shown in Figure 3.1. Examining the error of the square approximation we find that, using (3.1) and (3.3),

$$|f^2 - (\mathcal{P}^\epsilon f)^2|_\infty \leq |f + \mathcal{P}^\epsilon f|_\infty \cdot |f - \mathcal{P}^\epsilon f|_\infty \leq \tilde{c}_1 \epsilon \leq \tilde{c}_3 N_s^{-p}. \tag{3.5}$$

What happens when we differentiate a function? Will the approximate derivative be of order p? In contrast to representing and squaring a function, the error at the points in the SPR will not be exactly zero when differentiating a function. Therefore we can choose to measure the error in maximum norm over the points in the SPR. We introduce the norm

$$|g|_{\epsilon,\infty} = \max_{x_{j,k}} |g(x_{j,k})|,$$

where $x_{j,k}$ are the points in the SPR. The pointwise error in the derivative approximation will then be

$$|f' - D_p \mathcal{P}^\epsilon f|_{\epsilon,\infty},$$

where D_p is the finite difference approximation of the derivative of an SPR of order p, as described in section 2. We have a truncation error for the finite difference approximation of the first derivative of the order h^p. The function values used in the finite difference approximation also have an error of order h^p due to the interpolation. We divide by h when forming the finite difference approximation leading to a total error of order h^{p-1}. We have that $h^p \sim \epsilon$ since the interpolation error is bounded by epsilon. Using (3.3) we have that $h \sim N_s^{-1}$; thus the first derivative error will be proportional to N_s^{-p+1}. We lose one order of approximation when differentiating the SPR compared with the order of the representation. The derivative of the test function was shown in Figure 3.1. Now we numerically examine the pointwise error of the approximation $D_p \mathcal{P}^\epsilon f$ as a function of N_s when $p = 2$ and $p = 4$ as shown in Figure 3.2. We find that the slopes of the graphs are approximately -1.5 and -3.5. We seem to lose only about half an order of approximation when differentiating the

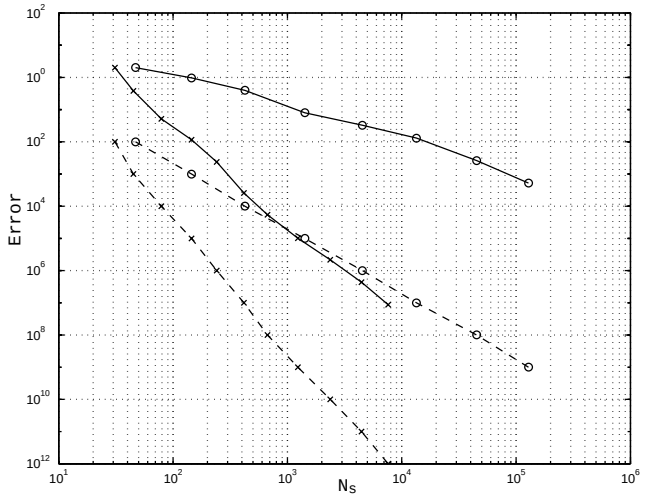


FIG. 3.2. *The pointwise error for the approximation of f' as a function of N_s (solid lines) when $p = 2$ (circles) and $p = 4$ (stars) for $\alpha = 10^4$. The dashed lines are the corresponding thresholds, ϵ, as a function of N_s.*

SPR. We also note that the error is approximately 1000ϵ and 100ϵ when p is 2 and 4, respectively, for this specific function.

What is the work associated with differentiating an SPR? At each point we have p points in the finite difference stencil. If a point is missing, we have to interpolate from a coarser scale. If we, for the moment, disregard the work associated with this interpolation, we need $2N_s$ flops when $p = 2$ and $6N_s$ flops when $p = 4$. In numerical experiments we found that for $p = 2$, we approach 2 flops/N_s as N_s grows (ϵ decreases). When $p = 4$, we approach 7 flops/N_s. Comparing this with the above stated work estimates of 2 and 6 flops/N_s, we conclude that the additional work for recursively interpolating function values during differentiation is small.

