An Adaptive Finite Difference Method for Time Dependent PDEs

by

Mats Holmström

Report No. 199/1997

UPPSALA UNIVERSITET
Institutionen för
teknisk databehandling

UPPSALA UNIVERSITY
Department of
Scientific Computing

An Adaptive Finite Difference Method for Time Dependent PDEs

Mats Holmström [1]

August, 1997

[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 time-dependent PDEs. The method is based on an interpolating wavelet transform using polynomial interpolation on dyadic grids. The adaptability is performed by choosing the representation based on the magnitude of the wavelet coefficients. Any finite difference discretization can then be used. As a numerical example the time-dependent, compressible, Navier-Stokes equations are solved for flow over a flat plate using centered finite differences. It is found that the proposed method outperforms a finite difference method on a uniform grid in terms of CPU time when a highly accurate solution is wanted.

Contents

1 Introduction

Solutions to partial differential equations (PDEs) often behave differently in different areas. Examples from fluid dynamics are shocks and boundary layers. In both cases the solution can be smooth in most of the solution domain, with small areas where there are steep gradients in the solution (around the shock and in the boundary layer, respectively). When solving such problems numerically we would like to adjust the computational grid to the solution. In terms of finite difference methods, we want to have many points in areas where there is a lot of variation in the solution, and few points in areas where the solution is smooth. We are then faced with two questions. How do we construct this non-uniform grid? How do we discretize the PDE on this grid? In this paper the sparse point representation (SPR) method provides answers to these questions.

We are interested in solving time-dependent PDEs time-accurately. That is, we do not seek some steady-state solution, but we want accurate solutions to the PDE at all times. Therefore we use the method of lines to separately discretize the PDE in space and time. This paper will concentrate on the space discretization. We then use a standard ODE solver of sufficiently high order for the time integration.

Several different approaches for combining finite difference and wavelet methods have been considered. Jameson [10, 11] uses wavelets for finding where to refine the grid in a finite difference method, and then uses finite difference stencils on an irregular grid. Harten [8] has 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 [17, 16], where the computation of certain matrix operators is needed, and Fröhlich and Schneider [6] in combination with non-compactly supported interpolating scaling functions. The interpolating wavelet transform in this paper has been used in a Galerkin method for solving elliptic problems on the interval by Bertoluzza, Naldi and Ravel [1]. A general filter bank method was described by Waldén [18]. In an earlier report by the author [9] the SPR method was introduced for solving hyperbolic PDEs, which is the basis of this work.

As a motivation for the SPR method; consider the following scenario. We want to solve a time-dependent PDE with solutions that has the features mentioned above (smooth in most parts with small areas of large variation). This can be done accurately by using a finite difference method on a fine grid. Assume that there are N points on the fine grid. If we want a more accurate solution we increase N. The problem is that when N get large we eventually find that the problem is too computationally expensive to solve. The SPR method essentially remove points from the fine grid in regions where the solution is smooth and keep points in regions of large variation. The method could be labeled a "sparse finite difference" method since we use adaptability in space to reduce the computational time, and memory requirements, of a standard finite difference method. A parameter ϵ controls the representation error on the new, thresholded, non-uniform grid, and implicitly controls the number of remaining points N_s, on this new grid. Conceptually we are still performing our computations on the fine uniform grid, but we have now introduced an error

proportional to ϵ at all removed grid-points. In practice the computation now only includes the remaining N_s points, but the SPR provides us with a method to reconstruct any point-value on the original fine grid, thus allowing us to use standard finite difference approximations to the derivatives although the grid is now non-uniform. The SPR of course introduce some computational overhead compared to a standard finite difference solver on the fine grid, so we want N_s to be substantially smaller than N, but this is true for the mostly smooth solutions described earlier.

In Section 2 we describe the physical problem, solving the time-dependent, compressible, Navier-Stokes equations for flow over a flat plate and present a discretization by a finite difference method. We then describe the SPR method and how it is applied, in the framework of the finite difference method, in Section 3. In Section 4 we compare the performance of the finite difference and the SPR method when applied to the flow problem.

