Solving Hyperbolic PDEs Using Interpolating Wavelets

by

Mats Holmström

Report No. 189/1996

UPPSALA UNIVERSITET
Institutionen för
teknisk databehandling

UPPSALA UNIVERSITY
Department of
Scientific Computing

Solving Hyperbolic PDEs Using Interpolating Wavelets

Mats Holmström [1]

December, 1996

[1]Uppsala University, Dept. of Scientific Computing, Box 120, S-751 04 Uppsala, Sweden. matsh@tdb.uu.se

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. Numerical examples are presented for one- and two-dimensional problems. It is found that the underlying finite difference method's order of convergence is preserved, and that the proposed method outperforms a finite difference method for certain problems in terms of flops.

Contents

1 Introduction

Solutions to hyperbolic partial differential equations (PDEs) often contain shocks. If we are interested in the precise location of a shock at different times, a fine computational grid is needed around the shock. Using a uniform grid we then have a fine grid in the whole computational domain. This is a waste of computational resources, since the fine grid is really only needed in the vicinity of the shock. Some sort of adaptability is needed. We would like a nonuniform grid, which is fine around shocks and coarse in areas where the solution is smooth.

Two questions arise if we want to construct such an adaptive method. Where do we refine? How do we refine? Both questions must be answered in the framework of the numerical method we use on a uniform grid. Our hope is that by thresholding the solution's expansion coefficients in a suitable basis we can answer both of these questions and achieve the goal of automatic adaptability. While the interpolating basis we will use is not, strictly speaking, a wavelet basis, many of the ideas steam from previous work done on using wavelet methods for solving hyperbolic PDEs.

The idea behind solving hyperbolic PDEs in a wavelet basis is that certain functions are well compressed in such a basis. By well compressed, we mean that few basis functions are needed to represent the function with a small error. Examples of suitable functions are those that are mostly smooth, with small areas of sharp variation. For hyperbolic PDEs this could be shocks in an otherwise smooth fluid flow, as mentioned above, or in acoustics a low frequency wave with localized high frequency bursts superimposed.

Several different approaches for solving hyperbolic PDEs using wavelets have been considered. Jameson [13, 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. Bihari and Harten [4] have used wavelets to localize where to apply ENO methods on a uniform grid. Hierarchical grids in combination with wavelets have been used by Vasilyev and Paolucci [22, 21], where the computation of certain matrix operators is needed, and Fröhlich and Schneider [10] in combination with noncompactly supported interpolating scaling functions. A method where all operations are performed in the wavelet basis was examined by Bacry, Mallat and Papanicolaou with special attention to time adaptability. Such a wavelet method was also considered by the author and Waldén [12], where it was noted (as has been noted by others) that nonlinear operators such as multiplication are too computationally expensive when done directly in a wavelet basis. Keiser [15] has used coiflets to get approximations of point values in a wavelet method, thus simplifying the treatment of nonlinearities. The interpolating wavelet transform has been used in a Galerkin method for solving elliptic problems on the interval by Bertoluzza, Naldi and Ravel [2].

Our motivation for using a wavelet representation is, as noted above, that certain functions have a sparse representation in a wavelet basis. Assume that a function is represented by N points on a uniform grid, and the same function is represented, with an error ϵ, by N_s wavelet coefficients, where $N_s \ll N$. We would like to be able to compute derivatives and multiply functions in this wavelet basis in $\mathcal{O}(N_s)$ time. The interpolating wavelet transform provides the means to achieve this goal. We present a sparse wavelet transform from a sparse wavelet representation to a sparse point representation. The chosen basis has the property that each wavelet coefficient corresponds to a function value at a grid point. This is a key element of the proposed method, since we then have a fast algorithm to move between the physical space, where the computations are made, and the wavelet space, where we choose the points to include in our sparse point representation by thresholding the wavelet coefficients. Thus we take advantage of wavelet's ability to find a good representation, while avoiding the difficulties of differentiating and applying nonlinear operators in the wavelet basis. This is similar to the ideas behind pseudo spectral methods, where one choose the best space to apply an operator and transforms between the representations. One difference is that our method is sparse, while pseudo spectral methods use a full representation. The transform is fast; it takes $\mathcal{O}(N_s)$ time with a small constant. Once we have a sparse point representation, multiplication is almost trivial, and differentiation can be done by centered finite differences, since the sparse wavelet transform provides us with a rule to compute function values at points that are not included in the sparse point representation. Boundary conditions can be treated in the same way as when using a finite difference method on a uniform grid, i.e., by using one-sided stencils on outflow boundaries and prescribing boundary values at inflow boundaries. The use of centered stencils for the differentiation makes generalizations to higher dimensions straight forward and avoids the computation of weights for nonuniform finite difference stencils. The numerical experiments show the method's efficiency when applied to problems whose solutions are well compressed in a wavelet basis.

In Section 2 we describe the interpolating wavelet transform and basis. We then introduce the sparse wavelet and point representation in Section 3, along with operations on these representations and we also describe the time stepping method and examine the stability of the proposed method. In Section 4 and 5 we present the results of numerical experiments in one and two dimensions for linear and nonlinear problems.

2 Background

We will start by presenting the basis and transform that we will use. In Section 2.1 we present the interpolating subdivision scheme by Deslauriers and Dubuc [7, 9]. We then, in Section 2.2, describe the extension to an interpolating wavelet transform that was introduced by Donoho [8]. This is done on the real line. In Section 2.3 we discuss how to treat boundaries and present an interpolating transform adapted to an interval of the real line. Finally, we introduce the multi-dimensional transform in Section 2.4.

2.1 An Interpolating Subdivision Scheme

First we present the interpolating subdivision scheme introduced by Deslauriers and Dubuc [7, 9]. Assume that we have a set of dyadic grids on the real line,

$V_j = \{x_{j,k} \in \mathbb{R} : x_{j,k} = 2^{-j}k, k \in \mathbb{Z}\}$, $j \in \mathbb{Z}$, i.e., the grid points in V_0 are the integers and for V_1 also the half-integers. In general, the dyadic grid V_{j+1} contains all the grid points in V_j, and also additional points inserted halfway in-between each of the points in V_j. The locations of points on such dyadic grids are illustrated in Figure 1.

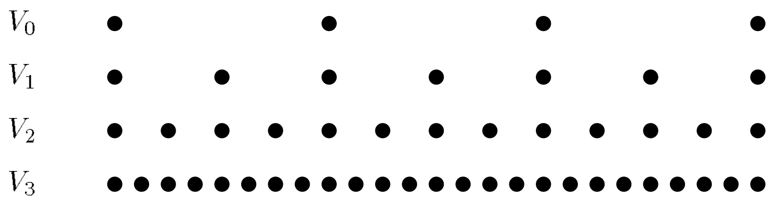


Figure 1: Examples of point positions on dyadic grids.

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 exists in V_j and the corresponding function values are kept unchanged. Values at the odd grid points $x_{j+1,2k+1}$ are computed by polynomial interpolation from the values at the even grid points. We denote this interpolating polynomial by $P_{j+1,2k+1}(x)$. The degree, $p-1$, of this polynomial is odd to make the scheme symmetric, i.e., we interpolate from an even number of function values. Thus we associate one such polynomial to all odd numbered grid points in V_{j+1}. Formally, we define one step in the subdivision scheme as

$$\left\{ \begin{array}{lcl} f_{j+1,2k} & = & f_{j,k}, \\ f_{j+1,2k+1} & = & P_{j+1,2k+1}(x_{j+1,2k+1}), \end{array} \right. \quad \forall k \in \mathbb{Z},$$

where $P_{j+1,2k+1}(x)$ is chosen such that $P_{j+1,2k+1}(x_{j,k+l}) = f_{j,k+l}$ for $-p/2 < l \leq p/2$. So we use p symmetric points on the coarser grid V_j to interpolate one new function value on the finer grid V_{j+1}. An example of this interpolation is shown in Figure 2, using a cubic polynomial ($p = 4$).

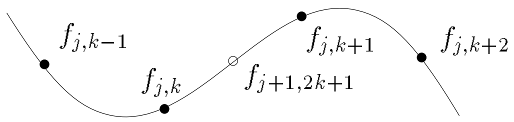


Figure 2: Generating a function value at an even numbered grid point on V_{j+1} by cubic interpolation from V_j.

This is a general method that, in the one-dimensional case, also can be used on nonuniform grids. The evaluation of the interpolating polynomial can then be done by, e.g. Neville's algorithm [19]. Since we are working on dyadic grids, we can explicitly derive the interpolating polynomial, and compute the values at odd grid points from values on the coarser grid directly, by

$$f_{j+1,2k+1} = (f_{j,k} + f_{j,k+1})/2, \quad \text{if} \quad p = 2, \text{ and}$$

$$f_{j+1,2k+1} = \left(-f_{j,k-1} + 9f_{j,k} + 9f_{j,k+1} - f_{j,k+2}\right)/16, \quad \text{if} \quad p = 4.$$

In general we have the relation

$$f_{j+1,2k+1} = \sum_{l=-p/2+1}^{p/2} h_l f_{j,k+l} = \sum_{l=1}^{p/2} h_l (f_{j,k+l} + f_{j,k-l+1}),$$

where the second equality follows from the symmetry of the filter coefficients, h_l. This results in, at most, $3p/2 - 1$ flops (fundamental floating point operations) for the computation of each new odd coefficient. For general p the filter coefficients are

$$h_l = (-1)^{(p/2+l-1)} \frac{\Pi_{i=0}^{p-1}(i - p/2 + 1/2)}{(l-1/2)(p/2+l-1)!(p/2-l)!}, \quad -p/2 < l \leq p/2.$$

Repeating the above described subdivision recursively we get representations on successively finer grids V_j as j increases, and in the limit $j \to \infty$, we have a representation on all dyadic rationals. What are the properties of this limit function?

If we start the subdivision with the sequence $\{\delta_{0,k}\}_{k\in\mathbb{Z}}$ (where $\delta_{j,k} = 1$ if $j = k$ and zero otherwise) on V_0 and refine to V_l we will in the limit $l \to \infty$ get what we will call the scaling function $\varphi(x)$ (called the fundamental function by Deslauriers and Dubuc). From the construction follows that $\varphi(x)$ has compact support on $[-p+1, p-1]$ and is symmetric around $x = 0$. We can also note that φ is cardinal, in the sense $\varphi(k) = \delta_{0,k}, k \in \mathbb{Z}$. If we take one step in the subdivision scheme for the sequence $\{\delta_{0,k}\}_{k\in\mathbb{Z}}$ we can verify the two-scale relation

$$\varphi(x) = \sum_{k=-p+1}^{p-1} g_k \varphi(2x-k) = \sum_{k=-p+1}^{p-1} \varphi(k/2)\varphi(2x-k), \tag{1}$$

where the coefficients $g_k = \varphi(k/2)$, $k = -p+1, \ldots, p-1$, are defined from the filter coefficients of the subdivision scheme by

$$g_k = \begin{cases} 1 & k = 0, \\ 0 & k \neq 0,\ k \text{ even}, \\ h_{(k+1)/2} & k \neq 0,\ k \text{ odd}. \end{cases}$$

Using integer translates of $\varphi(x)$ we have a basis in V_0, and we define the interpolant of any continuous $f(x)$ in V_0 as

$$\mathcal{P}_0 f(x) = \sum_k f_{0,k}\varphi(x-k).$$

If we denote translations and dilations of $\varphi(x)$ by $\varphi_{j,k}(x) = \varphi(2^j x - k)$ we can define the interpolant of $f(x)$ in any V_j by

$$\mathcal{P}_j f(x) = \sum_k f_{j,k}\varphi_{j,k}(x),$$

where $\{\varphi_{j,k}\}_{k\in\mathbb{Z}}$ is a basis in V_j. Note that we here use V_j to denote function spaces, while we earlier used it to denote grids. Since the basis functions are cardinal functions, $\varphi_{j,k}(x_{j,l}) = \delta_{k,l},\ j,k,l \in \mathbb{Z}$, we have a one-to-one correspondence between grid points and basis functions and therefore we will continue with this slight abuse of notation.

Corresponding to (1) we have the general two-scale relation

$$\varphi_{j,k}(x) = \sum_{l=-p+1}^{p-1} g_l \varphi_{j+1,l}(x),$$

so a basis function in V_j is a linear combination of basis functions in V_{j+1}.

We also have the property of polynomial reproduction, following from the polynomial interpolation in the subdivision scheme. If the original function values $f_{j,k}$ are from a polynomial of degree l, the interpolant in any V_j will be exactly this polynomial, if $l < p$. That is,

$$\mathcal{P}_j x^l = \sum_k (x_{j,k})^l \varphi_{j,k}(x) = x^l, \quad \text{for } l = 0, 1, \ldots, p-1.$$

We then say that φ is of order p. In Figure 3 we have plots of $\varphi(x)$ for different values of p. For $p = 2$ we recognize the piecewise linear hat function. Note that for $p > 2$ the basis functions are *not* piecewise polynomial.

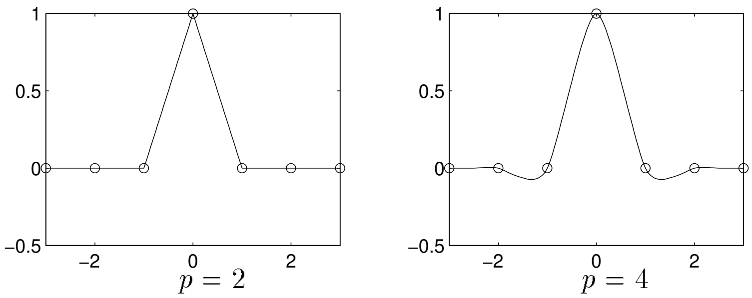


Figure 3: The scaling function, $\varphi(x)$, for two different values of p with the interpolation points indicated by circles.

Interestingly, there is a connection between interpolating subdivision and orthogonal wavelet bases. The scaling function in the interpolating subdivision scheme, $\varphi(x)$ is also the autocorrelation function of the compactly supported Daubechies wavelet, as shown by Beylkin and Saito [3]. That is, if φ is of order p then $\varphi(x) = \int \tilde{\varphi}(y)\tilde{\varphi}(y-x)dy$, where $\tilde{\varphi}$ is the scaling function corresponding to Daubechies wavelets with $p/2$ vanishing moments.