3.2. A linear problem. In this section we examine the performance of the proposed SPR method when solving a linear advection equation on the unit interval. Specifically, we will solve

$$\begin{cases} u_t = u_x, & 0 \le x \le 1, & t > 0, \\ u(x,0) = u_0(x), & u = u(x,t), & u(1,t) = u_0(t). \end{cases} \tag{3.6}$$

The left boundary is an outflow boundary and the right boundary is an inflow boundary. The solution, $u(x,t) = u_0((x+t) \bmod 1)$, is a periodic translation of the initial function. Note that the boundary conditions are chosen so that the solution will be the same as for a problem with periodic boundary conditions. As an initial function we choose the function

$$u_0(x) = \sin(2\pi x) + e^{-\alpha(x-1/2)^2}.$$

This is the test function used in the previous section, shown in Figure 3.1. This mostly smooth function with a peak will test our method's ability to follow features of the solution that move in time. In this case the grid refinement should follow the peak in

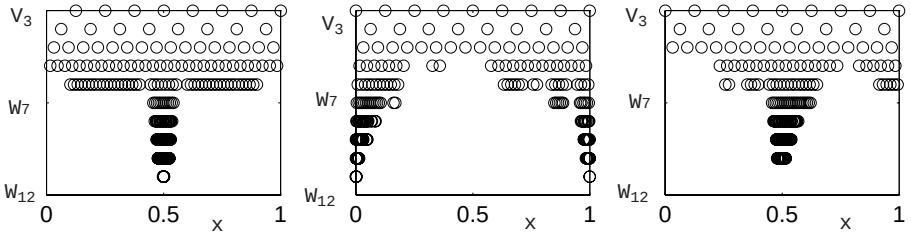


FIG. 3.3. *The sparse wavelet representation of the solution: at* $t = 0$ (a), *at* $t = 0.5$ (b), *and at* $t = 1$ (c) *when* $p = 4$, $\epsilon = 10^{-5}$, *and* $\alpha = 10^4$.

the solution. After space discretization, forming the SPR, we have a system of ODEs to solve. We solve this system using the classical fourth-order Runge–Kutta method. Since we are interested in the error from the space discretization we choose the time step small enough that this error dominates the total error. The time step, Δt, is chosen as $\Delta t = kh_{\min}$, where $h_{\min}$ is the smallest distance between points in the current SPR. In all the experiments of this section $k = 0.5$ was used.

The sparse wavelet representation at different times is displayed in Figure 3.3. We see that the refinement follows the peak in the solution. If we examine the number of significant coefficients as a function of time, we find that N_s is oscillating around a mean value of 217. That N_s does not increase significantly over time shows that the refinement automatically follows the peak in the solution.

We now compare the error at $t = T$ with the work needed to compute the solution for different values of ϵ. We also make a comparison with a finite difference method on a uniform grid. First compare some work estimates for a finite difference method of order p on a uniform grid. Let $N = 1/h$ be the number of points in the space discretization. Let $m = T/\Delta t$ be the number of time steps, where T is the final time of the solution. The amount of work in flops is then

$$\text{flops} \sim m \cdot N \sim N^2,$$

since we have a CFL-condition, $\Delta t \sim h \sim 1/N$. Assuming that the time discretization is at least as accurate as the space discretization we can estimate the error at time T, $e(T)$, as

$$|e(T)|_\infty \sim N^{-p} \quad \text{or} \quad |e(T)|_\infty \sim (\text{flops})^{-p/2}.$$

Thus we have an estimate of the error at T in terms of the number of points, N, or the work in flops. For the SPR method, instead of N we have N_s and the time step $\Delta t \sim h_{\min}$. Due to this dependence on $h_{\min}$ we no longer have any simple bound on $|e(T)|_\infty$.

In Figure 3.4 $|e(1)|_{\epsilon,\infty}$ is plotted as a function of the time average of N_s for the SPR method when $p = 4$ and for a fourth-order finite difference method as a function of the number of grid points. The slope for the finite difference method is -4; for the SPR method the slope is similar, thus the SPR method seems to be of the same order as the finite difference method. We also note that the error is approximately 1000ϵ for this specific problem.