2 Background

In Section 2.1 we present the time-dependent, compressible, Navier-Stokes equations. We then describe the geometry and the associated boundary conditions of the specific problem we want to solve in Section 2.2, flow over a flat plate. In Section 2.3 we describe the finite difference approximation that we will use.

2.1 The Navier-Stokes Equations

The time-dependent compressible Navier-Stokes equations in non-dimensional form can be written as

$$\left\{\begin{array}{rcl} \rho_t &=& -\rho_x u - \rho_y v - \rho(u_x + v_y) \\ u_t &=& -uu_x - vu_y + \frac{1}{\rho \mathrm{Re}}\left((2\mu+\lambda)u_{xx} + (\mu+\lambda)v_{xy} + \mu u_{yy}\right) \\ && -\frac{R}{\rho}(\rho_x T + \rho T_x) \\ v_t &=& -vv_y - uv_x + \frac{1}{\rho \mathrm{Re}}\left((2\mu+\lambda)v_{yy} + (\mu+\lambda)u_{xy} + \mu v_{xx}\right) \\ && -\frac{R}{\rho}(\rho_y T + \rho T_y) \\ T_t &=& -uT_x - vT_y - (\gamma-1)(u_x+v_y)T + \frac{\gamma k}{\rho \mathrm{RePr}}(T_{xx}+T_{yy}) \\ && +\frac{\gamma-1}{\rho \mathrm{Re}R}\left(\lambda(u_x+v_y)^2 + \mu(u_y+v_x)^2 + 2\mu(u_x^2+v_y^2)\right) \end{array}\right. \quad (1)$$

where ρ is the density, u is the velocity in the x-direction, v in the y-direction and T is the temperature. The parameters are the Reynolds number Re, the shear viscosity μ, the second viscosity λ, the ratio of specific heats γ, the heat conductivity k, the Prandtl number Pr and the parameter R, which is related to the free stream Mach number, M_∞ by $R = (\gamma M_\infty^2)^{-1}$.

2.2 Flow Over a Flat Plate

The geometry of the flow problem is shown in Figure 1. This problem have been studied by many authors, e.g., in an article by Koren [12]. We set free stream values for the dependent variables, and denote them by $\rho_\infty = 1$, u_∞, $v_\infty = 0$ and $T_\infty = 1$.

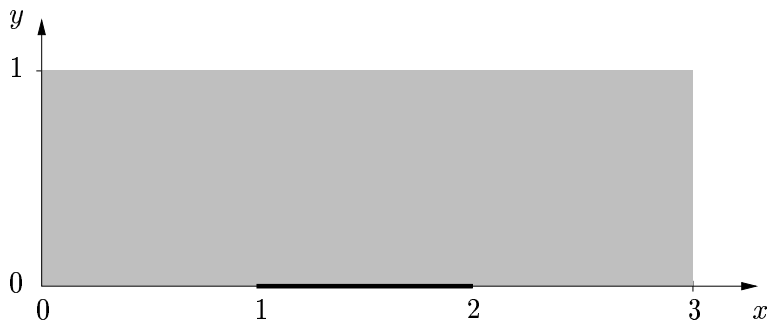

Figure 1: The geometry of the plate-flow problem. To the left is the inflow boundary and to the right the outflow. The black rectangle is the plate, which is really infinitely thin. The gray area is the computational domain.

On the upper boundary ($y = 1$) we set all normal derivatives to zero. On the outflow boundary ($x = 3$) we set the normal derivative of v and T to zero and specify $u = u_\infty(t)$. On the inflow boundary we set $v = v_\infty$ and require that

$$u + \frac{2\sqrt{\gamma RT}}{\gamma - 1} = Tp^{(1-\gamma)/\gamma},$$

where the pressure, $p = \rho RT$. These boundary conditions have been shown to be stable [7]. On the plate we have the no-slip condition $u = v = 0$ and we specify the temperature, $T = T_\infty$. On the non-plate part of the lower boundary we use symmetry for all variables, except v for which anti-symmetry is used. The plate will at time zero be at rest and we will then accelerate the plate along the negative x-axis. This can be simulated by choosing a time-dependent free stream x-velocity, $u_\infty = u_\infty(t)$. We have used the function

$$u_\infty(t) = \begin{cases} 0, & t \le 0, \\ t^2(3 - 2t), & 0 < t < 1, \\ 1, & t \ge 1. \end{cases}$$

to provide a smooth acceleration of the plate. We also need to add the term $u'_\infty(t)$ in the right hand side of the u-component of the Navier-Stokes equations (1) to compensate for the moving frame of reference.