2.2 An Interpolating Wavelet Transform

The scaling function spaces introduced in the previous section generated a ladder of spaces

$$\cdots \subset V_{j-1} \subset V_j \subset V_{j+1} \subset \cdots$$

and the interpolating subdivision scheme enabled us to move to the right in this ladder (refinement). If we want to move to the left (coarsening), without losing information, we can introduce additional spaces W_j that encode the difference between V_j and V_{j+1}, i.e., $V_{j+1} = V_j \oplus W_j$, where $\oplus$ is the nonorthogonal direct sum. In the previous section we saw that we could represent a function in V_{j+1} as

$$\mathcal{P}_{j+1} f(x) = \sum_k f_{j+1,k} \varphi_{j+1,k}(x).$$

Introducing a basis $\{\psi_{j,k}\}_{k \in \mathbb{Z}}$ of W_j, we can write

$$\mathcal{P}_{j+1} f(x) - \mathcal{P}_j f(x) = \sum_k d_{j,k} \psi_{j,k}(x),$$

where $\psi_{j,k}(x) = \psi(2^j x - k)$. The function $\psi(x)$ is our wavelet and we call the $d_{j,k}$s wavelet coefficients. It turns out that one possible choice of wavelet is a translation and dilation of the scaling function itself. That is, we define $\psi(x) = \varphi(2x - 1) = \varphi_{1,1}(x)$. This is, strictly speaking, not a wavelet since usually one requires that a wavelet has at least one vanishing moment, i.e., $\int \psi(x) dx = 0$. Nevertheless we will use the terms wavelet and wavelet coefficient, since the rest of the theory fits in the framework of wavelet methods. This wavelet, corresponding to the interpolating subdivision scheme, was introduced by Donoho [8]. What we have is a nonorthogonal multiresolution analysis and we can now choose between two different representations of a function in V_{j+1},

$$\sum_k f_{j+1,k} \varphi_{j+1,k}(x) \quad \text{or} \quad \sum_k f_{j,k} \varphi_{j,k}(x) + \sum_k d_{j,k} \psi_{j,k}(x).$$

Given a representation of a function in the $\varphi_{j+1,k}$ basis, we thus can decompose it into a coarser scale representation in $\varphi_{j,k}$ and a correction in $\psi_{j,k}$. Starting with a representation on a fine grid V_J we can repeat this decomposition $J - j_0$ times and get

$$\sum_k f_{J,k} \varphi_{J,k}(x) = \sum_k f_{j_0,k} \varphi_{j_0,k}(x) + \sum_{j_0 \le j < J} \sum_k d_{j,k} \psi_{j,k}(x).$$

On the left hand side we have a scaling function representation on a fine grid, and on the right we have decomposed it into a scaling function representation on a coarse grid V_{j_0} and wavelets on successively finer scales. We call this a wavelet representation. As for the scaling function representation, we have a one-to-one correspondence between wavelet coefficients and point values. In Figure 4 we have an illustration of such a wavelet representation for $J = 3$ and $j_0 = 0$.

How do we compute the wavelet coefficients, $d_{j,k}$? First we consider one step of the transform. We have all the function values $f_{j+1,k}$ at the grid points in V_{j+1}. We do the same interpolation as in the previous section, and then set each $d_{j,k}$ to the difference between the known function value $f_{j+1,2k+1}$ and the value predicted by the interpolating polynomial,

$$d_{j,k} = f_{j+1,2k+1} - P_{j+1,2k+1}(x_{j+1,2k+1}), \quad \forall k \in \mathbb{Z}.$$

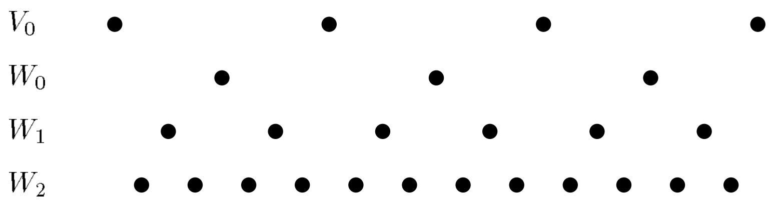


Figure 4: Example of a wavelet representation where each point corresponds to the location of a scaling function.

That is, the wavelet coefficients encode the error when interpolating the function values from a coarser scale. This is one step of the transform. The interpolation and computation of $d_{j,k}$ is illustrated in Figure 5.

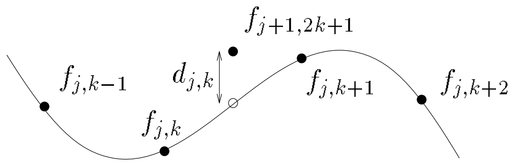


Figure 5: Generating a wavelet coefficient $d_{j,k}$ in W_j by cubic interpolation from function values on V_j.

Repeating this recursively for $j = J-1, J-2, \ldots, j_0$ we have an algorithm for computing the wavelet representation from the scaling function representation. 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,

$$f_{j+1,2k+1} = d_{j,k} + P_{j+1,2k+1}(x_{j+1,2k+1}), \quad \forall k \in \mathbb{Z}, \quad j = j_0, j_0+1, \ldots, J-1.$$

This transform is a special case (the wavelet coefficients are independent of each other) of the well-known fast wavelet transform.

We can now see some advantages of our choice of wavelet and scaling function. First of all the whole transform can be done in situ. Given an array of function values we simply replace every other function value by the computed wavelet coefficient. This is then repeated recursively for the remaining function values to generate the complete wavelet representation. Secondly, if we examine the possibilities for parallelizing the transform, we note that the computation of each wavelet coefficient is completely independent of all the other wavelet coefficients. Thus we can do the forward transform in $\mathcal{O}(1)$ time, given enough processors! For the inverse transform, a point value can only be computed when the point values at a coarser scale is available. So we can only parallelize each level of the transform, leading to $\mathcal{O}(\log_2 N)$ parallel execution time. We will see other advantages of our basis choice when introducing the sparse wavelet transform in Section 3.2.

2.3 Boundaries

Several ways of adapting wavelet bases to boundaries have been presented, e.g. by Cohen and Daubechies [5] and by Andersson et al. [1]. An advantage of interpolating wavelets is that the adaption to an interval is straightforward due to the usage of interpolating polynomials in the construction. We choose to present the theory for the unit interval but this choice is arbitrary and easily changed (at least to another interval with dyadic endpoints). First we define a set of dyadic grids on the unit interval, $V_j = \{x_{j,k} = 2^{-j}k : 0 \leq x_{j,k} \leq 1, k \in \mathbb{Z}\}$, $j \in \mathbb{Z}$. Then we have 2^j+1 points on each grid. We assume that $2^j+1 \geq p$, since we need p points for each interpolation. In the interior of the grids we can clearly use the same interpolating wavelet transform as on the real line. Close to the boundaries we have to modify the algorithm. If any of the points that define the interpolating polynomial is not available because it lies outside the boundary we simply use the closest p points that lie inside the interval on the coarser grid; thus we use the most symmetric grid points available on the coarser grid to define the interpolating polynomial. This is the *only* modification needed to adapt the interpolating wavelet transform to an interval. The refinement relation is

$$f_{j+1,2k+1} = \sum_l h_{k,l} f_{j,k+l}.$$

Now the filter coefficients, $h_{k,l}$, depend on the location, k, since we have to make the above described adjustments at boundaries. In Table 1 we present the coefficients for $p = 4$ (no boundary adjustment is needed when $p = 2$). The

Table 1: Filter coefficients, $h_{k,l}$, for the refinement relation at the left boundary when $p = 4$. The weights are in reversed order at the right boundary.

l	-1	0	1	2	3
$k = 0$		5/16	15/16	-5/16	1/16
$k > 0$	-1/16	9/16	9/16	-1/16	

weights are in reversed order at the right boundary.

In Figure 6 we have plots of the four leftmost boundary scaling functions in V_3 when $p = 4$. We note that the fourth function, $\varphi_{3,3}$, is not the basis function on the interior of the interval, but is modified due to the use of one-sided interpolation near the boundary. In Appendix A, MATLAB code for the forward and inverse interpolating wavelet transform on the unit interval is provided.

A more detailed presentation of interpolating subdivision, the interpolating wavelet transform, and the related lifting scheme can be found in a paper by Sweldens and Schröder [20].

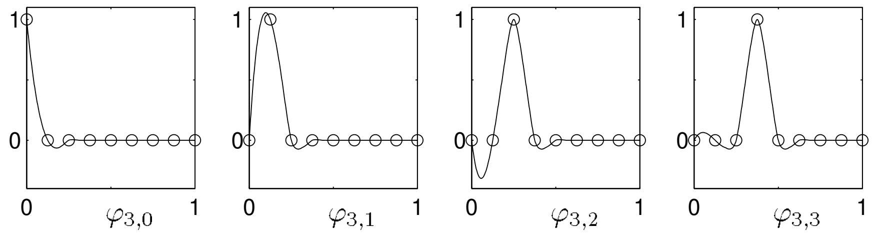
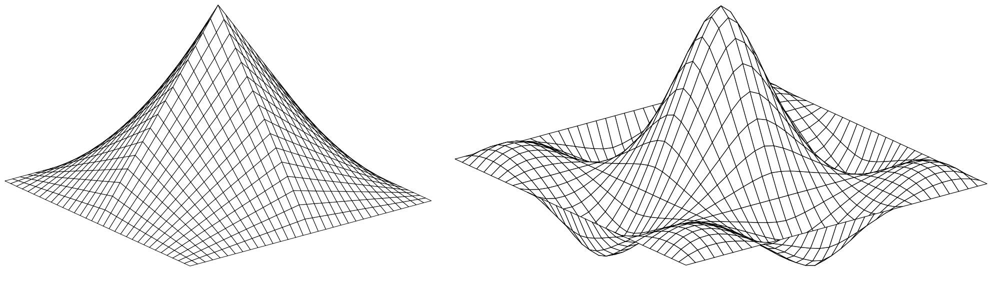


Figure 6: The left boundary scaling functions in V_3 when $p = 4$ with the interpolation points indicated by circles.



(a) $p = 2$ (b) $p = 4$

Figure 7: The two-dimensional scaling function $\Phi(x, y)$.

2.4 A Multi-Dimensional Transform

As is done by Daubechies [6], we construct multi-dimensional bases by tensor products of the V_j spaces. In the two-dimensional case we define spaces $\mathbf{V}_j, j \in \mathbb{Z}$, by

$$\mathbf{V}_j = V_j \otimes V_j = \overline{\operatorname{Span}\{f(x)g(y); f, g \in V_j\}}.$$

Then, we again have a ladder of spaces

$$\cdots \subset \mathbf{V}_{j-1} \subset \mathbf{V}_j \subset \mathbf{V}_{j+1} \subset \cdots,$$

with the corresponding detail spaces $\mathbf{W}_j$. Now the basis functions in $\mathbf{V}_0$ are $\Phi(x, y) = \varphi(x)\varphi(y)$ and all its $\mathbb{Z}^2$-translates. Plots of $\Phi(x, y)$ when $p = 2$ and $p = 4$ are shown in Figure 7. We now have three wavelets, instead of one; $\Psi^1(x, y) = \Phi(2x-1, 2y)$, $\Psi^2(x, y) = \Phi(2x, 2y-1)$ and $\Psi^3(x, y) = \Phi(2x-1, 2y-1)$ whose $\mathbb{Z}^2$-translates generate a basis in $\mathbf{W}_0$. Again, every basis function can be identified with a point, and the location of these points in $\mathbf{V}_1$ are shown in Figure 8. The implementation of the two-dimensional transform on the unit square is simply the one-dimensional transform applied first to rows, and then to columns of the grids. More precise, in analogy with the one-dimensional case, let us define dyadic grids $\mathbf{V}_j = \{(x_{j,k}, y_{j,l}) \in \mathbb{R}^2 : x_{j,k} = 2^{-j}k, y_{j,l} = 2^{-j}l\}$, $j \in \mathbb{Z}$.

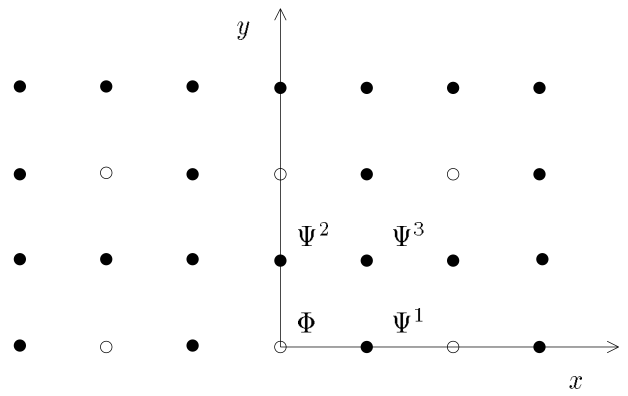


Figure 8: The location of scaling functions and wavelets in $\mathbf{V}_1$.

We then define function values on $\mathbf{V}_j$ as $\{f_{j,k,l}\}_{k,l\in\mathbb{Z}}$, where $f_{j,k,l} = f(x_{j,k}, y_{j,l})$. Given function values in $\mathbf{V}_{j+1}$ we have a representation in $\mathbf{V}_j \oplus \mathbf{W}_j$ by the relation

$$\sum_{k,l} f_{j+1,k,l}\Phi_{j+1,k,l}(x,y) = \sum_{k,l} f_{j,k,l}\Phi_{j,k,l}(x,y) + \sum_{m=1}^{3}\sum_{k,l} d^m_{j,k,l}\Psi^m_{j,k,l}(x,y).$$

How do we compute the wavelet coefficients; $d^m_{j,k,l}$, $m = 1, 2, 3$; given the function values $f_{j+1,k,l}$, shown in Figure 9(a)? As mentioned above, we make a one-dimensional, one level, transform along rows of the grid (in the x- or k-direction). These interpolations provide us with the wavelet coefficients $d^1_{j,k,l}$ and intermediate values $\tilde{d}^3_{j,k,l}$ as shown in Figure 9(b). We then transform along columns of the grid (in the y- or l-direction). This results in wavelet coefficients $d^2_{j,k,l}$ and $d^3_{j,k,l}$ and completes the one-level transform as shown in Figure 9(c). The inverse transform is simply the reversal of these steps. If we examine each wavelet coefficients dependence on function values we have the dependencies shown in Figure 10.

The extensions to higher dimensions are obvious. In the three-dimensional case, we have seven wavelets, and in the general n-dimensional case, we have $2^n - 1$ wavelets, that all are translations and dilations of the scaling function.

3 Theory

In the following sections we will describe the different building blocks needed in a method that uses interpolating wavelets for solving PDEs. First of all the sparse wavelet representation of functions is described in Section 3.1. Then, in Section 3.2, we describe a transform to a sparse point representation. In the remaining sections we describe how to perform necessary operations such as multiplication, differentiation and time marching. Finally, we attempt to justify the stability of the proposed method by studying the stability of a simplified problem.