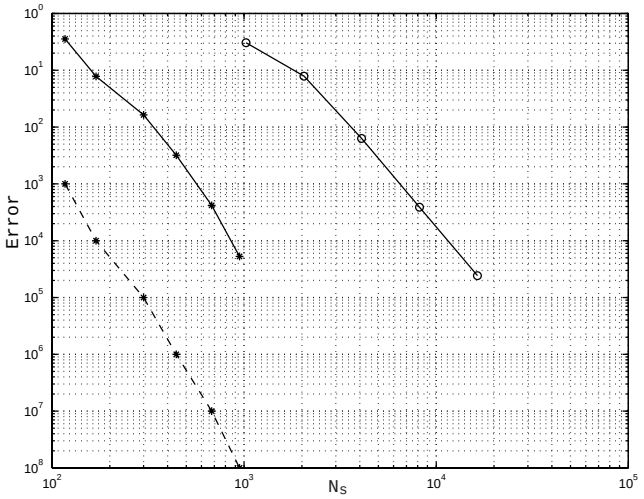


FIG. 3.4. *The error at $t = 1$ for the SPR method when $p = 4$ (solid line with stars) and a fourth-order finite difference method (circles) as a function of the time average of the number of significant coefficients. The initial function has $\alpha = 10^5$. The dashed line is the threshold for the SPR method, ϵ, as a function of N_s.*

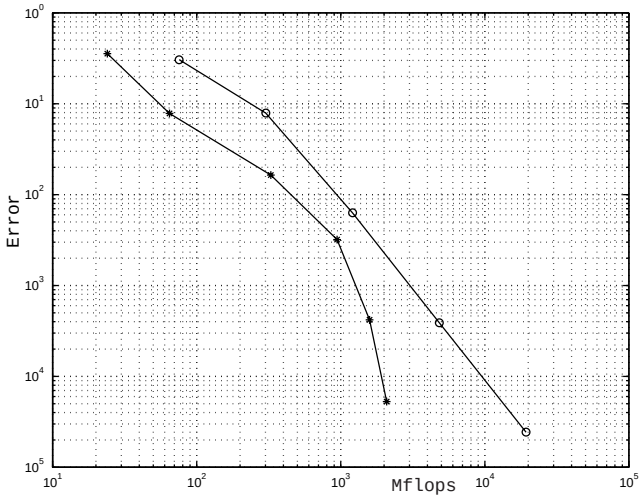
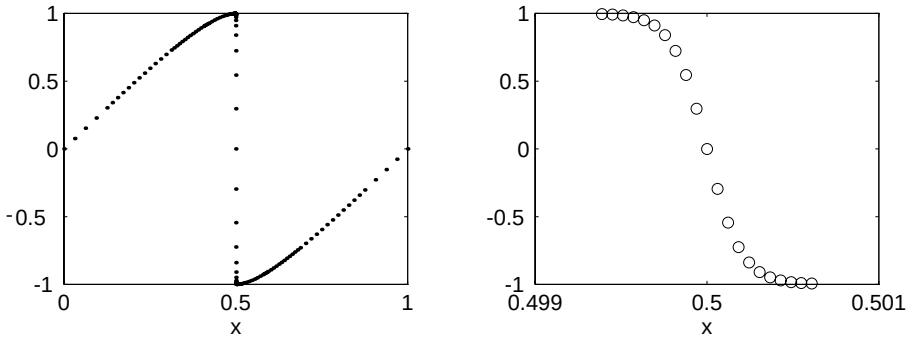


FIG. 3.5. *The error at $t = 1$ for the SPR method when $p = 4$ (stars) and a fourth-order finite difference method (circles) as a function of Mflops (Megaflops used in the computation). The initial function has $\alpha = 10^5$.*

In Figure 3.5 we plot $|e(1)|_{\epsilon,\infty}$ as a function of the amount of work in flops. Here also, the orders of the methods are roughly similar, thus the SPR method, for this specific problem when $p = 4$, seems to preserve the order of the work required when we want accurate solutions.