2.3 A Finite Difference Method

The problem is discretized separately in time and space using the method of lines. In time we can use any standard method for solving the resulting system of ODEs. We are interested in solving the PDE time-accurately and use an explicit time-stepping method. Since this paper concentrates on the space discretization, we choose an explicit ODE solver of at least as high order as the space discretization. In the numerical experiments we have used the standard 4th order Runge-Kutta method, and choose a small enough time-step so that the errors in the computations are mainly from the space discretization.

In space we discretize the Navier-Stokes equations by a centered finite difference scheme of order p. On the boundaries we use one-sided stencils where boundary conditions are not available.

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

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

Here h is the distance between grid points. 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 1 the filter coefficients for the first derivative approximation at the left boundary are shown when $p = 2$ and 4. The coefficients at the right boundary are reversed in order and of opposite sign. In Table 2 the filter coefficients for the second derivative approximation at the left boundary are shown when $p = 2$ and 4. The coefficients at the right boundary are reversed in order. In two dimensions the partial derivatives in each direction are evaluated using the above described one-dimensional approximations. Mixed derivatives of the type u_{xy} are approximated by successive approximations of the first derivative in each direction.

Table 1: 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			-3/2	2	-1/2		
	≥ 1		-1/2	0	1/2			
$p = 4$	0			-25/12	4	-3	4/3	-1/4
	1		-1/4	-5/6	3/2	-1/2	1/12	
	≥ 2	1/12	-2/3	0	2/3	-1/12		

This centered discretization (of order 2 and 4) was shown to be stable by Sjögreen [14] for a linearized problem. The full Navier-Stokes equations was found to be stable when some artificial viscosity was added.

Artificial viscosity is introduced by adding the term

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

in all the discrete equations. We have used the parameter value $d = 0.05$ in all our numerical experiments since that led to stable solutions. In Table 3 the filter coefficients for the artificial viscosity at the left boundary are shown. The coefficients at the right boundary are reversed in order.

Table 2: 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			2	-5	4	-1		
	≥ 1		1	-2	1				
$p=4$	0			15/4	-77/6	107/6	-13	61/12	-5/6
	1		5/6	-5/4	-1/3	7/6	-1/2	1/12	
	≥ 2	-1/12	4/3	-5/2	4/3	-1/12			

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

$l=$	-2	-1	0	1	2	3	4	5
0			3	-14	26	-24	11	-2
1		2	-9	16	-14	6	-1	
≥ 2	1	-4	6	-4	1			

3 The SPR Method

We will briefly present the Sparse Point Representation (SPR) and the underlying interpolating subdivision scheme along with the extension to an interpolating wavelet transform in Section 3.1. In Section 3.2 we discuss the choice of transform in two dimensions. Finally, in Section 3.3 we present a block SPR method. Details of the SPR, and its applications for solving some hyperbolic PDEs, can be found in an earlier paper by the author [9].

3.1 The Sparse Point Representation—SPR

The starting point is the interpolating subdivision scheme introduced by Deslauriers and Dubuc [3, 5]. Assume that we have a set of dyadic grids on the real line, $V_j = \{x_{j,k} \in \mathbb{R} : x_{j,k} = 2^{-j}k, k \in \mathbb{Z}\}$, $j \in \mathbb{Z}$. 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 2. Function values on finer grids are generated by interpolation from the next coarser grid by a polynomial of degree $p-1$.