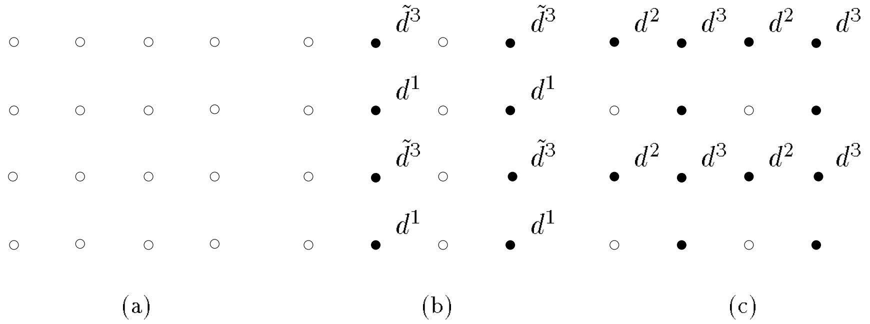


Figure 9: The computational steps of the two-dimensional transform. Unfilled circles denote function values. Filled circles denote wavelet coefficients and intermediate values.

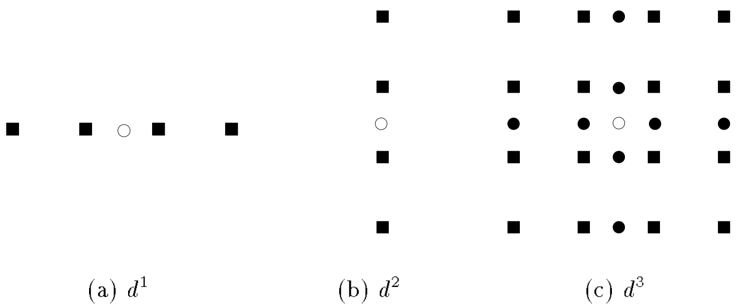


Figure 10: The function values that each wavelet coefficient is computed from when $p = 4$. Unfilled circles denote wavelet coefficients. filled squares denote function values at a coarser grid. Filled circles denote function values at the same grid.

3.1 The Sparse Wavelet Representation

The reason that we are interested in wavelet methods in the first place is the possibility of compressing the representations of functions in a wavelet basis. This is usually done by thresholding of the wavelet coefficients. Assume that we have the wavelet representation

$$\mathcal{P}_J f(x) = \sum_k f_{j_0,k}\varphi_{j_0,k}(x) + \sum_{j_0 \le j < J} \sum_k d_{j,k}\psi_{j,k}(x).$$

To get a sparse representation we remove all wavelet coefficients with magnitude smaller than some threshold value ϵ. Then, we have the thresholded expansion

$$\mathcal{P}_J^{\epsilon} f(x) = \sum_k f_{j_0,k}\varphi_{j_0,k}(x) + \sum_{(j,k)\in I(\epsilon)} d_{j,k}\psi_{j,k}(x).$$

We call this the sparse wavelet representation of the function f. A coefficient that is retained after thresholding is called significant. The set $I(\epsilon)$ of significant coefficients is defined as

$$I(\epsilon) = \{(j,k) \;:\; j_0 \le j < J,\ |d_{j,k}| > \epsilon\}.$$

Remember that the wavelet coefficients are measures of the error (in maximum norm) when interpolating point values on finer scales from coarser. So in the cubic case the wavelet coefficients measure failure of the function to be cubic, locally. Loosely speaking, after thresholding the function by ϵ, we have a representation of the function that is locally cubic, and approximates the original function, before thresholding, with an error ϵ. When we have an expansion on a bounded domain, we denote the number of significant coefficients and scaling function coefficients by N_s, and the total number of coefficients, before thresholding, by N. In Figure 11 an example of a sparse wavelet representation is shown. We note that each wavelet coefficient has two neighbors on the next finer level. We call these coefficients the wavelet coefficients children. In the two-dimensional case each wavelet coefficient has eight children.

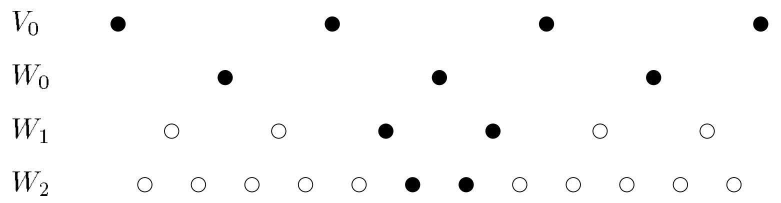


Figure 11: Example of a sparse wavelet representation. Solid dots represent scaling function coefficients and significant wavelet coefficients

Which functions will be well compressed (in the sense $N_s \ll N$) in the interpolating wavelet basis? It is well-known that for orthogonal wavelet bases the coefficients will have a fast decay over the scales if the function is smooth [6], thus smooth functions will be well represented by few wavelet coefficients. Fortunately, the same is true for the interpolating wavelet basis, as shown by Donoho [8].

3.2 The Sparse Point Representation—SPR

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 inverse full 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 the sparse wavelet representation

$$\sum_k f_{j_0,k}\varphi_{j_0,k}(x) + \sum_{(j,k)\in I(\epsilon)} d_{j,k}\psi_{j,k}(x).$$

We do the inverse transform, but *only* for those points that correspond to the significant wavelet coefficients in $I(\epsilon)$. If any point value is needed that do not exist, we interpolate the value from a coarser scale, recursively. The algorithm will terminate since we have all values on the coarsest grid, V_{j_0}. We call this the inverse sparse wavelet transform that leads to a sparse point representation (SPR). The SPR is the set of function values $\{f_{j,k}\}_{(j,k)\in J(\epsilon)}$ where

$$J(\epsilon) = \{(j_0,k)\ :\ k\in\mathbb{Z}\}\cup\{(j+1,2k+1)\ :\ (j,k)\in I(\epsilon)\}.$$

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 require $\mathcal{O}(N_s)$ flop, 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 only computed to decide if the corresponding point value is to be included in the SPR or not.

3.3 Multiplication

Multiplication of two functions can be done directly in the basis of the sparse wavelet representation but will be costly, due to interactions between scales in the wavelet representation. Roughly, each wavelet coefficient in the product will depend on all wavelet coefficients in the operands whose basis functions overlap the wavelet coefficients basis function. The same reasoning holds true for other nonlinear operations. The reason being the wavelet basis' hierarchical structure, where many fine scale basis functions overlap each coarse scale basis function. Since multiplication is a local operation it seems natural to work with point values. As an example, given the SPR of a function, taking the square is trivial. Each function value in the SPR is simply squared. If the SPR is $\{f_{j,k}\}_{(j,k)\in J(\epsilon)}$, we compute

$$g_{j,k} = f_{j,k}^2, \qquad \forall (j,k)\in J(\epsilon).$$

Our approximation of the square is then the SPR $\{g_{j,k}\}_{(j,k)\in J(\epsilon)}$. Multiplying two functions with different sparsity pattern demands a little more thought. To make the multiplication in $\mathcal{O}(N_s)$ time we need to specify the SPR that we want to represent the result in. The SPR of the product can for example be chosen as that of one of the operands or as the union of the two operands representations. Assume that the operands are $\{u_{j,k}\}_{(j,k)\in J_u(\epsilon)}$ and $\{v_{j,k}\}_{(j,k)\in J_v(\epsilon)}$. If we want the result represented as the union of the two SPRs, we compute

$$g_{j,k} = u_{j,k}v_{j,k}, \qquad \forall (j,k) \in J(\epsilon) = J_u(\epsilon) \cup J_v(\epsilon).$$

If a point value is missing, it is again interpolated from a coarser scale in the SPR. Our approximation of the product is then the SPR $\{g_{j,k}\}_{(j,k)\in J(\epsilon)}$.

3.4 Differentiation

Differentiation can also be done directly in the sparse wavelet representation, but will be costly for the same reasons mentioned for multiplication. Instead we propose to use a finite difference approach in the SPR. The reason being that, as noticed earlier, our SPR is locally polynomial with an error ϵ. So we use a finite difference stencil of order p (the same order as that of the representation). For each point that we want to approximate the derivative in, 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. The finite difference first derivative approximation is

$$f'(x) \approx \frac{1}{h}\sum_l g'_l f(x+lh)$$

and the second derivative approximation is

$$f''(x) \approx \frac{1}{h^2}\sum_l g''_l f(x+lh).$$

On an interval the filter coefficients, g'_l and g''_l, depend on x since we use one-sided approximations near the boundaries. In Table 2 the filter coefficients for the first derivative approximation at the left boundary are shown when $p = 2$ and $p = 4$. The coefficients at the right boundary are reversed in order and of opposite sign. In Table 3 the filter coefficients for the second derivative approximation at the left boundary are shown when $p = 2$ and $p = 4$. The coefficients at the right boundary are reversed in order. In two dimensions the partial derivatives in each direction is evaluated using the above described one-dimensional approximations. The step length, h, is chosen as the distance to the closest point in the SPR as measured along any of the coordinate directions.

Table 2: Filter coefficients, g'_l, for the first derivative approximation at the left boundary. The coefficients at the right boundary are reversed in order and of opposite sign.

	$l =$	-2	-1	0	1	2	3	4
$p = 2$	$0 \leq x < h$			-3/2	2	-1/2		
	$x \geq h$		-1/2	0	1/2			
$p = 4$	$0 \leq x < h$			-25/12	4	-3	4/3	-1/4
	$h \leq x < 2h$		-1/4	-5/6	3/2	-1/2	1/12	
	$x \geq 2h$	1/12	-2/3	0	2/3	-1/12		

Table 3: Filter coefficients, g''_l, for the second derivative approximation at the left boundary. The coefficients at the right boundary are reversed in order.

	$l =$	-2	-1	0	1	2	3	4	5
$p = 2$	$0 \leq x < h$			2	-5	4	-1		
	$x \geq h$		1	-2	1				
$p = 4$	$0 \leq x < h$			15/4	-77/6	107/6	-13	61/12	-5/6
	$h \leq x < 2h$		5/6	-5/4	-1/3	7/6	-1/2	1/12	
	$x \geq 2h$	-1/12	4/3	-5/2	4/3	-1/12			

3.5 Time Evolution in the SPR

We want to solve the initial boundary value problem

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

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, $J(\epsilon)$, 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 perform a sparse wavelet transform, threshold the sparse wavelet representation and invert the transform. To allow the basis to change we have to extend the sparse wavelet representation after the thresholding. First in space, we add some neighbor coefficients to all wavelet coefficients. How many depend on the PDEs wave propagation speed. Then the basis is able to adjust when the solution is moving. Secondly in frequency, we add two children to all coefficients larger in magnitude than some constant δ. In the two-dimensional case we add eight children. The choice $\delta = c\epsilon$, where c is a constant, has worked well in the numerical experiments. If $c < 1$ we refine (add children to) all wavelet coefficients. This refinement allow 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 the solution. We could use the union of the components SPRs, but maybe more interesting would be to let each component have 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.

To more precisely describe the SPR method, let us define the operators

- T_ϵ for the thresholding of the wavelet coefficients, i.e., the removal of points in the SPR whose corresponding wavelet coefficient has magnitude less than ϵ,
- E for the extension of the SPR, both in space and frequency,
- D for the approximation of the continuous operator $\mathcal{P}$, and
- P, a projection operator that enforce the boundary conditions [11].

We denote by v the SPR of the approximate solution. The SPR method is then defined by the following algorithm.

1. $v \leftarrow ET_\epsilon u_0, \qquad t \leftarrow 0.$

2. The system of ODEs
$$\frac{dv}{dt} = Dv$$
is advanced to time $t \leftarrow t + \Delta t$, where Δt is related to $h_{\min}$ by a CFL condition.

3. The boundary conditions are enforced by the projection operator,
$$v \leftarrow Pv.$$

4. We extend the SPR,
$$v \leftarrow Ev.$$

5. *Goto* 2.

3.6 Stability

When the threshold, $\epsilon \to 0$ the proposed method goes to an ordinary finite difference method on a uniform grid. For a finite value of ϵ it is difficult to do a stability analysis due to the nonlinear nature of the thresholding. Probably the crucial part of the method, from a stability point of view, is the interpolation of points from coarser grids when using the finite difference method described in Section 3.4. Because of that, we examine the stability of the following model problem
$$\begin{cases} u_t = au_x, \quad u = u(x,t), \quad t > 0, \quad x \in \mathbb{R}, \\ u(x,0) = u_0(x). \end{cases}$$
Discretizing this problem in space by second order centered finite differences on a nonuniform grid with step-size h for $x < 0$ and $2h$ for $x > 0$ we have the semi-discrete problem

$$\begin{cases} \dfrac{dw_j}{dt} = aD_0(h)w_j, \quad j < 0, \\ \dfrac{dw_0}{dt} = \dfrac{a}{2h}\left(\dfrac{1}{2}(w_0 + w_1) - w_{-1}\right) = a\dfrac{w_0 + w_1 - 2w_{-1}}{4h}, \\ \dfrac{dw_j}{dt} = aD_0(2h)w_j, \quad j > 0, \\ w_j(0) = f_j. \end{cases} \tag{2}$$

Where $D_0(h)w_j = (w_{j+1} - w_{j-1})/2h$ and we have used linear interpolation to compute the function value at the missing point $x = h$. This corresponds to using a hat-function as a scaling function ($p = 2$). The grid is shown in Figure 12. The grid can be thought of as arising in our interpolation method, on two grids, when all the wavelet coefficients for $x > 0$ have been thresholded on the finer grid. To simplify the stability analysis, we rewrite this scalar

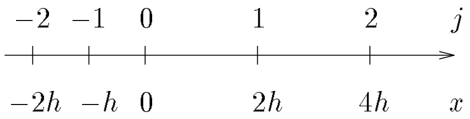


Figure 12: The nonuniform grid.

Cauchy problem as a system with two components and a boundary by folding the x-axis at $x = 0$ and introducing the new variables u_j and v_j,

$$\begin{cases} u_j = w_j, & j > 0, \\ v_{-j} = w_j, & j < 0, \\ u_0 = v_0 = w_0, & j = 0. \end{cases}$$

Substituting into (2) we have the quarter-space problem

$$\begin{cases} \dfrac{d}{dt}\begin{pmatrix} u_j \\ v_j \end{pmatrix} = \begin{pmatrix} a/2 & 0 \\ 0 & -a \end{pmatrix} D_0 \begin{pmatrix} u_j \\ v_j \end{pmatrix}, \quad j > 0, \\ u_0 = v_0, \\ \dfrac{du_0}{dt} = a\dfrac{u_0 + u_1 - 2v_1}{4h}, \\ \begin{pmatrix} u_j(0) \\ v_j(0) \end{pmatrix} = \begin{pmatrix} f_j^I \\ f_j^{II} \end{pmatrix}. \end{cases} \tag{3}$$

This is a semi-discrete system of two scalar equations, coupled by the boundary condition. Using standard methods for stability analysis of finite difference schemes [11], one can then prove that the approximation is stable. This is shown by Olsson and Petersson [17] in the context of overlapping grid methods.

4 One-Dimensional Numerical Experiments

We now turn to some numerical examples in one dimension. In Section 4.1 we study some properties of the sparse point representation (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 4.2. In Section 4.3 we then solve a nonlinear problem; Burgers' equation.

