ESAIM: PROCEEDINGS, December 2011, Vol. 34, p. 184-239
E. Cancès, V. Louvet and M. Massot, Editors

TUTORIALS ON ADAPTIVE MULTIRESOLUTION FOR MESH REFINEMENT APPLIED TO FLUID DYNAMICS AND REACTIVE MEDIA PROBLEMS.

Christian TENAUD[1] and Max DUARTE[2]

Abstract. This work aims at evaluating in practical situations the capability of the mesh refinement technique based on the multiresolution adaptive method coupled with high resolution spatial and temporal approximations, to recover elementary physical mechanisms by achieving gains in both CPU time and memory use compared to single grid computations. We first present a summary of the multiresolution procedure. We then describe MR algorithms. Finally, the evaluation of the method is presented on several well known numerical test cases in 1D and multi-D configurations.

Résumé. Ce travail a pour objectif d'évaluer dans des situations pratiques, la capacité d'une méthode de raffinement automatique de maillage, basée sur une méthode de multirésolution adaptative et couplée à des approximations spatiales et temporelles de haute résolution, à reproduire les mécanismes physiques fondamentaux en assurant des gains substantiels, par rapport à des simulations sur un maillage unique, à la fois sur les temps CPU et sur le stockage mémoire. Nous présentons tout d'abord un résumé de la procédure de multirésolution. Nous décrivons ensuite l'algorithme de multirésolution (MR). L'évaluation de la méthode est enfin présentée à travers l'application à des cas tests bien répertoriés, dans des configurations 1D et multidimensionnelles.

INTRODUCTION

During the last decades, computer power has considerably increased. For fluid dynamic problems, this has motivated developments of high-order numerical methods to increase the prediction capability of numerical codes. Direct Numerical Simulation (DNS) rapidly became a powerful tool for performing fine analysis of flow dynamics [26] since all the length-scales are represented in the simulation. On the one hand, the quality of the results mainly depends on the capability of the numerical scheme to capture the governing dynamical process. In this context, important works have been conducted on numerical approximations which can both represent small scale structures with the minimum of numerical dissipation and capture the compressible features responsible for discontinuity creation with robustness (we refer to [2,15,22,23,29,30,32–34,37], for more details). Besides the numerical scheme on the other hand, the quality of solutions also depends on the capability of the computational grid to capture all the length-scales involved in the dynamical mechanisms. In that sense, Adaptive techniques

[1] LIMSI - CNRS UPR 3251, Campus Universitaire, Bât. 508, rue John Von Neumann, 91403 Orsay Cedex, France.
[2] Laboratoire EM2C - CNRS UPR 288, École Centrale Paris, Grande Voie des Vignes, 92295 Châtenay-Malabry Cedex, France.

for problems exhibiting locally steep gradients or shock-like structures have been developed since the end of the 1970s. Pioneering works on adaptive methods like Multi-Level Adaptive Techniques (MLAT) (Brandt [10]) or Adaptive Mesh Refinement (AMR) (Berger *et al* [4–6]) demonstrated that a set of locally refined grids can be used in regions where steep gradients of high truncation errors are found. However, the data compression rate is based on a criterion that neither refers to the quality of the solution nor to an error norm of the solution. To overcome this difficulty, adaptive multiresolution methods, based on Harten's pioneering work [21], have been developed for 1D and 2D hyperbolic conservation laws (Cohen *et al* [14], Müller *et al* [17]). They have then been extended to 3D parabolic problems (Roussel *et al* [31]). First simulations of 3D supersonic flows in the laminar regime using adaptive multiresolution methods were performed by Bramkamp *et al* [8,9], with separate RK/ENO time-space discretizations. It has been shown in these papers that a high compression rate can be reached for solutions with inhomogeneous regularity. For an overview on adaptive multiresolution techniques, we refer to the books of Cohen [11] and Müller [27].