The wavelet coefficients associated with the interpolating subdivision scheme was introduced by Donoho [4] (for the linear case independently by Harten [8]). The wavelet coefficients are computed as the difference between a known function value and the value predicted by interpolating subdivision. That is, the wavelet coefficients encode the error when interpolating the function values from a coarser scale. This can also be formalized in terms of wavelets and scaling

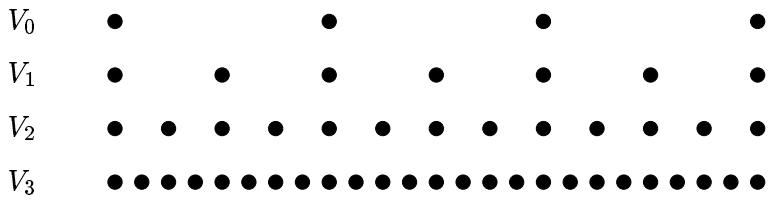

Figure 2: Examples of point positions on dyadic grids.

functions [9].

The sparse point representation (SPR) is created by retaining only those points that correspond to the significant wavelet coefficients, i.e. whose magnitude is grater than ϵ. If any point value is needed that do not exist in the SPR, we interpolate the value from a coarser scale by interpolating subdivision, recursively.

An advantage of interpolating subdivision, and the associated wavelet transform, is that the adaption to an interval is straightforward due to the usage of interpolating polynomials in the construction. 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.

A more detailed presentation of interpolating subdivision, the interpolating wavelet transform, and the related lifting scheme can also be found in a paper by Sweldens and Schröder [15].

3.2 The Choice of Basis

To represent a function in higher dimensions one can construct bases from a one-dimensional basis. For wavelet bases there are two commonly used approaches, which we in this section will call the standard and the non-standard basis. The former uses tensor products of one-dimensional basis functions to create higher dimensional basis functions. The latter instead uses tensor products of the one-dimensional wavelet and scaling function spaces [2]. For multi-dimensional SPRs we have used the non-standard basis construction since it is necessary if we want to keep the property of each wavelet coefficient corresponding to a point value. This being a necessity for the fast algorithms for multiplication and differentiation. On the other hand, this approach can lead to less compression of certain functions than if we had used the standard basis. An example is a smooth function in two dimensions with a large gradient along a line that is oriented parallel with a coordinate axis. Since the standard basis includes basis functions that are elongated along the coordinate axes, while all basis functions in the non-standard basis are square, it is possible that we will have a good representation using few standard basis functions along the line, while in the non-standard case we will need many basis functions on the finest scale along the line to get a good representation.

To investigate this we examine a function that is relevant to the topic of this article; the velocity parallel to the plate in the two-dimensional Blasius

solution to flow over a flat plate [13]. Here the sharp variation of the function is restricted to the boundary layer on the plate. The Blasius solution is shown in Figure 3 for three different values of x. Since the velocity goes from zero to one

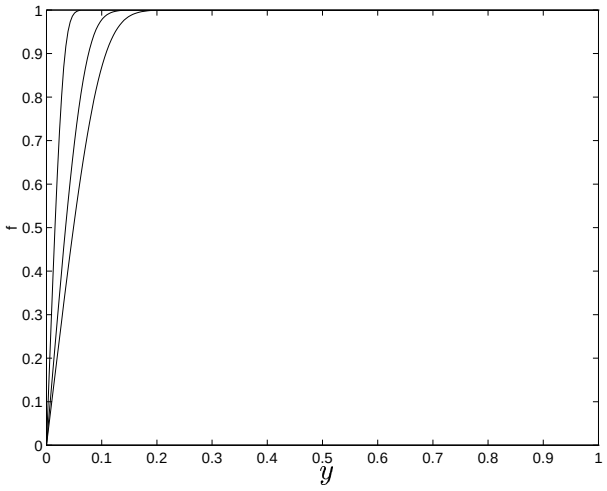

Figure 3: The Blasius solution at $x = 0.1$, 0.5 and 1.0, when $\mathrm{Re} = 1000$.

in the boundary layer, which grows as we move along the plate, we expect that a standard basis will yield a sparser representation than the non-standard basis used in the SPR. In Table 4 the point-wise error and compression in a standard basis is compared with the SPRs non-standard basis. As is evident in the table,

Table 4: The point-wise error and compression in a standard basis and a non-standard basis. The test case is the Blasius solution at $\mathrm{Re} = 1000$ on a 257×257 point discretization of the unit square. The threshold was choosen to get errors of comparable sizes.