4.1 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 13 along with its square and derivative.

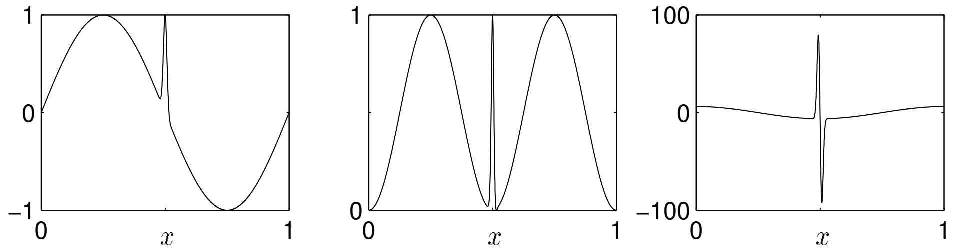


Figure 13: To the left the test function, $f(x) = \sin(2\pi x) + \exp(-\alpha(x-1/2)^2)$ when $\alpha = 10^4$. In the middle the square of the test function, and to the right the derivative of the test function.

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

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

Let us define define the interpolation $\mathcal{P}^\epsilon f = \mathcal{P}_j^\epsilon f$, where j is chosen large enough such that $I(\epsilon)$ 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 [8] has shown that, for smooth enough f,

$$|f - \mathcal{P}^\epsilon f|_\infty \le c_1 \epsilon, \tag{4}$$

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

$$N_s \le c_2 \epsilon^{-1/p} \tag{5}$$

or equivalently

$$\epsilon \le c_2^p N_s^{-p}. \tag{6}$$

Combining (4) and (5) we have a bound on the error in terms of N_s,

$$|f - \mathcal{P}^\epsilon f|_\infty \le c_3 N_s^{-p}. \tag{7}$$

This means that our sparse interpolating wavelet representation is of order p *in the number of significant coefficients*, N_s. We will verify these relations numerically for our test function. In Figure 14 we have an illustration of the test functions sparse wavelet expansion when $\epsilon = 10^{-2}$ and $p = 4$. As expected, most coefficients in the expansion are used to resolve the peak around $x = 1/2$. This value of ϵ results in $N_s = 33$. The error $f - P^\epsilon f$ when $\epsilon = 10^{-3}$ is displayed in Figure 15, where we note that the error is distributed over the whole unit interval. This distribution of the error is a feature of all thresholded wavelet approximations. Notice that the error is bounded by ϵ on the interval. We can also see that the error is zero at all points that are included in our wavelet

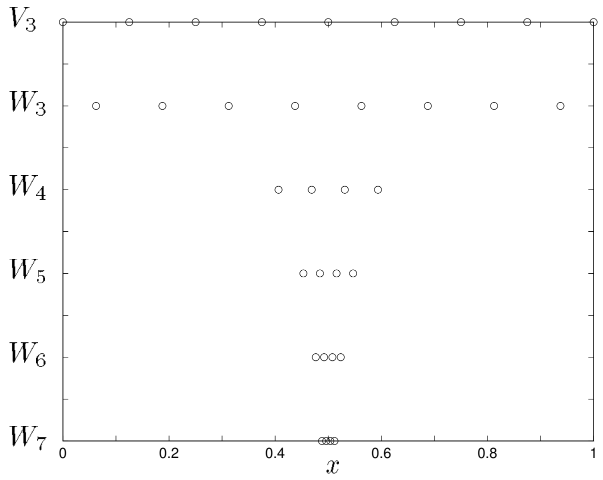


Figure 14: The sparse wavelet representation of the test function when $\alpha = 10^4$, $p = 4$ and $\epsilon = 10^{-2}$. Each circle corresponds to a significant coefficient and $N_s = 33$.

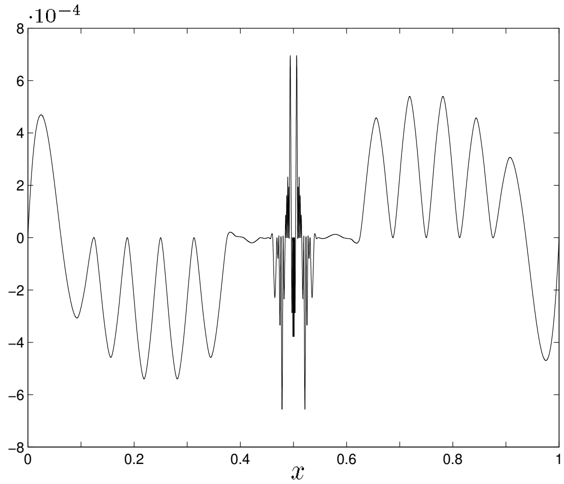


Figure 15: The representation error $f - P^\epsilon f$, when $\alpha = 10^4$, $p = 4$ and $\epsilon = 10^{-3}$.

representation. This follows from the fact that the SPR is exact at the included points. Now we examine the error in maximum norm, $|f - P^{\epsilon} f|_{\infty}$, as a function of the number of significant coefficients, N_s, for linear and cubic interpolation ($p = 2$ and $p = 4$). The result is shown in Figure 16. The slopes of the graphs are -2.1 and -3.8 which is consistent with Equation (7). In Figure 17 we have

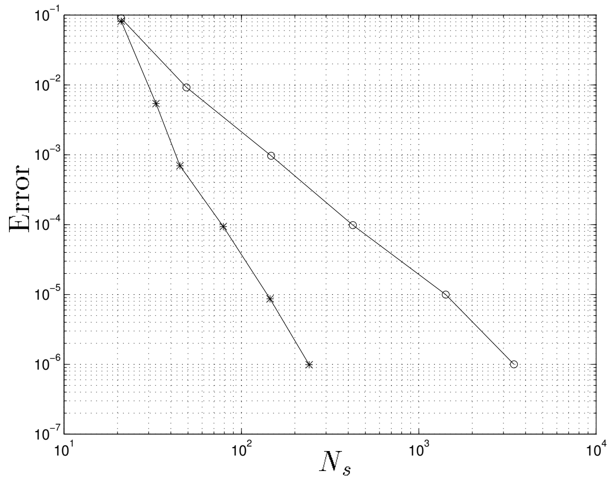


Figure 16: The representation error, $|f - P^{\epsilon} f|_{\infty}$, for the test function as a function of N_s when $\alpha = 10^4$ for $p = 2$ (circles) and $p = 4$ (stars).

plotted N_s as a function of the threshold, ϵ, when $p = 2$ and $p = 4$. According to Equation (5) the dependence should be $N_s \leq c_2 \epsilon^{-1/p}$. Again we measure the slopes and find the consistent values of $-0.47 \approx -(2.1)^{-1}$ and $-0.25 = (4.0)^{-1}$, respectively. When thresholding an SPR we need to interpolate values at all

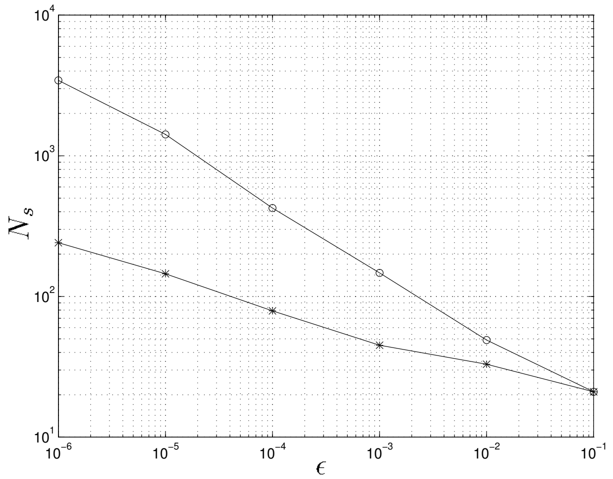


Figure 17: The number of significant coefficients, N_s, as a function of the threshold, ϵ, for the SPR of the test function when $\alpha = 10^4$ for $p = 2$ (circles) and $p = 4$ (stars).

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 that we interpolate from 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. In Figure 18 we show the number of flops per significant coefficient as a function of N_s for interpolating the function values in the SPR of our test function when $p = 2$ and $p = 4$. We never need more than 2 and 6 flops per coefficient—thus thresholding is not expensive, even if done recursively.

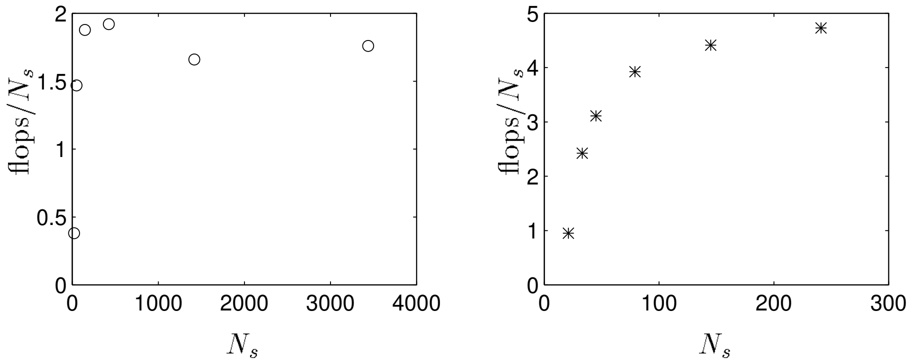


Figure 18: The number of flops per significant coefficient as a function of N_s for thresholding the test functions SPR when $\alpha = 10^4$. To the left for $p = 2$ and to the right for $p = 4$.

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 13. Examining the error of the square approximation we find that, using Equations (4) and (6),

$$|f^2 - (\mathcal{P}^\epsilon f)^2|_\infty \leq |f + \mathcal{P}^\epsilon f|_\infty \cdot |f - \mathcal{P}^\epsilon f|_\infty \leq \tilde{c}_1 \epsilon \leq \bar{c}_1 N_s^{-p}. \qquad (8)$$

If we examine the error for $(\mathcal{P}^\epsilon f)^2$ as a function of N_s when $p = 2$ and $p = 4$, shown in Figure 19, we find that Equation (8) is valid, since the slopes of the graphs are -1.9 and -4.0; thus the approximation of the square of the test function is of order p in the significant coefficients.

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

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

The point-wise error in the derivative approximation will then be

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

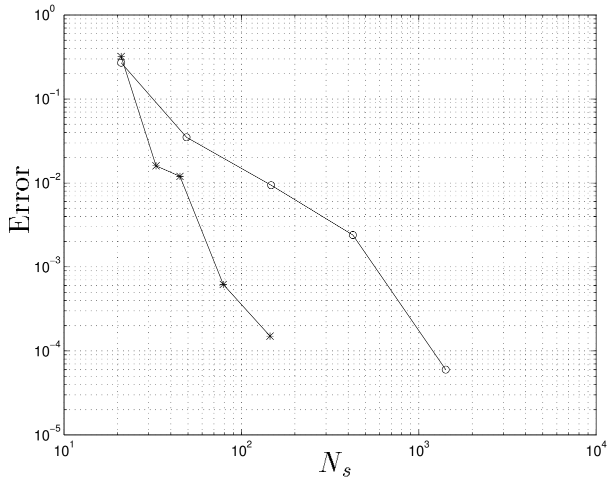


Figure 19: The error $|f^2 - (\mathcal{P}^\epsilon f)^2|_\infty$, for the square of the test function's SPR as a function of N_s when $p = 2$ (circles) and $p = 4$ (stars).

where D_p is the finite difference approximation of the derivative of an SPR of order p, as described in Section 3.4. 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 Equation (6) 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 13. Now we numerically examine the point-wise 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 20. We find that the slopes of the graphs are approximately -1.4 and -3.4. We seem to lose only about half an order of approximation when differentiating the SPR. 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 Figure 21 the actual number of flops per significant coefficient are shown for differentiating the test function. When $p = 2$ 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.

If we examine the point-wise error for a fixed threshold, as shown in Figure 22, we find that the error for the derivative approximation is larger around $x = 1/2$ than in the rest of the interval. We can compare this with the representation error in Figure 15 which was evenly distributed. This is not surprising,

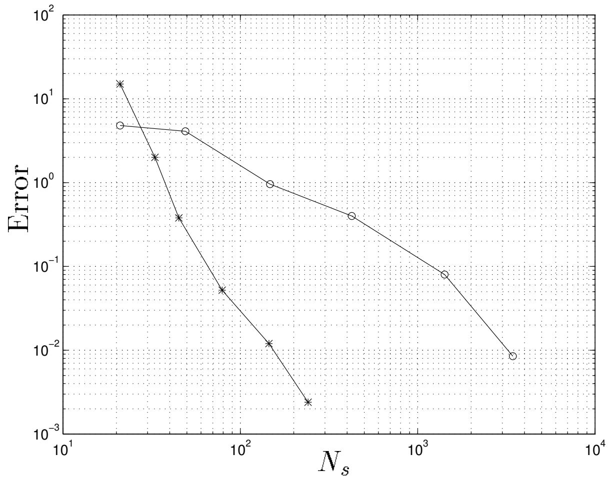


Figure 20: The pointwise error for the approximation of f' as a function of N_s when $p = 2$ (circles) and $p = 4$ (stars) for $\alpha = 10^4$.

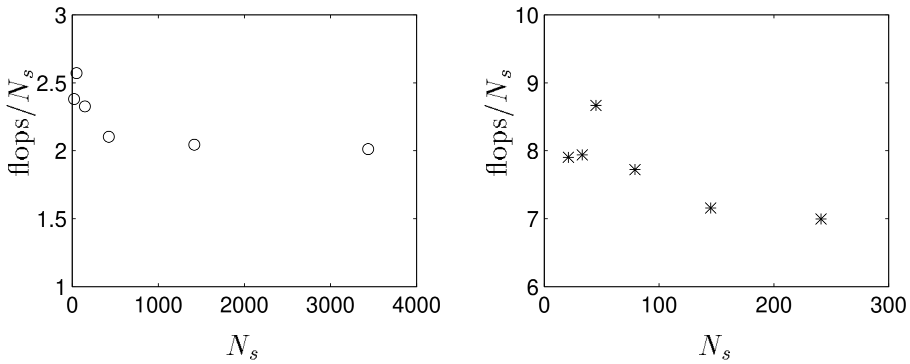


Figure 21: The number of flops per significant coefficient when approximating the derivative of the test function's SPR by finite differences. To the left when $p = 2$ and to the right when $p = 4$ for $\alpha = 10^4$.

since our representation is constructed to minimize the representation error and not to minimize the error when used to represent the derivative.