3.3. A nonlinear problem. The linear advection problem in the previous section was a test of the SPR method's ability to follow features in the solution auto-



(a) (b)

FIG. 3.6. *The solution at* $t = 0.25$ (a) *and a blowup around the shock* (b) *when* $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$, *and* $k = 0.25$.

matically. Now we will test the methods ability to automatically refine the SPR when features develop in time. We will solve the nonlinear Burgers's equation on the unit interval with Dirichlet boundary conditions,

$$\text{(3.7)} \qquad \begin{cases} u_t + uu_x = \mu u_{xx}, & 0 \le x \le 1, & t > 0, \\ u(x,0) = u_0(x), & u = u(x,t), & u(0,t) = u(1,t) = 0. \end{cases}$$

The initial function is a sine wave,

$$u_0(x) = \sin(2\pi x).$$

The solution will develop a sharp gradient at $x = 1/2$, which will reach its maximum around $t = 0.25$ when the extrema of the initial function have advected to $x = 1/2$ and the solution is close to a saw-tooth function when the viscosity, μ, is small. Again, this is a function that should be well compressed in a wavelet basis since it is smooth in most of the domain, except for a small interval of sharp variation. This will be a good test of our methods ability to refine the representation frequency-wise, that is, the ability to refine the grid around $x = 1/2$ as the shock develops.

The computed solution, using the SPR method, at $t = 0.25$ is shown in Figure 3.6 when $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$, and $k = 0.25$. That the SPR method is able to refine the grid around $x = 1/2$ as the shock develops is evident in Figure 3.7. From a representation in V_6 at $t = 0$, we have a representation in V_{14} at $t = 0.25$; i.e, the grid is $2^8 = 256$ times finer.

4. Two-dimensional numerical experiments. Now we will examine the SPR method's performance for some two-dimensional test problems. First we examine the error when representing and differentiating a two-dimensional function in section 4.1. In section 4.2 we solve a linear advection problem on the unit square. Finally we solve a nonlinear problem in section 4.3, a two-dimensional counterpart to the one-dimensional Burgers's equation.

In all two-dimensional numerical experiments we refine one extra level for all significant coefficients. All experiments are done in the cubic basis, $p = 4$. The boundary conditions are periodic for all two-dimensional test problems.

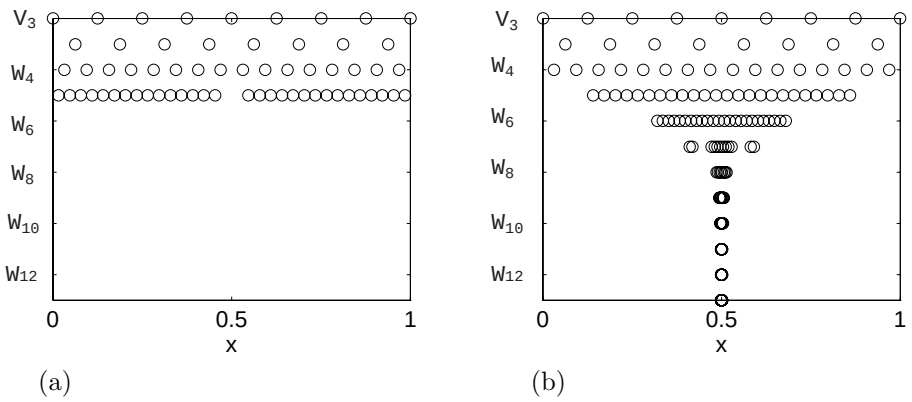


FIG. 3.7. *The sparse wavelet representation of the solution: at* $t = 0$ (a) *and at* $t = 0.25$ (b) *when* $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$, *and* $k = 0.25$.

4.1. Representation and differentiation. Multidimensional bases are constructed as tensor products of the one-dimensional V_j spaces. The implementation of the two-dimensional transform is then the one-dimensional transform applied first to rows and then to columns of the grid. Some adjustments of the method are needed in two dimensions. First we choose the space step $h_{\min}$ as the distance along the x- or y-axis to the closest point. As in the one-dimensional case, we recursively evaluate function values that are not in the SPR, but we also store the evaluated values to reduce the flop count. We now check that the order relations that we verified in the one-dimensional case hold in two dimensions. As a test function we use a function that is smooth and slowly varying, except for a small region,

$$f(x,y) = e^{-\alpha((x-1/2)^2+(y-1/2)^2)} - 0.2 \cdot \sin(2\pi x)\sin(2\pi y),$$

where the width of the peak is controlled by the parameter α. The results of numerical experiments are consistent with the relations