	p	N_s	ϵ	Error
Standard	2	1024	$0.5 \cdot 10^{-4}$	$1.44 \cdot 10^{-4}$
SPR	2	2976	$1.0 \cdot 10^{-4}$	$1.42 \cdot 10^{-4}$
Standard	4	540	$0.5 \cdot 10^{-4}$	$1.17 \cdot 10^{-4}$
SPR	4	900	$1.0 \cdot 10^{-4}$	$1.59 \cdot 10^{-4}$

the standard basis needs about 1/3 of the coefficients compared to the SPR when $p = 2$, and about 1/2 of the coefficients when $p = 4$. Considering that the original number of coefficients on the finest grid was 66049 we still have very good compression using the SPR. The percentage of retained coefficients are 4.5% and 1.4% respectively for the SPR. The locations of the retained points in the SPR are shown in Figure 4.

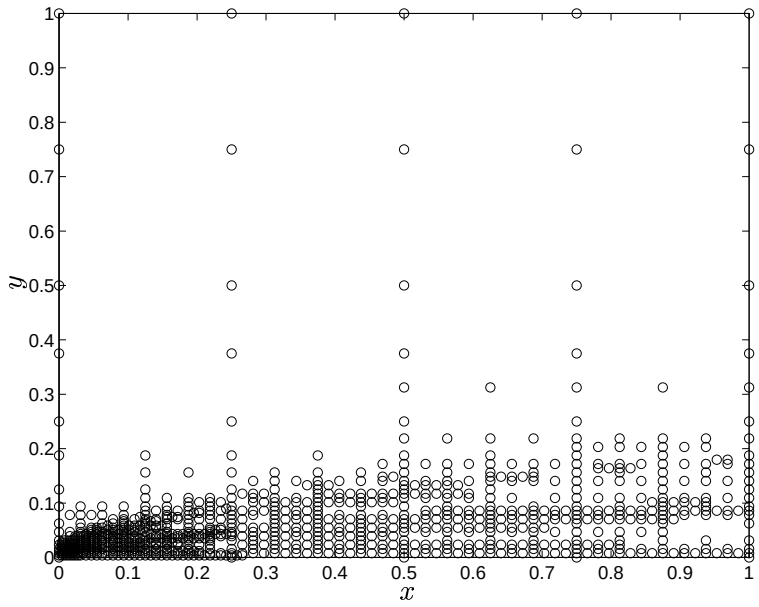

Figure 4: The locations of the points in the SPR for a thresholded Blasius solution, when $\epsilon = 10^{-4}$, $p = 4$ and $\mathrm{Re} = 1000$.

3.3 A Two-Dimensional Block SPR

The choice of the sparse representation is important from a performance point of view. On one hand we get good compression by using the SPR introduced in [9] where the representation consists of single point values. On the other hand, using the SPR makes the overhead in terms of computational complexity and memory requirements large. We therefore choose to use a block approach. The representation consists of a hierarchy of $n_{bi} \times n_{bj}$ grids, with the value of each component of the system of PDEs that we want to solve stored at each grid-point. Here n_{bi} is the number of points in each block in the x-direction, and n_{bj} in the y-direction.

The algorithm for refinement of a grid is as follows — for each grid: perform one step of the interpolating wavelet transform. If any of the generated wavelet coefficients are greater then ϵ in magnitude, split the grid in four, else keep the grid. This is done recursively, starting with grid level 0 and continuing until no further refinement is necessary, or we have reached some specified finest level. For details of the refinement strategy, as applied to SPRs, see [9]. An example of the resulting block SPR is shown in Figure 5. Note that the union of the grids cover the whole domain, while their intersection is empty. The block SPR needs to change over time as the solution changes, so we perform the above thresholding after each time step. If all four grids, that earlier were refined from the same grid on the next coarser scale, have wavelet coefficients whose magnitude is smaller then ϵ we replace the four grids with one on the next coarser scale. If we want to allow for moving features in the solution we also split grids with neighbors on a finer scale one extra level. Since we