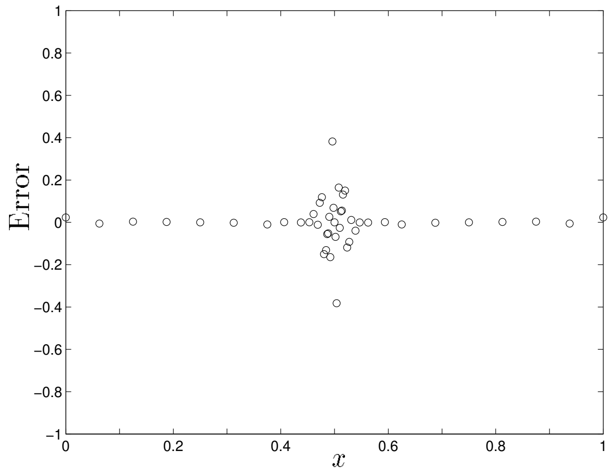


Figure 22: The point-wise error at all points in the SPR of the approximation to the test function's derivative when $p = 4$, $\epsilon = 10^{-3}$ and $\alpha = 10^4$.

4.2 A Linear Problem

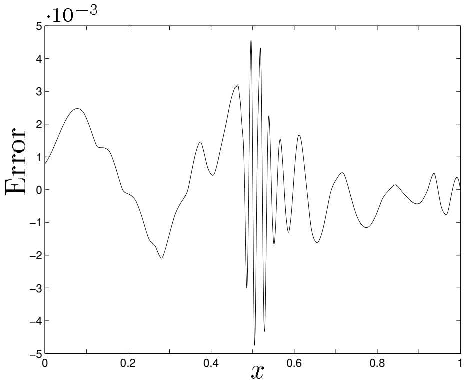
In this section we examine the performance of the proposed SPR method when solving a linear advection equation on the unit interval. Specifically, we will solve

$$\begin{cases} u_t = u_x, & 0 \le x \le 1, \quad t > 0, \\ u(x,0) = u_0(x), & u = u(x,t), \quad u(1,t) = u_0(t). \end{cases} \tag{9}$$

The left boundary is an outflow boundary and the right boundary is an inflow boundary. The solution is a periodic translation of the initial function, $u(x,t) = u_0((x+t) \bmod 1)$. Note that the boundary conditions are chosen so that the solution will be the same as for a problem with periodic boundary conditions. As 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 13. 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 the solution. After space discretization, forming the SPR, we have a system of ODEs to solve. We solve this system using the classical 4th 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. To the left in Figure 23 we



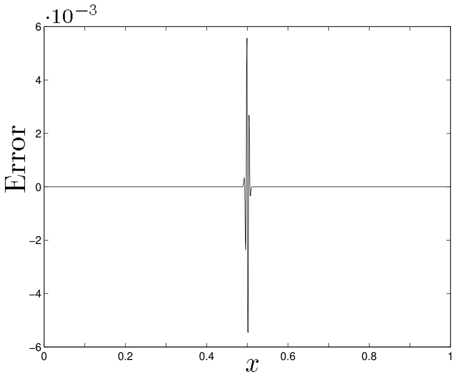


Figure 23: To the left the error at $t = 1$ when $p = 4$ and $\epsilon = 10^{-5}$. To the right the error at $t = 1$ for a 4th order finite difference method with $N = 1025$. The initial function u_0 has $\alpha = 10^4$.

display the error in the solution at $t = 1$ when $p = 4$ and $\epsilon = 10^{-5}$. The error is distributed over the whole interval, although largest around $x = 1/2$. We can compare this with the error at $t = 1$ when using a 4th order finite difference method on a uniform grid of 1025 points, shown in the same figure. This error is concentrated to a small interval around $x = 1/2$. We can interpret this fact as follows: the finite difference method's error is determined by the error in the approximation around the peak, thus the high accuracy in the smooth regions is wasted if all we want to control is the maximum error.

If the SPR method behaves as a finite difference method we should have a linear error growth in time (since the solution to this specific problem does not grow in time). The error in maximum norm when $p = 4$, $\epsilon = 10^{-5}$ and $\alpha = 10^4$ is shown as a function of time in Figure 24. We see a linear growth of the error until the peak is advected out of the domain at $t = 0.5$, then we again have linear growth until $t = 1$.

The sparse wavelet representation at different times is displayed in Figure 25. We see that the refinement follows the peak in the solution. In Figure 26 we have plotted the number of significant coefficients as a function of time. As we can see, N_s is slowly varying around a mean value of approximately 400. That N_s does not increase significantly over time shows that the refinement automatically follow 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 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

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

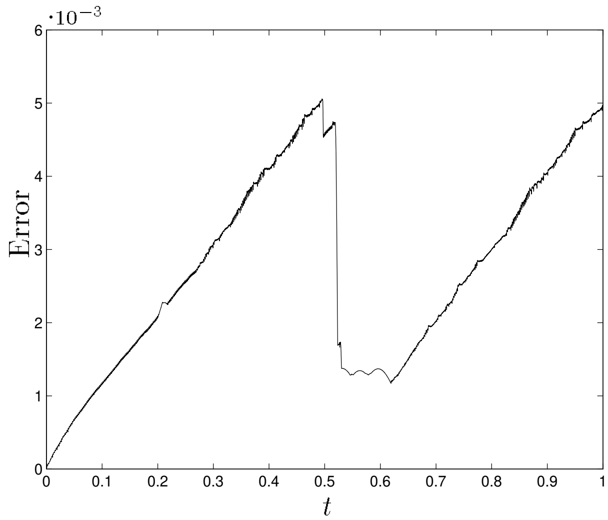


Figure 24: The error in maximum norm when $p = 4$, $\epsilon = 10^{-5}$ and $\alpha = 10^4$, as a function of time.

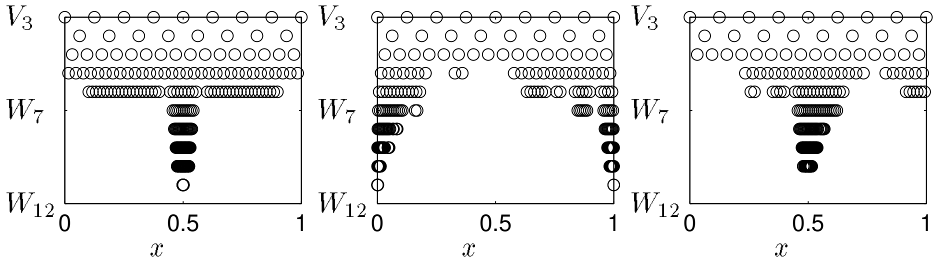


Figure 25: The sparse wavelet representation of the solution; at $t = 0$ to the left, at $t = 0.5$ in the middle, and at $t = 1$ to the right, when $p = 4$, $\epsilon = 10^{-5}$ and $\alpha = 10^4$.

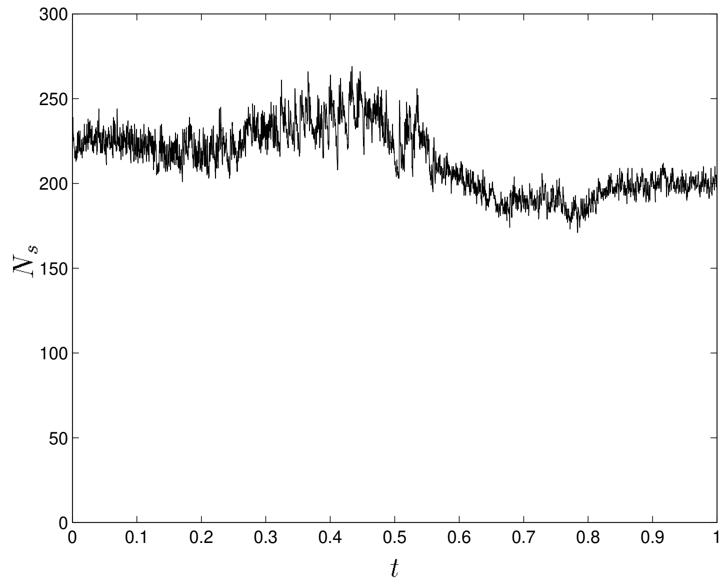


Figure 26: The number of significant coefficients as a function of time when $p = 4$, $\alpha = 10^4$ and $\epsilon = 10^{-5}$.

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

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

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 27 $|e(1)|_{\epsilon,\infty}$ is plotted as a function of N_s for the SPR method when $p = 4$, and for a 4th order finite difference method. The slope for the finite difference method is -3.8, for the SPR method the slope is between -4 and -6. Thus the SPR method seems to be of the same, or higher, order as the finite difference method. In Figure 28 we plot $|e(1)|_{\epsilon,\infty}$ as a function of the amount of work in flops. This result is even more striking, since the order is -1.9 for the finite difference method and -5.4 for the SPR method (when the error is smaller than 0.01). Thus the SPR method, for this specific problem when $p = 4$, is not of order 2 in the number of flops but of order greater than 5 for small enough ϵ.

4.3 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 automatically. Now we will test the methods ability to automatically refine the SPR when features develop in time. We will solve the nonlinear Burgers' equation on the unit interval with

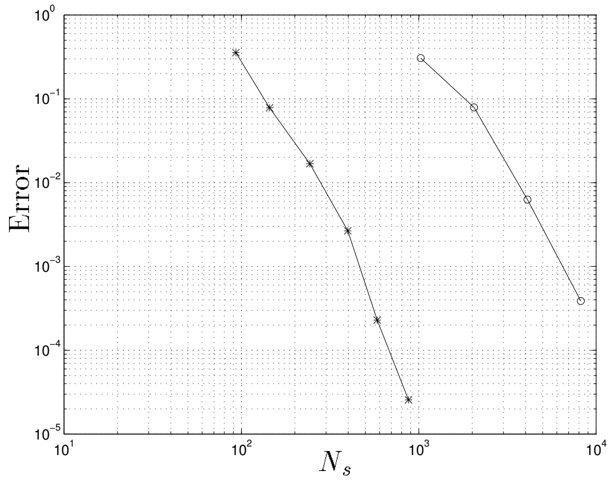


Figure 27: The error at $t = 1$ for the SPR method when $p = 4$ (stars) and a 4th order finite difference method (circles) as a function of the number of significant coefficients at $t = 0$. The initial function has $\alpha = 10^5$.

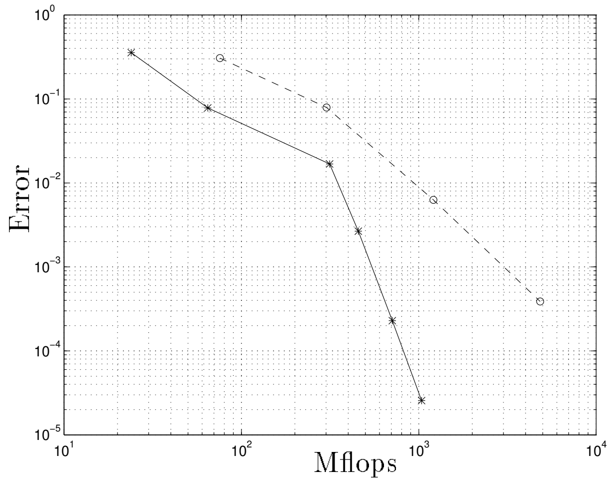


Figure 28: The error at $t = 1$ for the SPR method when $p = 4$ (stars) and a 4th order finite difference method (circles) as a function of Mflops (Megaflops used in the computation). The initial function has $\alpha = 10^5$.

Dirichlet boundary conditions,

$$\begin{cases} u_t + uu_x = \mu u_{xx}, & 0 \le x \le 1, \quad t > 0, \\ u(x,0) = u_0(x), & u = u(x,t), \quad u(0,t) = u(1,t) = 0. \end{cases} \tag{10}$$

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 29 when $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$ and $k = 0.25$. That the SPR method

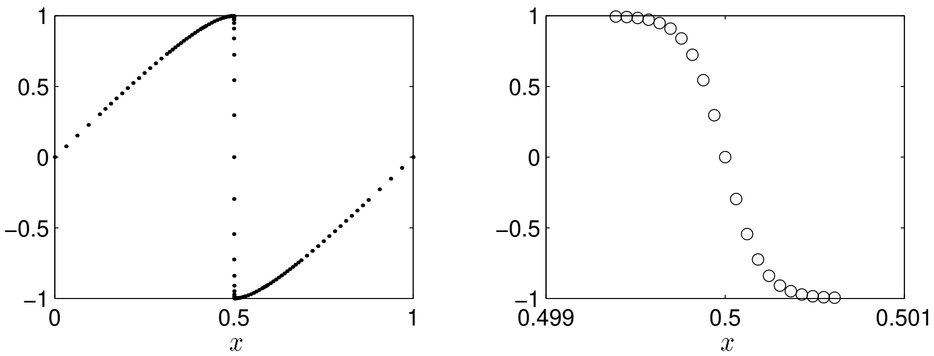


Figure 29: The solution at $t = 0.25$ to the left, and a blowup around the shock to the right, when $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$ and $k = 0.25$.

is able to refine the grid around $x = 1/2$ as the shock develops is evident in Figure 30. 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. The time evolution of N_s is shown in Figure 31. The number of significant coefficients increase, as the gradient around $x = 1/2$ increase, until the shock is resolved on the finest grid. The number of significant coefficients then decrease since the solution away from the shock gets smoother (approximately linear). The adaptability of the time step is also shown in Figure 31. As is evident we make savings in work, since we can take large time steps in the beginning of the computation.

5 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

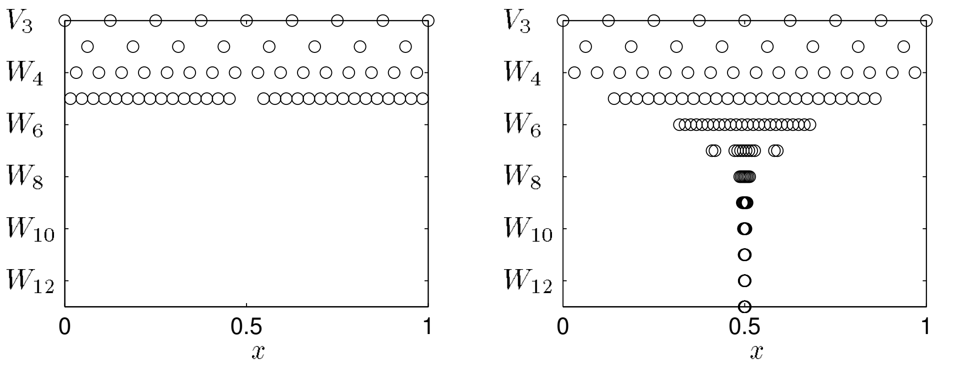


Figure 30: The sparse wavelet representation of the solution; at $t = 0$ to the left, and at $t = 0.25$ to the right, when $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$ and $k = 0.25$.