$$\sqrt{N_s} \le c_2 \epsilon^{-1/p}$$

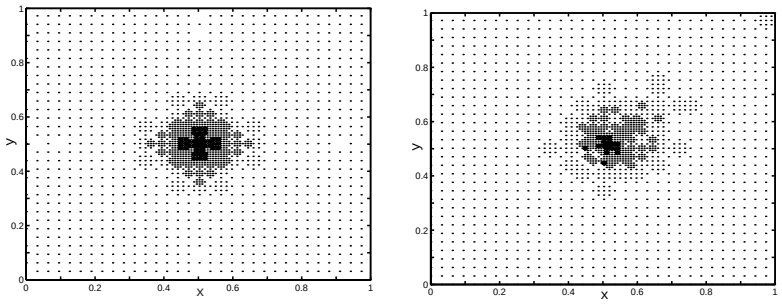
$$\text{and} \quad |f - \mathcal{P}^\epsilon f|_\infty \le c_3 (\sqrt{N_s})^{-p},$$

which are the two-dimensional counterparts to the one-dimensional estimates, (3.2) and (3.3). For the error in the derivative approximation we seem to lose half an order of approximation, as in the one-dimensional case.

The number of flops per significant coefficient for computing the derivative is at least 6, but it will be higher since we have to interpolate nonexistent function values from coarser grids. To reduce the work, as mentioned above, we temporary store every evaluated function value to avoid making the same interpolation twice. The number of flops per significant coefficient for computing the derivative is found to be around 13, higher than the lower bound of 6 but still a small number.

4.2. A linear problem. As in the one-dimensional case, we first examine a linear advection problem. Again it will test the method's ability to follow features of the solution. We will use periodic boundary conditions on the unit square to avoid influences from boundary conditions. The problem can be stated as

$$(4.1)\quad \begin{cases} u_t = u_x + u_y, & 0 \le x, y \le 1, & t > 0, \\ u(x,y,0) = u_0(x,y), & u(1,y,t) = u(0,y,t), & u(x,1,t) = u(x,0,t). \end{cases}$$



(a) (b)

FIG. 4.1. (a) *The SPR at* $t = 0$ ($N_s = 2072$) *and* (b) *the SPR at* $t = 1$ ($N_s = 1865$) *when* $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$, *and* $k = 1$.

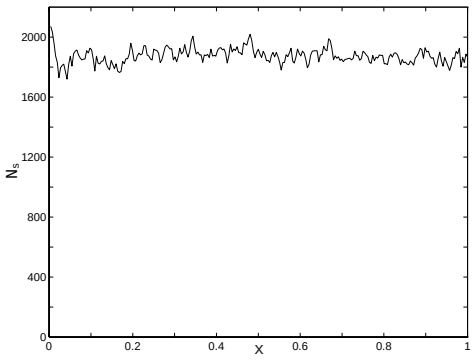


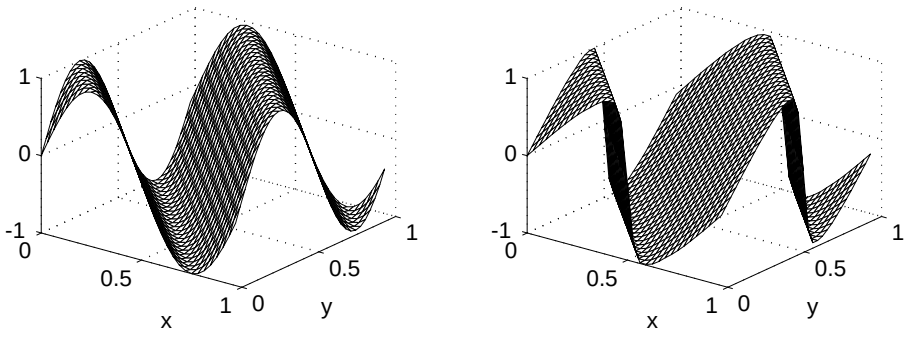
FIG. 4.2. *The number of significant coefficients as a function of time when* $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$, *and* $k = 1$.

The solution is a translation of the initial function with speed $\sqrt{2}$, $u(x, y, t) = u_0(x + t \bmod 1, y + t \bmod 1)$. The initial function is the smooth function with a localized spike,

$$u_0(x, y) = e^{-\alpha\left((x-1/2)^2+(y-1/2)^2\right)} - 0.2\sin(2\pi x)\sin(2\pi y).$$

The SPR at $t = 0$ and at $t = 1$ is displayed in Figure 4.1. We see that the points are still concentrated around the spike at $t = 1$. Thus the grid refinement follows the peak in the solution. We see that the refinement somewhat trails the peak. This is probably due to the dispersive errors of the underlying finite difference method. The important observation is that this trailing "tail" does not grow in time. This observation is verified if we examine the time evolution of the number of significant coefficients that are plotted in Figure 4.2. It oscillates, but the average is constant over time.