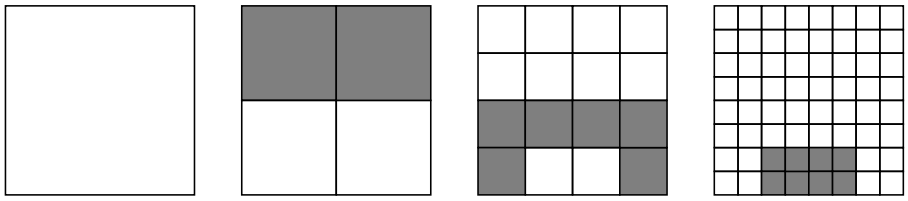

Figure 5: Example of a block SPR. Each square is a block grid with $n_{bi} \times n_{bj}$ points and the blocks included in the representation are shaded. The numbers denote the grid levels.

need neighboring points when approximating derivatives by finite differences the grids need to communicate point values. To separate computations and communication we add a layer of ghost points around each block grid. The number of points in the layer is proportional to the order of the finite difference method, e.g., one point when $p = 2$ and two points when $p = 4$. Examples of such block grids are shown in Figure 6. We then only need to update the

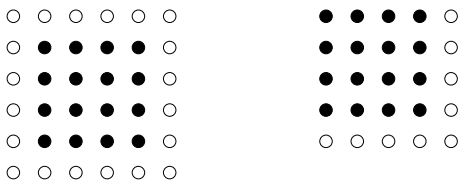

Figure 6: To the left an interior block grid and to the right a boundary block grid (upper left corner). Filled circles denote points included in the representation while unfilled circles denote ghost points. The block size is 4 in both directions ($n_{bi} = 4$ and $n_{bj} = 4$).

function values at ghost points between the stages in the Runge-Kutta solver. The advantage of this approach is the locality of reference in the computations. This is important for performance on serial computers with cache memory, but even more so if we want to use parallel computers with distributed memory.

When a point value is needed that is not in the block SPR, we use interpolating subdivision to generate the point value from coarser grids.

4 Numerical Experiments

We compare the block SPR solver with the finite difference (FD) solver for a low Reynolds number flow. The aim of the comparison is to verify the correctness of the sparse solution and to estimate the sparse solver's computational overhead. We choose a low Reynolds number (Re = 100) to be able to compute a finite difference solution in reasonable time. In what follows, if not stated otherwise,

we have used the values $d = 0.05$, $\nu = 0.05$, $\mathrm{Pr} = 0.71$, $M_\infty = 0.1$, $\mu = 1$, $\lambda = -2/3$ and $\gamma = 1.4$. The timings were made on one of the processors (EV5/300 MHz) on a Digital Alpha Server 8200. The block SPR solver was implemented in C++, while the finite difference solver was implemented in Fortran 90.

In Figure 7 the CPU time as a function of grid points in the x-direction, n_i, is shown. For the SPR method n_i is the maximum number of points on the finest level. The grid dimensions are 129×65, 257×129 and 512×256. The block sizes are $n_{bi} = 32$ and $n_{bj} = 16$. The interpolation is linear ($p = 2$). The error in the solution (as compared to an FD solution on a finer grid) for the FD and the SPR method was comparable in size when the grid size on the finest level of the SPR was equal to the FD grid size and $\epsilon = 10^{-4}$. The error was measured in maximum norm along the line $x = 1.5$ in the middle of the plate to avoid influence from the singularity at the front plate corner (as was done by Koren [12]). As could be expected, for $n_i = 129$ the FD method outperforms