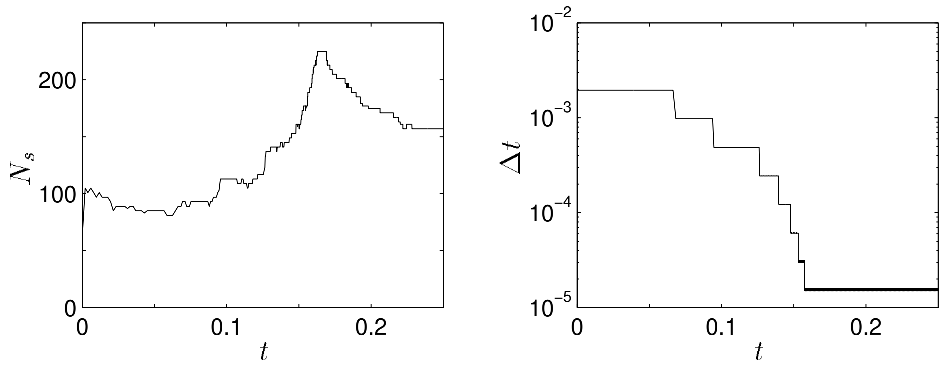


Figure 31: The number of significant coefficients as a function of time to the left, and the time step as a function of time to the right, when $\mu = 10^{-4}$, $p = 4$, $\epsilon = 10^{-5}$ and $k = 0.25$.

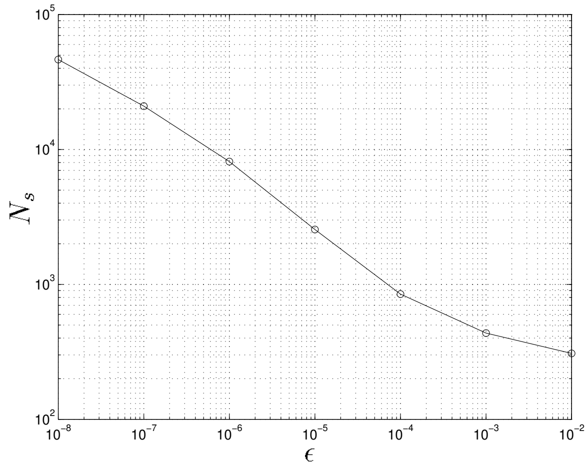


Figure 32: The number of significant coefficients as a function of the threshold. The finest level is $J = 10$. For the test function $f(x, y)$, when $\alpha = 10^3$.

a two-dimensional function in Section 5.1. In Section 5.2 we solve a linear advection problem on the unit square. We then solve an advection problem with variable coefficients in Section 5.3. Finally we solve a nonlinear problem in Section 5.4; a two-dimensional counterpart to the one-dimensional Burgers' equation.

In all two-dimensional numerical experiments we set $\delta < \epsilon$, i.e., we refine one level for all significant coefficients. See Section 3.5 for a definition of δ. All experiments are done in the cubic basis, $p = 4$. The boundary conditions are periodic for all two-dimensional test problems.

5.1 Representation and Differentiation

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 holds 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. In Figure 32 we plot the number of significant coefficients as a function of the threshold and in Figure 33 the error in the representation, and in the x-derivative, as a function of N_s. We find that the slopes of the graphs of N_s and the representation error are -0.49 and -1.97, respectively. This is consistent with the relations

$$\sqrt{N_s} \leq c_2\epsilon^{-1/p}, \text{ and}$$

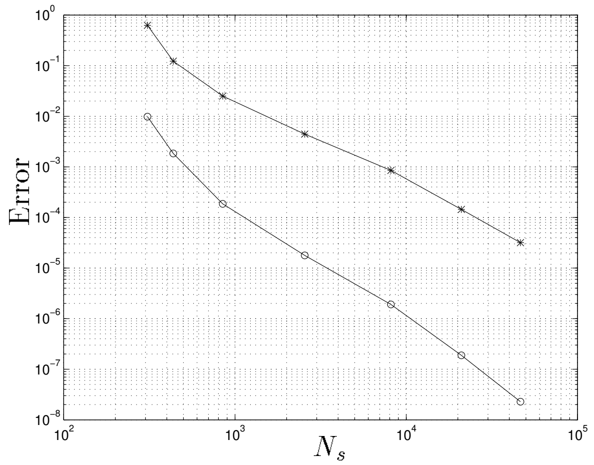


Figure 33: The error in the representation (circles), and in the x-derivative (stars), as a function of N_s. The finest level is $J = 10$. For the test function $f(x, y)$, when $\alpha = 10^3$.

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

that are the two-dimensional counterparts to the one-dimensional estimates, Equations (5) and (6). For the error in the derivative approximation we have a slope of -1.57. As in the one-dimensional case, we seem to lose half an order of approximation.

We also examine the work for differentiating the SPR of the test function. In Figure 34 the error is displayed as a function of the number of flops to compute the derivative. The slope is around -1.5. Comparing with the above dependence of the error on N_s we see that the work is proportional to N_s. The number of flops per significant coefficient for computing the derivative is at least 6, but 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 shown in Figure 35. We see that the number is around 13, higher than the lower bound of 6, but still a small number.

5.2 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

$$\begin{cases} u_t = u_x + u_y, & 0 \leq x, y \leq 1, \quad u = u(x,y,t), \quad t > 0, \\ u(x,y,0) = u_0(x,y), & u(1,y,t) = u(0,y,t), \quad u(x,1,t) = u(x,0,t). \end{cases} \tag{11}$$

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

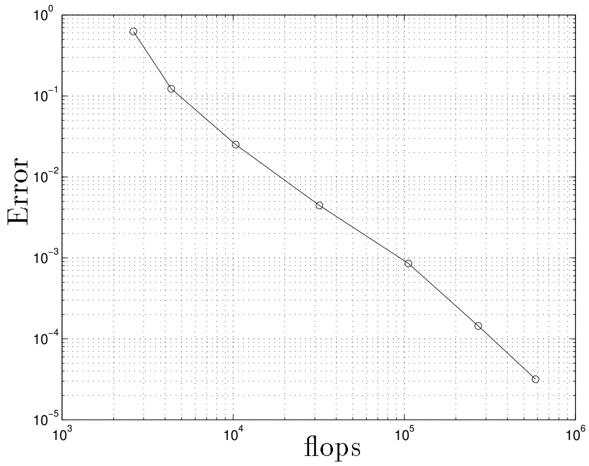


Figure 34: The error in the approximation of the x-derivative of the test functions SPR when $\alpha = 10^3$ as a function of flops.

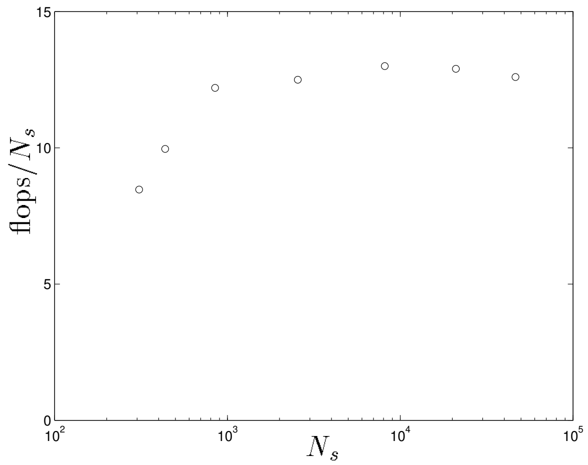


Figure 35: The number of flops per significant coefficient for computing the derivative of the test function's SPR when $\alpha = 10^3$.

a localized spike,

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

The computed solution at $t = 1$ when $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$ and $k = 1$ is shown in Figure 36. Since $t = 1$ the exact solution is the initial function. The error in the solution is shown in Figure 37. Note that the threshold, ϵ,

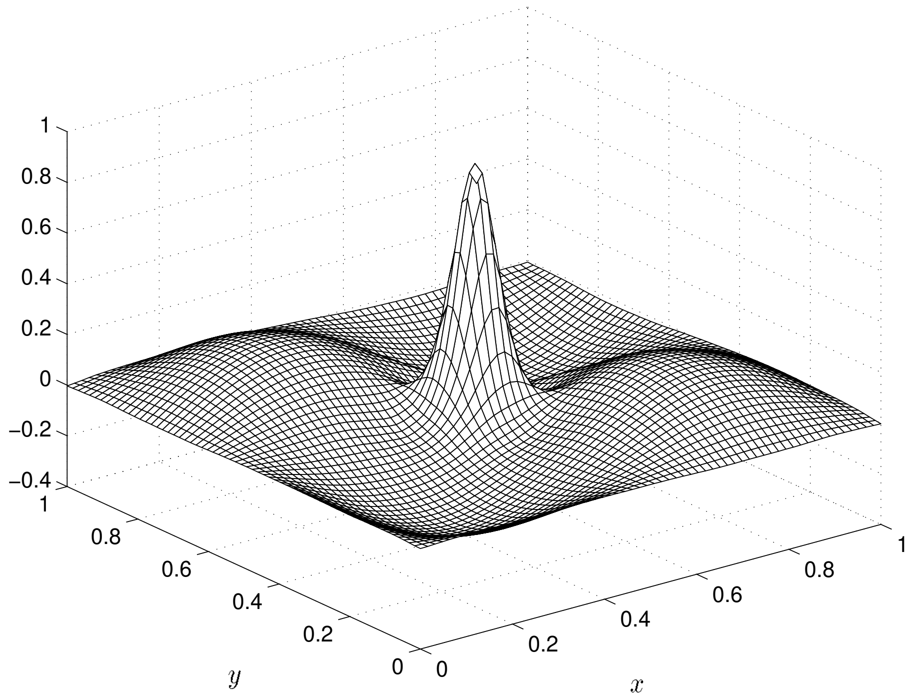


Figure 36: The solution at $t = 1$ when $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$ and $k = 1$.

is quite large. The SPR at $t = 0$ and at $t = 1$ is displayed in Figure 38. We see that the points are still concentrated around the spike at $t = 1$. Thus the grid refinement follow 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 is plotted in Figure 39. It oscillates, but the average is constant over time. To examine the order of the method we show the error at $t = 1$ as a function of the number of significant coefficients in Figure 40, both for the SPR method and for a 4th order finite difference method. The order of the different methods seems to be the same, but the number of significant coefficients needed for a certain error is smaller by at least a factor 10 for the SPR method compared with the finite difference method, when N_s is large enough. In terms of Mflops needed to achieve a certain accuracy, the same pattern is evident as shown in Figure 41. The orders of the two different methods seems to be the same but the SPR

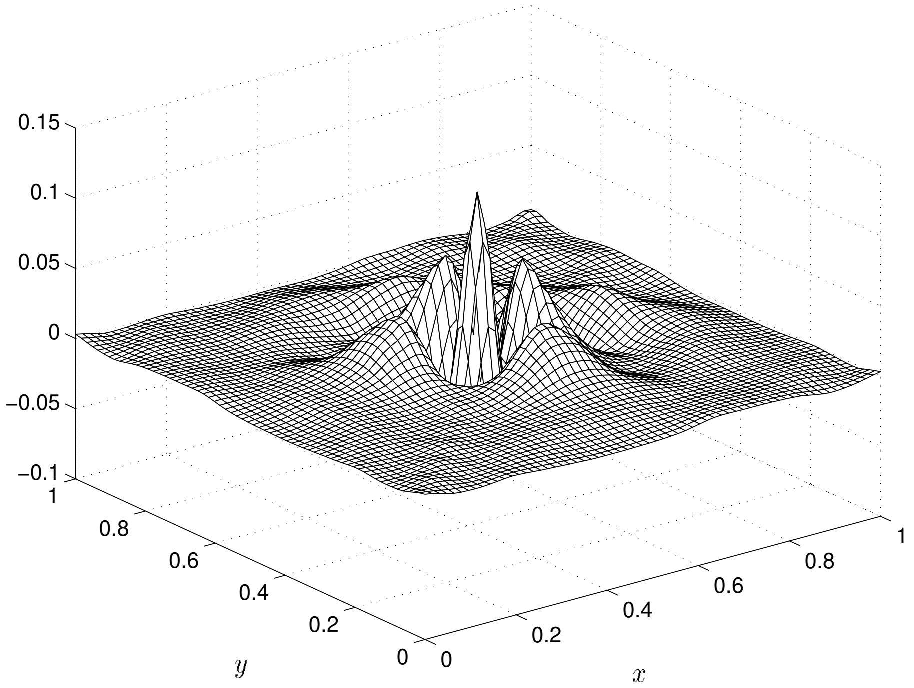
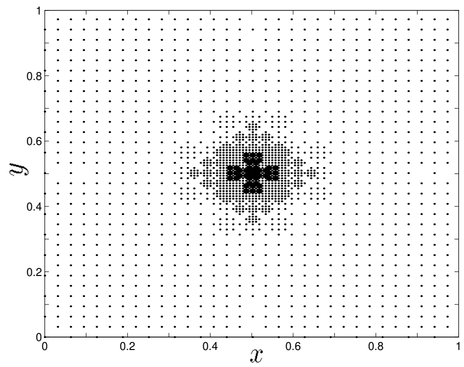
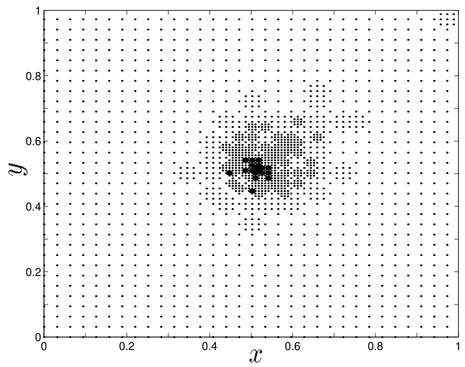


Figure 37: The error at $t = 1$ when $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$ and $k = 1$.



(a) $N_s = 2072$



(b) $N_s = 1865$

Figure 38: To the left the SPR at $t = 0$, to the right the SPR at $t = 1$ when $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$ and $k = 1$.

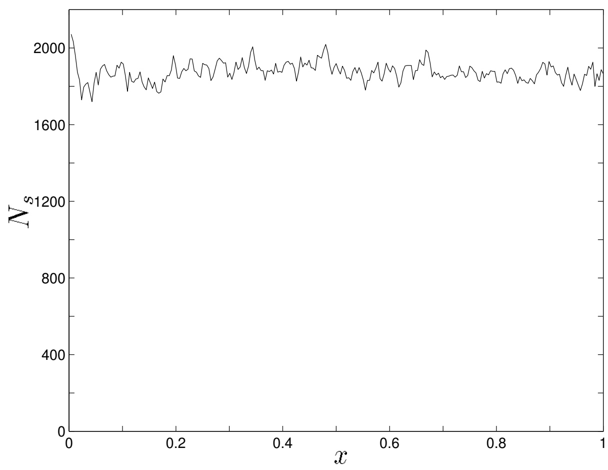


Figure 39: The number of significant coefficients as a function of time when $\epsilon = 10^{-3}$, $\alpha = 200$, $J = 7$ and $k = 1$.