This tutorial aims at evaluating in practical situations the capability of the multiresolution adaptive technique coupled with high resolution spatial and temporal approximations to recover elementary physical mechanisms by achieving gains in both CPU time and memory use compared to single grid computations. The tutorial is then organized as follows. We first present in section 1, a summary of the multiresolution procedure. We then describe MR algorithms in section 2. Finally, the evaluation of the method is presented in section 3 on several well known numerical test cases in 1D and multi-D configurations.

1. MULTIRESOLUTION PROCEDURE

Multiscale approximation techniques are powerful tools for numerical analysis of Partial Differential Equations (PDE). Through thresholding coefficients of a discretized solution, Multiscale basis allow to produce a discretization at a coarser grid-level in regions where the function is smooth enough while details are retained at finer grid-levels when the function is singular. For PDEs resolution in Computational Fluid Dynamics (CFD) domain, the idea sought after is to employ a multiscale technique to approximate solutions with a minimum of memory use and a lower CPU time consumption.

In this section, we describe adaptive multiresolution algorithms for numerical solutions of the initial value problem for conservation laws. We consider the following set of equations:

$$\begin{cases} \dfrac{\partial \mathbf{w}}{\partial t} + \dfrac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{x}} = \mathbf{S}(\mathbf{w}) & \text{in } \Omega, \\ \mathbf{w}(\mathbf{x},0) = \mathbf{w}_0(\mathbf{x}), & \\ \mathbf{w}(\mathbf{x},t) = \mathbf{g}(\mathbf{x},t) & \text{on } \partial\Omega. \end{cases} \tag{1}$$

Here Ω is the computational domain, $\mathbf{w}$ is the vector of p components of conservative variables, $\mathbf{f}$ and $\mathbf{S}$ are the flux vector and the source terms, respectively, both depending on $\mathbf{w}$ and possibly on $\nabla\mathbf{w}$. Let $x_j^0 = j.h_0,\ 0 \leq j \leq N_0$ be a uniform partition of the computational domain Ω into N_0 intervals of size h_0. We then introduce a control volume (V_j^0) and define a cell as: $V_j^0 := \left[x_{j-1}^0, x_j^0\right],\ 1 \leq j \leq N_0$. Solving equation 1 implies looking for successive approximations ($\left(v_j^0\right)^n$) to the average value of the solution $\mathbf{w}(\mathbf{x},t)$ in cells $[x_{j-1}, x_j],\ 1 \leq j \leq N_0$, at time $n\ \delta t$:

$$\left(v_j^0\right)^n = \frac{1}{|V_j^0|} \int_{V_j^0} \mathbf{w}(\mathbf{x}, n\ \delta t)\ d\mathbf{x}, \tag{2}$$

where n is the number of time-steps δt and $|V_j^0| = \int_{V_j^0} d\mathbf{x}$ is the measure of the control volume.

Since the finite volume computation is based on cell-average values of the conservative variables, the cell-average multiresolution analysis [21] is mainly employed. In the context of adaptive mesh refinement, we here represent data on a set of nested dyadic grids by using the principle of the multiresolution analysis.

1.1. **Hierarchy of nested grids**

We consider a set of nested dyadic grids. For simplicity, we only consider here embedded Cartesian grids. Information about structured nested grids in general coordinate system can however be found in [27]. Nested grid techniques have also been developed for unstructured grids [11, 28], where triangle or tetrahedral elements are considered.

We note $l = 0, 1, ..., L$ the grid level from the coarsest ($l = 0$) to the finest ($l = L$). At each grid level (l), we consider a partition of cells V_j^l of the computational domain Ω. V_j^l is considered as a control volume in the Finite-Volume terminology. The hierarchy of nested grids is based on a tree data structure which is used to encode the solution by means of the multiresolution analysis. Examples of embedded grids on a tree data structure are presented in figures (1, 2, and 3). The *nodes* are elements of the tree, noted Λ (corresponding to