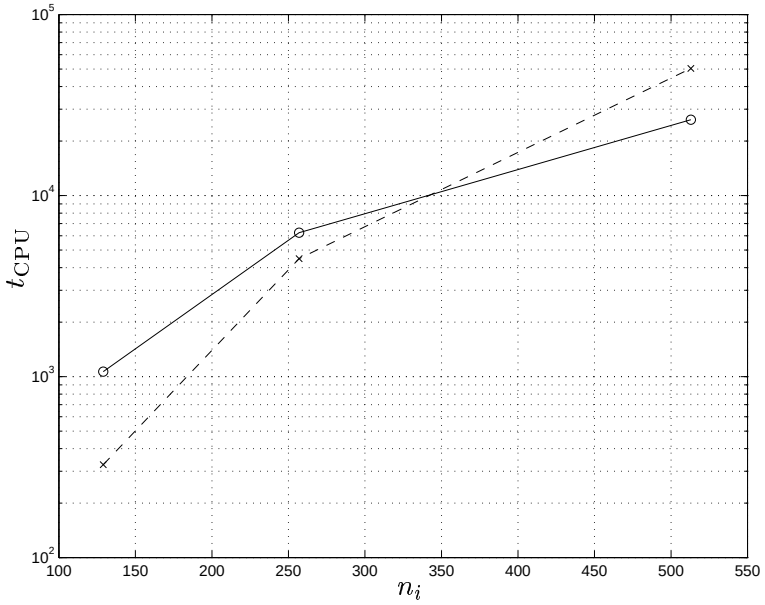

Figure 7: The CPU time as a function of grid points in the x-direction. Solid line for the sparse solver and dashed line for the finite difference solver, $\Delta t = 0.0004, 0.0002$ and 0.0001 when $p = 2$.

the block SPR method and is also faster when $n_i = 257$. This is due to the overhead associated with the sparse solver, e.g., updating the ghost points and the fact that the sparse structure makes the code more difficult for the compiler to optimize. For small enough problems the standard FD method will always be faster. But for $n_i = 513$ the block SPR method is significantly faster. So the larger the problem is the more we save in computational time by using the block SPR method. The reason that the timings are not on a straight line is probably cache effects. For small problems all data fits in the cache while larger problem sizes makes the computational speed limited by access times to main memory.

In Figure 8 the corresponding times are shown when $p = 4$. The same trend

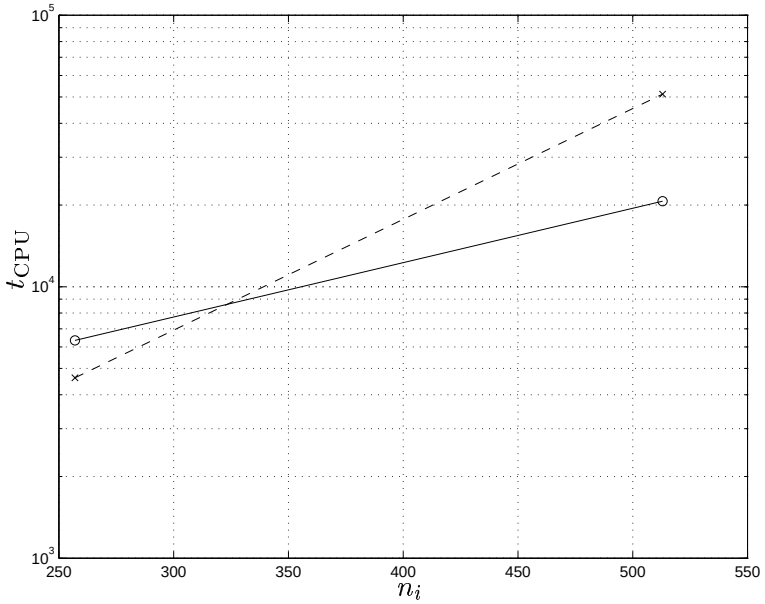

Figure 8: The CPU time as a function of grid points in the x-direction. Solid line for the sparse solver and dashed line for the finite difference solver, $\Delta t = 0.0002$ and 0.0001 when $p = 4$.

as when $p = 2$ is evident. The FD solver is slightly faster when $n_i = 257$ but for $n_i = 513$ the block SPR method is faster.

The savings in computational times are linked to the savings in memory requirements. In Table 5 we show the number of blocks in the representations at $t = 1$ for the above presented numerical examples. We see that the compression

Table 5: The number of blocks in the representations at $t = 1$ for the above presented numerical examples.

p	n_i	Number of blocks/total
2	129	16/16
2	257	55/64
2	513	109/256
4	257	37/64
4	513	46/256

is better for cubic than for linear interpolation.