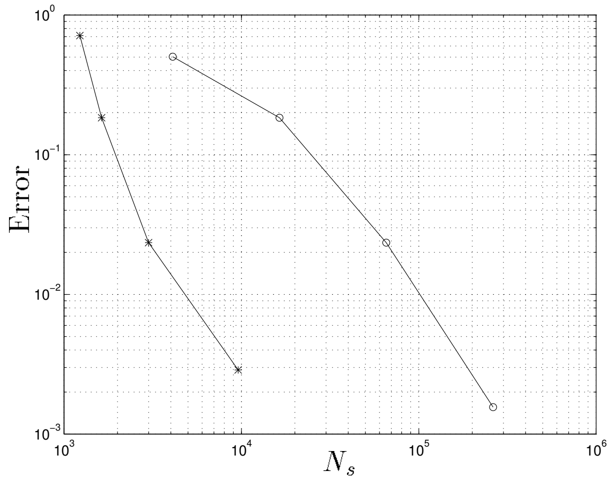


Figure 40: The error at $t = 1$ as a function of the number of significant coefficients, for the SPR method (stars) and a finite difference method (circles). $\alpha = 1000$, $k = 1$ and $J = 7, 8, 9$.

method uses about 1/10 of the flops needed by the finite difference method to produce a solution of the same accuracy.

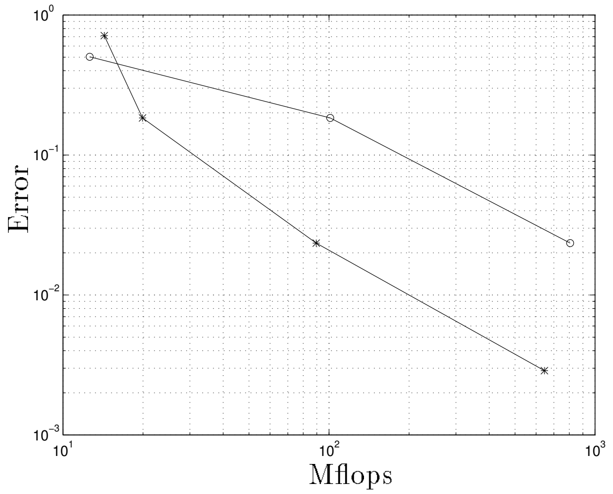


Figure 41: The error at $t = 1$ as a function of Mflops, for the SPR method (stars) and a finite difference method (circles). $\alpha = 1000$, $k = 1$ and $J = 7, 8, 9$.

5.3 A Variable Coefficient Problem

A test of the SPR method's ability to follow features of the solution along curved paths in two dimensions can be constructed by introducing variable coefficients for the advection equation. The variable coefficient problem is

$$\begin{cases} u_t - 2\pi(y - 1/2)u_x + 2\pi(x - 1/2)u_y = 0, & t > 0, \\ u(x, y, 0) = u_0(x, y), & u = u(x, y, t). \end{cases} \tag{12}$$

The solution is a clockwise rotation of the initial function around $(x, y) = (0.5, 0.5)$. To solve the problem on the unit square we prescribe an initial function that is zero (to working precision) outside the circle of radius one centered at $(0.5, 0.5)$,

$$u_0(x) = e^{-\alpha\left((x-3/4)^2+(y-1/2)^2\right)},$$

that is, a peak centered at $(0.75, 0.5)$.

The solution at $t = 1$ when $\epsilon = 10^{-4}$ is shown in Figure 42. The exact solution is the initial function, u_0, since $t = 1$. The corresponding error at $t = 1$ when $\epsilon = 10^{-4}$ is shown in Figure 43. The SPRs at $t = 0$ and $t = 1$ is displayed in Figure 44. We see that the refinement follows the peak as it makes a full clockwise rotation around $(0.5, 0.5)$. There is a trailing "tail" of refinement behind the peak. We also have some isolated "islands" of refinement left behind in the wake of the peak. The number of significant coefficients have increased by approximately 1/3 from $t = 0$ to $t = 1$. The number of significant coefficients

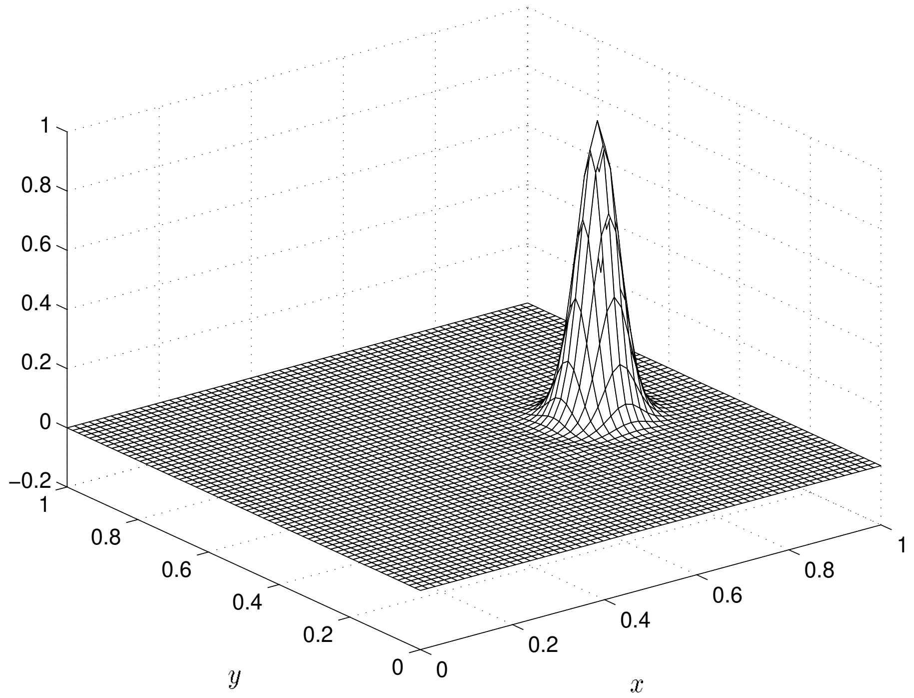


Figure 42: The solution at $t = 1$ when $\alpha = 400$, $\epsilon = 10^{-4}$ and $k = 1/\pi$.

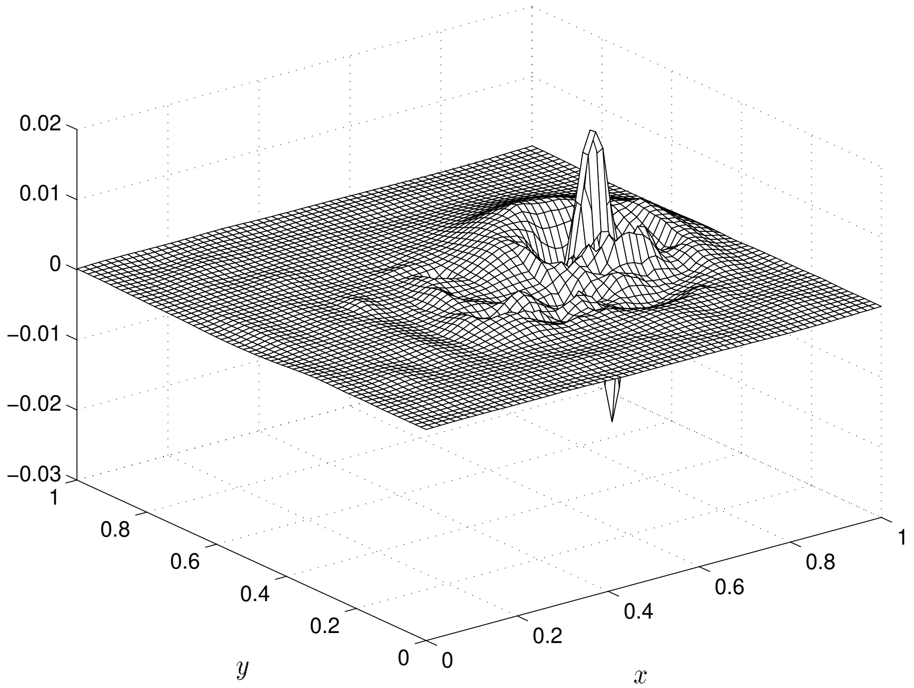
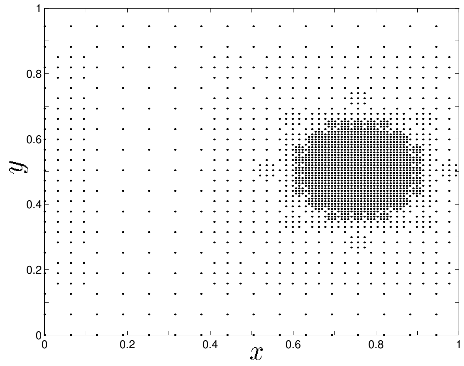
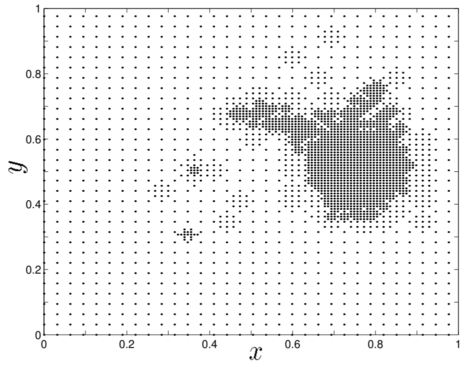


Figure 43: The error at $t = 1$ when $\alpha = 400$, $\epsilon = 10^{-4}$ and $k = 1/\pi$.



(a) $N_s = 1964$



(b) $N_s = 2679$

Figure 44: To the left the SPR at $t = 0$, to the right the SPR at $t = 1$. The parameter values are $\alpha = 400$, $\epsilon = 10^{-4}$ and $k = 1/\pi$.

as a function of time is plotted in Figure 45. We see that the increase in N_s occurs in the beginning of the computation. After that, N_s oscillates around an average value. This is reassuring; a "tail" develops, but it does not grow after an initial transient. We now choose a sharper peak, $\alpha = 1000$, and compare the error at $t = 1$ for the SPR method with the error for a 4th order finite difference method as a function of the number of significant coefficients. The result is shown in Figure 46. The order of convergence for the SPR method is comparable to the order of the finite difference method (actually slightly higher). Comparing the number of flops required to achieve a certain accuracy, we again find that the two methods have comparable order of convergence, as shown in Figure 47. The SPR method require about 1/5 of the number of flops that the finite difference method require to achieve a certain accuracy for this specific problem.

5.4 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-dimensional Burgers' equation, [1]

$$\begin{cases} u_t + u(u_x + u_y) = \mu(u_{xx} + u_{yy}), \quad 0 \le x, y < 1, \quad t > 0, \\ u(x, y, 0) = u_0(x, y), \quad u(1, y, t) = u(0, y, t), \quad u(x, 1, t) = u(x, 0, t). \end{cases} \tag{13}$$

[1]This is the scalar counterpart to the two-dimensional system called the bi-dimensional Burgers' equation by Ponenti and Liandrat [18].

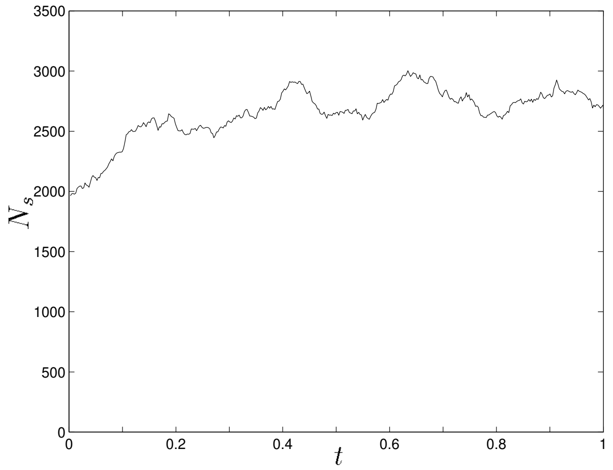


Figure 45: The number of significant coefficients as a function of time. The parameter values are $\alpha = 400$, $\epsilon = 10^{-4}$ and $k = 1/\pi$.

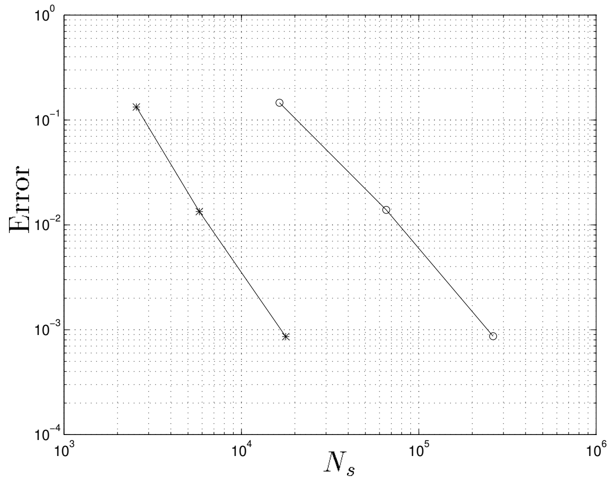


Figure 46: The error at $t = 1$ as a function of the number of significant coefficients, for the SPR method (stars) and a finite difference method (circles), when $\alpha = 1000$ and $J = 7, 8, 9$.

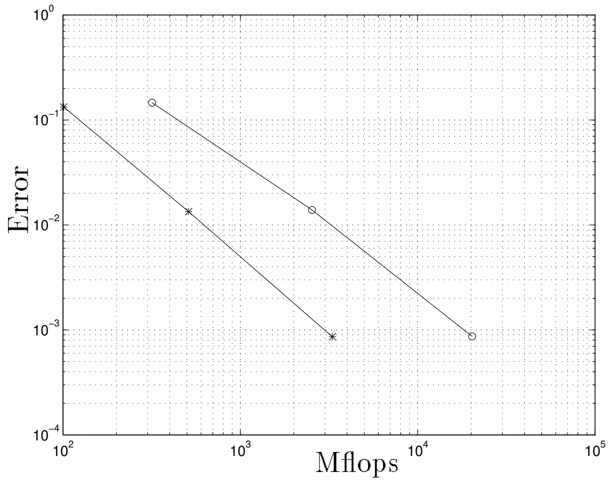


Figure 47: The error at $t = 1$ as a function of Mflops, for the SPR method (stars) and a finite difference method (circles), when $\alpha = 1000$ and $J = 7, 8, 9$.

with periodic boundary conditions. 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' 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 48 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

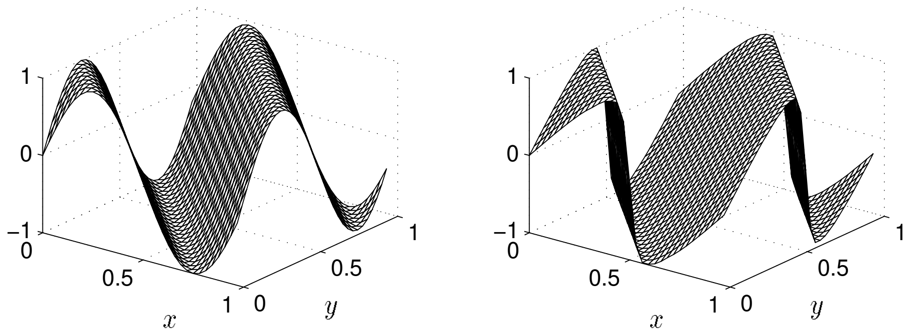


Figure 48: To the left the initial function, to the right the solution at $t = 0.09$, when $\epsilon = 10^{-3}$, $\mu = 10^{-2}$ and $J = 7$.