4.3. A nonlinear problem. Now we would like to examine the ability of the SPR method to refine the representation in two dimensions when gradients develop in the solution. We examine the following two-dimensional counterpart to the one-



(a) (b)

FIG. 4.3. (a) *The initial function and* (b) *the solution at* $t = 0.09$ *when* $\epsilon = 10^{-3}$, $\mu = 10^{-2}$, *and* $J = 7$.

dimensional Burgers's equation,[1]

$$\text{(4.2)} \quad \begin{cases} u_t + u(u_x + u_y) = \mu(u_{xx} + u_{yy}), & 0 \le x, y < 1, \qquad t > 0, \\ u(x, y, 0) = u_0(x, y), \qquad u(1, y, t) = u(0, y, t), \qquad u(x, 1, t) = u(x, 0, t), \end{cases}$$

where the boundary conditions are periodic. If we choose the initial function as a two-dimensional sine wave,

$$u_0(x, y) = \sin(2\pi(x + y)),$$

the solution along all lines parallel to the line $x = y$ will be the solution to the one-dimensional Burgers's equation, time dilated by a factor of 2. The equation is two-dimensional in the sense that shocks will develop along lines at a 45 degree angle to the x-axis. The gradient in the shock will reach its maximum at time $t = 0.125$ when the extrema of the sine wave have advected into the shock. In Figure 4.3 the initial function and the solution at $t = 0.09$ are displayed when $\epsilon = 10^{-3}$, $\mu = 10^{-2}$, and $J = 7$. The SPRs at $t = 0$ and $t = 0.09$ are shown in Figure 4.4. We see that a coarse grid is sufficient to represent the everywhere smooth solution at $t = 0$. At $t = 0.09$ the shock has started to develop, and the grid is refined around the shocks but coarsened elsewhere. We have seen that the SPR method is able to adapt the two-dimensional grid to developing features of the solution.

5. Conclusions. We have presented a method for adaptively solving hyperbolic PDEs. The adaptability is achieved by using an interpolating wavelet basis to sparsely represent the solution of the PDE. The representation automatically adapts when the solution changes over time. This adaption works both for features that are moving and for features that develop over time, such as shocks. The sparse representation leads to significant savings in the number of flops needed to achieve a solution with a certain accuracy in maximum norm. Since we in the current implementation have used a nonsparse data structure, no comparison was made in terms of CPU time. By using a sparse data structure the method should be competitive also in CPU

[1] This is the scalar counterpart to the two-dimensional system called the bidimensional Burgers's equation by Ponenti and Liandrat [15].

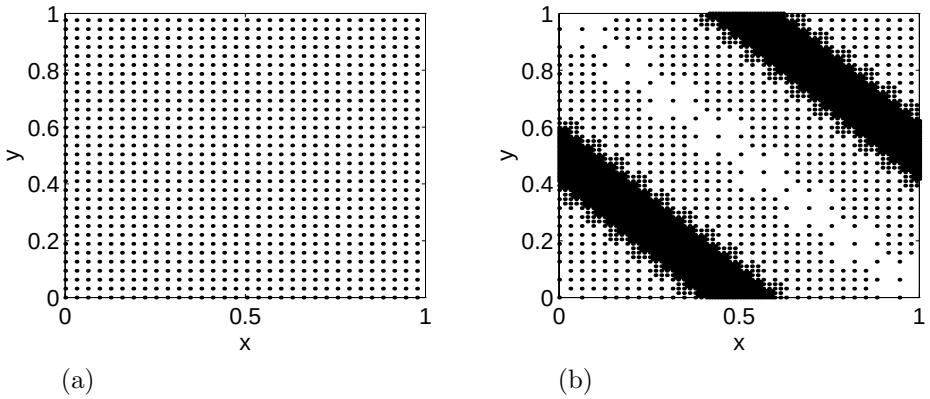


FIG. 4.4. (a) *The SPR at* $t = 0$, $N_s = 1024$ *and* (b) *the SPR at* $t = 0.09$, $N_s = 4352$ *when* $\epsilon = 10^{-3}$, $\mu = 10^{-2}$, *and* $J = 7$.