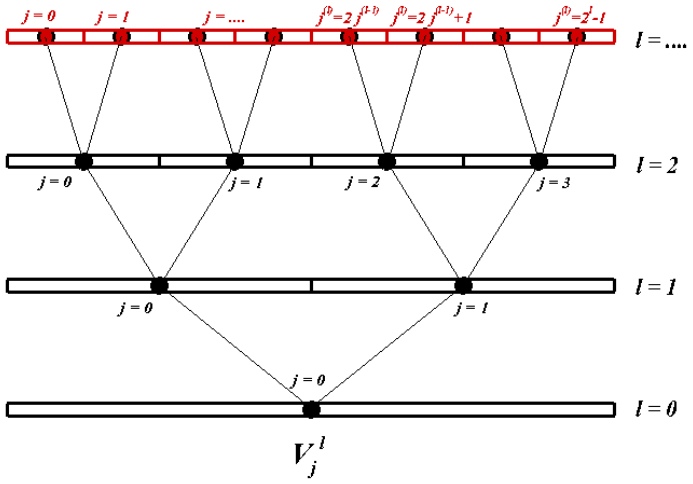


FIGURE 1. Dyadic tree data structure in 1D as a partition of the computational domain Ω.

black or red dots on Fig. 1), defined by the couple (j, l) corresponding to the position index (j) and the grid level (l). A control volume (V_j^l) is then associated to each node of the tree. The *root* (V_0^0) denotes the basis of the tree. The *leaves* denote upper elements (nodes at the highest grid level), which have been enhanced in red in figure 1. In N_{dim} dimensions, a *parent*-cell at a level l (V_j^l, in 1-D) has always $2^{N_{dim}}$ *children*-cells at the level $l + 1$ (V_{2j}^{l+1} and V_{2j+1}^{l+1} in 1-D). If a *node* $(j, l) \in \Lambda$ then the *parent*-cell $(j/2, l - 1) \in \Lambda$ and the *children*-cells $(2j, l + 1)$ and $(2j + 1, l + 1) \in \Lambda$.

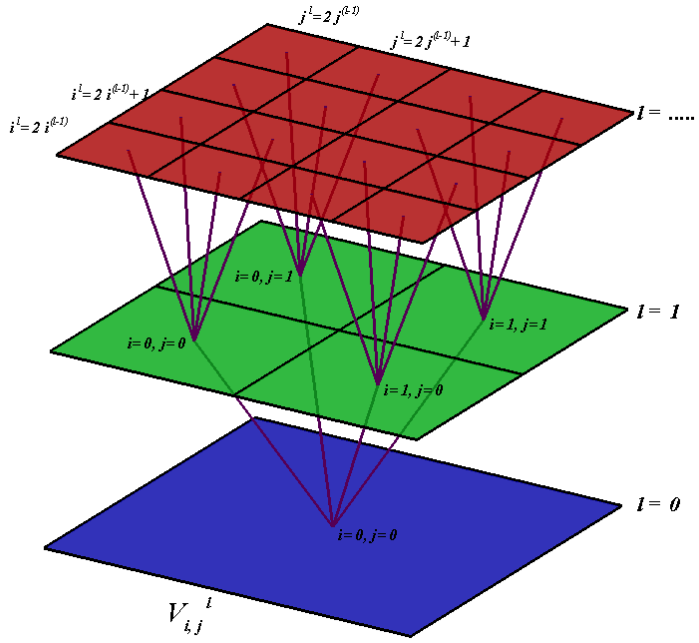


FIGURE 2. Example of a 2D dyadic tree data structure as a partition of the computational domain Ω.

The cell partition is dense in Ω, meaning that:

$$\Omega = \bigcup_{j \in I_l} V_j^l \quad \text{with} \quad \left| V_j^l \bigcap V_k^l \right| = 0 \quad \text{for } j \neq k;\ j,k \in I_l, \tag{3}$$

where I_l is the cell index set of Ω at grid level l. Moreover, we assume in a refinement process that between two consecutive grid-levels:

$$V_j^l = \bigcup_{p \in \mathcal{C}_j^l} V_p^{l+1}, \tag{4}$$

where $\mathcal{C}_j^l$ is the index set of the chidren-cells of V_j^l.