In Figure 9 the number of blocks in the block SPR is shown as a function of time. The plateau after $t \approx 0.6$ is due to the fact that the refinement has reached the finest level. We note that the number of blocks increases as the boundary layer grows until we reach a maximum value at $t \approx 0.5$.

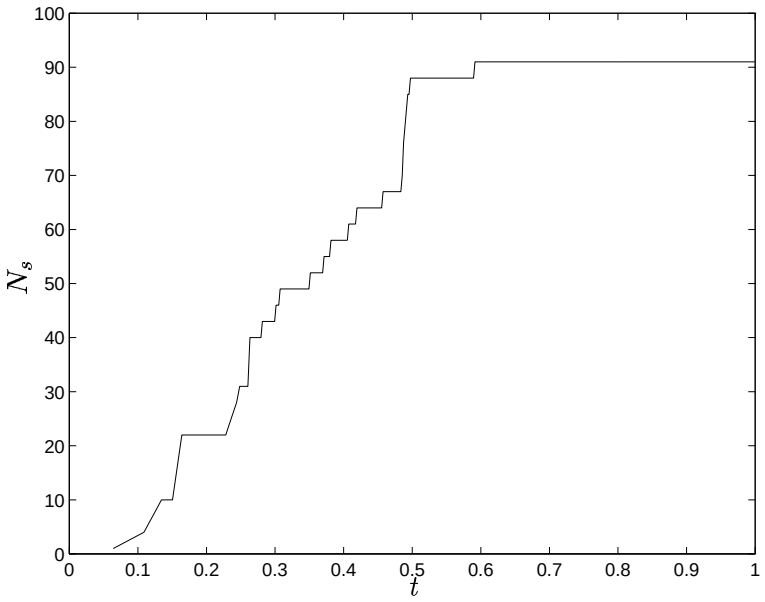

Figure 9: The number of blocks in the block SPR as a function of time. The block dimensions are $n_{bi} = 16$ and $n_{bj} = 8$ with a finest grid of 513×257. The time-step $\Delta t = 2 \cdot 10^{-4}$, $p = 2$ and the threshold $\epsilon = 10^{-2}$.

5 Conclusions

We have presented a method for adaptively solving time-dependent PDEs, the block SPR method. The adaptability is achieved by monitoring the wavelet coefficients generated by an interpolating wavelet transform. 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. The sparse representation lead to savings in the time needed to achieve a solution with a certain accuracy in maximum norm compared to a solution by finite differences. By using the block SPR method we also reduce the memory requirements. The method is most efficient when applied to problems with solutions that are smooth in most of the domain with small areas of sharp variation.

As an example of an application we solved the compressible, time dependent, Navier-Stokes equations for flow over a flat plate. Here the block SPR method automatically refines the representation in the boundary layer close to the plate. We found savings both in computational time and memory requirements compared to a solution by a finite difference scheme on a uniform fine grid.

References

[1] 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.

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

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

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

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

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

[7] Bertil Gustafsson and Arne Sundström. Incompletely parabolic problems in fluid dynamics. *SIAM Journal on Applied Mathematics*, 35(2):343–357, September 1978.

[8] Ami Harten. Adaptive multiresolution schemes for shock computations. *Journal of Computational Physics*, 115(2):319–338, 1994.

[9] Mats Holmström. Solving hyperbolic PDEs using interpolating wavelets. Technical Report 189, Dept. of Scientific Computing, Uppsala University, Box 120, S-751 04 Uppsala, Sweden, December 1996.

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

[11] 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.

[12] Barry Koren. Upwind discretization of the steady navier-stokes equations. *International Journal for numerical methods in fluids*, 11:99–117, 1990.

[13] Hermann Schlichting. *Boundary-Layer Theory*. McGraw-Hill, seventh edition, 1979.

[14] Björn Sjögreen. High order centered difference methods for the compressible navier-stokes equations. *Journal of Computational Physics*, 117(1):67–, March 1995.

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

[16] 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.

[17] 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.

[18] Johan Waldén. Filter bank methods for hyperbolic PDEs. Technical Report 185, Dept. of Scientific Computing, Uppsala University, Box 120, S-751 04 Uppsala, Sweden, 1996.