time. If a sparse data structure is used to store the SPR, we also reduce the memory requirement. The method is shown to out perform the finite difference method for these problems, in terms of flops for certain functions. These functions are those that can be well compressed in a wavelet basis, e.g., a function that is smooth in most of the domain with small areas of sharp variation. We have shown that multiplication, differentiation, and boundary conditions can be handled in a time that is proportional to the number of significant coefficients. This SPR method is easily extended to higher dimensions and systems of PDEs. Finally, the SPR method is easy both to implement and understand. An area of future work is implementing a suitable sparse data structure for the SPR.

REFERENCES

[1] E. BACRY, S. MALLAT, AND G. PAPANICOLAOU, *A wavelet based space-time adaptive numerical method for partial differential equations*, RAIRO Model. Math. Anal. Numer., 26 (1992), pp. 793–834.

[2] S. BERTOLUZZA, *Adaptive wavelet collocation method for the solution of Burgers equation*, Transport Theory Statist. Phys., 25 (1996), pp. 339–352.

[3] S. BERTOLUZZA, G. NALDI, AND J. C. RAVEL, *Wavelet methods for the numerical solution of boundary value problems on the interval*, in Wavelets: Theory, Algorithms and Applications, C. K. Chui, L. Montefusco, and L. Puccio, eds., Academic Press, New York, 1994, pp. 425–448.

[4] G. BEYLKIN AND J. M. KEISER, *On the adaptive numerical solution of nonlinear partial differential equations in wavelet bases*, J. Comput. Phys., 132 (1997), pp. 233–259.

[5] P. CHARTON AND V. PERRIER, *A pseudo-wavelet scheme for the two-dimensional Navier-Stokes equations*, Mat. Apl. Comput., 15 (1996), pp. 139–160.

[6] G. DESLAURIERS AND S. DUBUC, *Symmetric iterative interpolation processes*, Constr. Approx., 5 (1989), pp. 49–68.

[7] D. L. DONOHO, *Interpolating Wavelet Transforms*, Tech. report 408, Department of Statistics, Stanford University, Stanford, CA, 1992.

[8] S. DUBUC, *Interpolation through an iterative scheme*, J. Math. Anal. Appl., 114 (1986), pp. 185–204.

[9] J. FRÖHLICH AND K. SCHNEIDER, *An adaptive wavelet-vaguelette algorithm for the solution of PDEs*, J. Comput. Phys., 130 (1997), pp. 174–190.

[10] M. GERRITSEN AND P. OLSSON, *Designing an efficient solution strategy for fluid flows. 1. A stable high order finite difference scheme and sharp shock resolution for the Euler equations*, J. Comput. Phys., 129 (1996), pp. 245–262.

[11] A. Harten, *Adaptive multiresolution schemes for shock computations*, J. Comput. Phys., 115 (1994), pp. 319–338.
[12] M. Holmström and J. Waldén, *Adaptive wavelet methods for hyperbolic PDEs*, J. Sci. Comput., 13 (1998), pp. 19–49.
[13] L. Jameson, *On the Wavelet Optimized Finite Difference Method*, Tech. report ICASE 94-9, NASA Langely Research Center, Hampton, VA, 1994.
[14] L. Jameson, *A wavelet-optimized, very high order adaptive grid and order numerical method*, SIAM J. Sci. Comput., 19 (1998), pp. 1980–2013.
[15] P. Ponenti and J. Liandrat, *Numerical Algorithms Based on Biorthogonal Wavelets*, Tech. report ICASE 96-13, NASA Langely Research Center, Hampton, VA, 1996.
[16] O. V. Vasilyev and S. Paolucci, *A dynamically adaptive multilevel wavelet collocation method for solving partial differential equations in a finite domain*, J. Comput. Phys., 125 (1996), pp. 498–512.
[17] O. V. Vasilyev, S. Paolucci, and M. Sen, *A multilevel collocation method for solving partial differential equations in a finite domain*, J. Comput. Phys., 120 (1995), pp. 33–47.
[18] J. Waldén, *Filter bank methods for hyperbolic PDEs*, SIAM J. Numer. Anal., 36 (1999), pp. 1183–1233.