1.2. **Mean value multiresolution: Finite-Volume approach**

The mean-value multiresolution is based on the cell-average value on each control volume at level l, expressed as follow, for instance in 3-dimensions:

$$v_{i,j,k}^l = \frac{1}{|V_{i,j,k}^l|} \int_{V_{i,j,k}} \mathbf{w}(\mathbf{x},t)\, d\mathbf{x}, \tag{5}$$

where $V_{i,j,k}^l := [2^{-l}i, 2^{-l}(i+1)] \times [2^{-l}j, 2^{-l}(j+1)] \times [2^{-l}k, 2^{-l}(k+1)]$ with $i \in \mathbb{Z}$, $j \in \mathbb{Z}$, $k \in \mathbb{Z}$ and $|V_{i,j,k}^l| = \int_{V_{i,j,k}^l} d\mathbf{x}$ is the measure of the control volume.

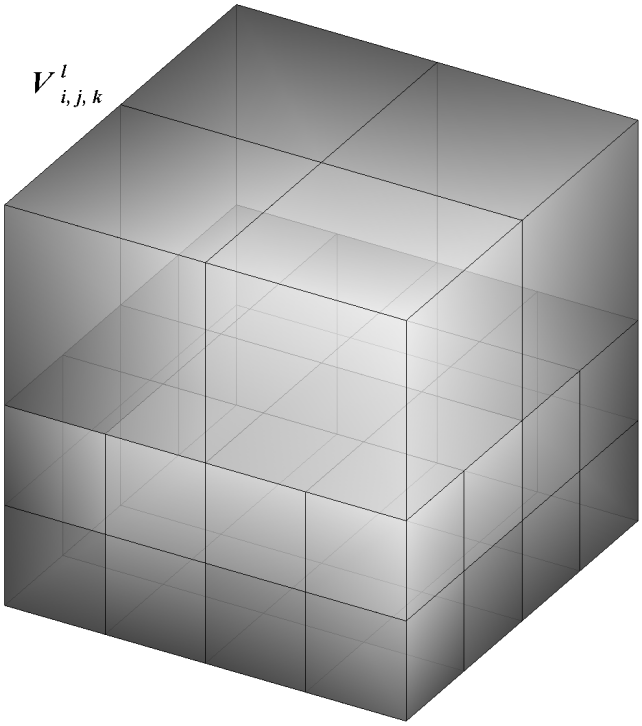


FIGURE 3. Example of a 3D dyadic tree data structure as a partition of the computational domain Ω.

If we introduce a dual scaling function ($\widetilde{\varphi}$) based on the so-called *box function*:

$$\widetilde{\varphi}_j^l = \frac{1}{|V_j^l|}\ \chi_{V_j^l}(x) \ \text{ with } \chi_{V_j^l} = \begin{cases} 1 & \text{if } \ x \in V_j^l; \\ 0. & \text{elsewhere;} \end{cases} \tag{6}$$

the cell-average value of a scalar integrable function $w \in L^1(\Omega)$ can be interpreted as the inner product:

$$v_j^l = < w, \widetilde{\varphi}_j^l >_\Omega, \tag{7}$$

where the inner product is defined by $< u, v >_\Omega := \int_\Omega uv\ dx$.

The scaling function is normalized in L^1: following (6), $||\widetilde{\varphi}^l||_{L^1} = 1$.

Relationships between cell-average values at different grid-levels (l) are based on the multiresolution analysis. Therefore, we define two operators, the *projection* and *prediction* operators, that allow us to represent a function by its cell-average values on a coarse grid (at grid level $l-1$), knowing the function cell-average values on a finer grid (at grid-level l), and in the opposite direction, to predict cell-average values of the function on a fine grid (at grid-level l) from the function cell-average values known on a coarser one (at level $l-1$).