Figure 49. 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 can examine

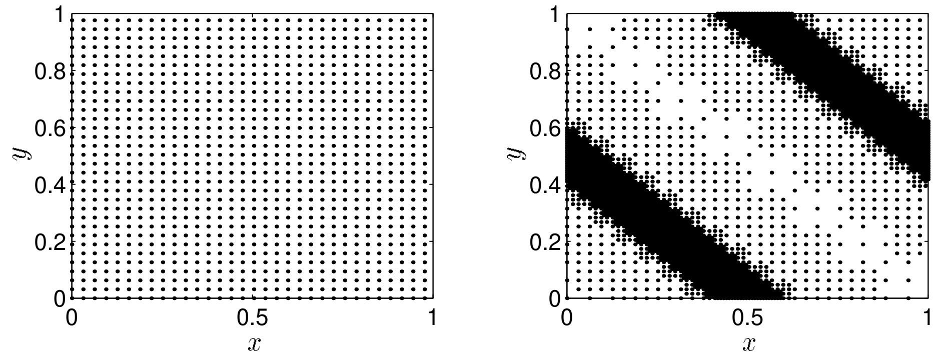


Figure 49: To the left the SPR at $t = 0$, $N_s = 1024$; to the right the SPR at $t = 0.09$, $N_s = 4352$; when $\epsilon = 10^{-3}$, $\mu = 10^{-2}$ and $J = 7$.

the solution in more detail by looking at it along the line $x = y$. A plot of the solution along this line at $t = 0$ and $t = 0.09$ is shown in Figure 50. We

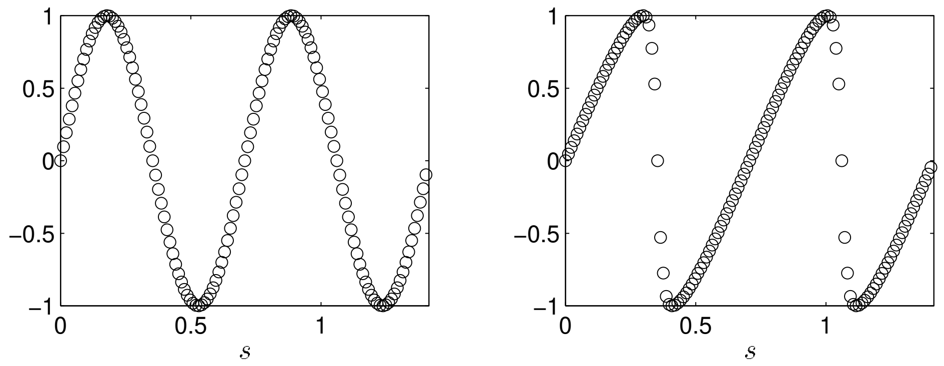


Figure 50: To the left the initial function along the diagonal (sx, sy), to the right the solution along the diagonal at $t = 0.09$, when $\epsilon = 10^{-3}$, $\mu = 10^{-2}$ and $J = 7$.

have seen that the SPR method is able to adapt the two-dimensional grid to developing features of the solution.

6 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 lead to significant savings in the number of flops needed to achieve a solution with a certain accuracy in maximum norm. Since we 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 time. If a sparse data structure is used to store the sparse point representation (SPR) we also reduce the memory requirement. The actual order of the method is found to be as high, or higher than for an ordinary finite difference method when applied to one- and two-dimensional problems. The method is shown to outperform 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] L. Andersson, N. Hall, B. Jawerth, and G. Peters. Wavelets on closed subsets of the real line. In *Recent Advances in Wavelet Analysis*, volume 3, pages 1–61. Academic Press, 1994.

[2] Silvia Bertoluzza, Giovanni Naldi, and Jean Christophe Ravel. Wavelet methods for the numerical solution of boundary value problems on the interval. In Charles K. Chui, Laura Montefusco, and Luigia Puccio, editors, *Wavelets: Theory, Algorithms and Applications*, pages 425–448. Academic Press, 1994.

[3] Gregory Beylkin and Naoki Saito. Multiresolution representation using the auto-correlation functions of compactly wavelets. *IEEE Transactions on Signal Processing*, 41(12):3584, December 1993.

[4] Barna L. Bihari and Ami Harten. Application of generalized wavelets: an adaptive multiresolution scheme. Presented at the International Conference on Wavelets, Taormina, Italy, October 14–20, 1993.

[5] Albert Cohen and Ingrid Daubechies. Wavelets on the interval and fast wavelet transforms. *Applied and Computational Harmonic Analysis*, 1:54–81, 1993.

[6] Ingrid Daubechies. *Ten Lectures on Wavelets*, volume 61 of *CBMS-NSF regional conferences series in applied mathematics*. SIAM, 1992.

[7] G. Deslauriers and S. Dubuc. Symmetric iterative interpolation processes. *Constructive Approximation*, 5(1):49–68, 1989.

[8] David L. Donoho. Interpolating wavelet transforms. Technical Report 408, Dept. of Statistics, Stanford University, November 1992.

[9] Serge Dubuc. Interpolation through an iterative scheme. *Journal of Mathematical Analysis and Applications*, 114:185–204, 1986.

[10] Jochen Frölich and Kai Schneider. An adaptive wavelet-vaguelette algorithm for the solution of nonlinear PDEs. Preprint SC 95-28, ZIB, Berlin, November 1995.

[11] Bertil Gustafsson, Heinz-Otto Kreiss, and Joseph Oliger. *Time Dependent Problems and Difference Methods.* Wiley, 1996.

[12] Mats Holmström and Johan Waldén. Adaptive wavelet methods for hyperbolic PDEs. Technical Report 171, Dept. of Scientific Computing, Uppsala University, Box 120, S-751 04 Uppsala, Sweden, December 1995.

[13] Leland Jameson. On the wavelet optimized finite difference method. Technical Report ICASE 94-9, NASA Langely Research Center, March 1994.

[14] Leland Jameson. A wavelet-optimized, very high order adaptive grid and order numerical method. Technical Report ICASE 96-30, NASA Langely Research Center, May 1996.

[15] J. Keiser. *Wavelet Based Approach to Numerical Solution of Nonlinear Partial Differential Equations.* PhD thesis, University of Colorado, Boulder, 1995.

[16] S. Mallat. Multiresolution approximation and wavelets. *Transactions of the American Mathematical Society*, 315:69–88, 1989.

[17] Fredrik Olsson and N. Anders Petersson. Stability of interpolation on overlapping grids. *Computers & Fluids*, 25(6):583–605, 1996.

[18] Pj. Ponenti and J. Liandrat. Numerical algorithms based on biorthogonal wavelets. Technical Report ICASE 96-13, NASA Langely Research Center, February 1996.

[19] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, 1980.

[20] Wim Sweldens and Peter Schröder. Building your own wavelets at home. Technical Report IMI 1995:5, Dept. of Mathematics, University of South Carolina, 1995.

[21] Oleg V. Vasilyev and Samuel Paolucci. A dynamically adaptive multilevel wavelet collocation method for solving partial differential equations in a finite domain. *Journal of Computational Physics*, 125(2):498–512, 1996.

[22] Oleg V. Vasilyev, Samuel Paolucci, and Mihir Sen. A multilevel collocation method for solving partial differential equations in a finite domain. *Journal of Computational Physics*, 120(1):33–47, 1995.

A Appendix

In this appendix we will present some source code for the one-dimensional sparse wavelet transform (SWT) on the unit interval. The functions are written in MATLAB, but are easily ported to other programming languages since no high level MATLAB commands are used. The code was written to clearly expose the structure of the algorithms so it is certainly not an optimal implementation.

When implementing the SWT one must decide how to represent the sparse wavelet representation (SWR) and the sparse point representation (SPR), i.e., how to keep track of the significant coefficients. To make the implementation memory efficient we would like to only store the significant coefficients. Thus we need some sort of sparse structure. Since we for now are interested in how the algorithms work, we choose another solution. All coefficients are stored in an array and we mark nonsignificant values by setting the corresponding element's value to the IEEE value `NaN` (Not-a-Number). When doing computations we then iterate over *all* coefficients, but only perform computations if the current coefficient is not a `NaN`. This can be checked in MATLAB by the function `isnan`. In other programming languages we can replace the `NaN`-value by any other bit-pattern that is certain to not occur during the computations and the `isnan`-function can be replaced by a user written one. We again stress the fact that this storage method is only intended to clearly expose the SWT algorithm, in a real-world implementation we must use sparse storage to avoid the performance loss of iterating over all coefficients and take advantage of the method's memory efficiency.

First we present the function `pr2spr` that given an array `u` with $2^l + 1 \geq p$ elements, returns the same array containing the SPR of `u`. The other arguments is the threshold ϵ and the order of the method, p. Note that we never actually form the SWR of `u`, since we only examine one point value at a time to decide if it should remain untouched or if it should be replaced by a `NaN`. The actual interpolation is done in the function `spval` that we will describe later.

```
function u = pr2spr(u, ep, p)

% PR2SPR transform a point representation to sparse point representation
%       PR2SPR(U,EP,P) sets a point value in U to NaN if the corresponding
%       wavelet coefficient has magnitude smaller than the threshold EP.  P
%       is the order of the interpolation.  L is the number of levels in
%       the transform.  The length of U must be on the form 2^L+1 >= P.
%       On the unit interval.

n = length(u);
l = log2(n-1) - log2(p);
spc = 1;
for j = l:-1:1                         % from fine to coarse
  spc = spc*2;
  for k = spc/2+1 : spc : n            % points on current level
    if ~isnan(u(k))
      if abs(u(k) - spval(u, k, p, l, spc)) < ep
        u(k) = nan;
      end
    end
  end
end
```

We now turn to the function **spr2pr** that does the inverse of the previous function. Given an array corresponding to an SPR it returns the same array, now contains the function values in all its elements, i.e., all **NaN**s have been replaced by interpolated function values.

```
function u = spr2pr(u, p)

% SPR2PR transform a sparse point representation to a point representation
%
%       SPR2PR(U,P) computes the point representation of U.  P is the
%       order of the interpolation.  L is the number of levels in the
%       transform.  On the unit interval.

n = length(u);
l = log2(n-1) - log2(p);
spc = 2^l;
for j = 1:l                            % from coarse to fine
  for k = spc/2+1 : spc : n            % points on current level
    if isnan(u(k))
      u(k) = spval(u, k, p, l, spc);
    end
  end
  spc = spc/2;
end
```

Lastly, we have the function that does the real work, **spval**. In this function we generate an interpolated function value from the function values at the coarser grid, recursively. As we can see linear and cubic interpolation is supported, but for higher orders we simply need to add the correct weights in the function. Note that the recursion is not needed in this application since the points we interpolate from always exist, but is needed when evaluating an SPR at an arbitrary point.

```
function uk = spval(u, k, p, l, spc)

% SPVAL evaluate a sparse point representation.
%
%       SPVAL(U,K,P,L,SPC) computes the value of U at K.  P is the order of
%       the interpolation.  L is the number of levels in the transform.
%       SPC is the spacing at the level above K. On the unit interval.

n = length(u);                          % total number of points
i = zeros(1,p);                         % index array
v = zeros(1,p);                         % interpolation values
h = zeros(1,p);                         % filter coefficients

for j = 1:p/2                           % points to interpolate from
  i(p/2+1-j) = k-(j-1/2)*spc;   i(p/2+j) = k+(j-1/2)*spc;
end

if p == 2                               % linear interpolation
  h = [1 1]/2;
elseif p == 4                           % cubic interpolation
  if i(1) < 1
    i = [1 1+spc 1+2*spc 1+3*spc];      % left boundary
    h = [5 15 -5 1]/16;
  elseif i(p) > n
    i = [n-3*spc n-2*spc n-spc n];      % right boundary
    h = [1 -5 15 5]/16;
  else                                  % interior
    h = [-1 9 9 -1]/16;
  end
end

for j = 1:p                             % values to interpolate from
  if isnan(u(i(j)))
    v(j) = spval(u, i(j), p, l);        % recursion
  else
    v(j) = u(i(j));
  end
end

uk = 0;                                 % compute interpolated value
for j = 1:p
  uk = uk + h(j)*v(j);
end
```

We now show an example of how to use the above presented functions. We generate a test function, transform it to an SPR, and transform back. We then plot the different representations. A sample MATLAB session follows, and in Figure 51 we display the resulting plot.

```
                        < M A T L A B (R) >
              (c) Copyright 1984-94 The MathWorks, Inc.
                        All Rights Reserved
                           Version 4.2c
                            Dec 31 1994

>> x = linspace(0,1,129);                    % 129 points on the finest scale
>> u = sin(2*pi*x) + exp(-1e3*(x-.5).^2);    % our test function
>> ep = 0.1;   p = 4;                        % threshold and order of approx.
>> usp = pr2spr(u,ep,p);                     % the SPR of u
>> upr = spr2pr(usp,p);                      % the reconstructed PR
>> plot(x,u,x,usp,'o',x,upr,'--')
```

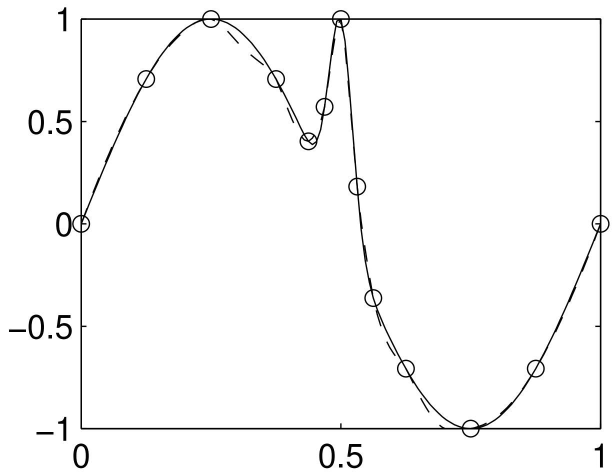


Figure 51: The resulting plot when executing the above MATLAB commands. Solid line is the original function, u. The circles is the SPR, and the dashed line the reconstruction on the finest scale.