1.2.1. *Projection operator*

The **projection** operator (noted $\mathbf{P}_{l+1\to l}$) allows one to compute the cell-average value of the solution at a grid node at level l from cell-average values of the solution known on children-nodes at grid level $l+1$. As far as grids are nested, the projection operator is *exact* and *unique* [11], given that cell-average values at two successive grid levels are related by:

$$\mathbf{P}_{l+1\to l}: \;\; v_j^l = \frac{1}{|V_j^l|}\sum_{p\in\mathcal{C}_j^l} |V_p^{l+1}|\; v_p^{l+1}; \tag{8}$$

where $\mathcal{C}_j^l$ denotes the index set of the $2^{N_{dim}}$ *children*-cells at grid level $(l+1)$ of the current cell V_j^l. For 1-dimension, $\mathcal{C}_j^l \in [2j, 2j+1]$. In such way, if the solution on leaves (at grid level l) is known, cell-average values can be calculated from grid level (l) down to the root-cell $(l=0)$.

1.2.2. *Prediction operator*

The **prediction** operator (noted $\mathbf{P}_{l\to l+1}$) maps $\mathbf{v}^l$ to an approximate value $\hat{\mathbf{v}}^{l+1}$ of $\mathbf{v}^{l+1}$. In contrast with the projection operator, there is an infinite number of choices for the definition of $P_{l\to l+1}$. Nevertheless, in order to be applicable and tractable in a graded tree structure, the prediction needs to be:

- *local*; this means that the interpolation stencil must contain the *parent*-cell and its nearest neighbors in each direction [11, 28].
- *consistent with the projection* operator, *i.e.* $\mathbf{P}_{l+1\to l}\circ\mathbf{P}_{l\to l+1} = Id$. In other words, the approximate value $\hat{\mathbf{v}}^{l+1}$ must be conservative with respect to the average values on the coarser grid, $\mathbf{v}^l$:

$$|V_j^l|\; v_j^l = \sum_{p\in\mathcal{C}_j^l} |V_p^{l+1}|\; \hat{v}_p^{l+1} \tag{9}$$

Let us notice that the linear property of the prediction operator is not necessarily required. Anyway, for simplicity of the numerical analysis, linear polynomial interpolation are often used to predict the approximated value $\hat{v}_p^{l+1}$. Information on non-linear operator can be found in Aràndiga *et al.* [3].

As the approximation stencil contains the *parent*-cell and its nearest neighbors, we can consider centered linear polynomial interpolations of order $o = 2\;s+1$ of accuracy, based on the s-nearest neighboring cells to approximate $\hat{v}_p^{l+1}$:

$$\mathbf{P}_{l\to l+1}: \begin{cases} \hat{v}_{2j}^{l+1} = v_j^l + \displaystyle\sum_{q=1}^{s} \xi_q \left(v_{j+q}^l - v_{j-q}^l\right), \\ \hat{v}_{2j+1}^{l+1} = v_j^l - \displaystyle\sum_{q=1}^{s} \xi_q \left(v_{j+q}^l - v_{j-q}^l\right), \end{cases} \tag{10}$$

where ξ_q are coefficients of the linear polynomial interpolation of accuracy order $o = 2\;s+1$, given in table 1 up to $s=5$. Let us mention that when $s=0$, the classical Haar basis is recovered. For $s>0$, considering the *child*-cell V_j^l, the stencil is:

$$R_j^l = \left\{ \frac{j}{2} + q,\; |q| \le s;\; (l-1) \right\}. \tag{11}$$

order (o)	s	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5
1	0	0	0	0	0	0
3	1	$\frac{-1}{8}$	0	0	0	0
5	2	$\frac{-22}{128}$	$\frac{3}{128}$	0	0	0
7	3	$\frac{-201}{1024}$	$\frac{11}{256}$	$\frac{-5}{1024}$	0	0
9	4	$\frac{-3461}{16384}$	$\frac{949}{16384}$	$\frac{-185}{16384}$	$\frac{35}{32768}$	0
11	5	$\frac{-29011}{131072}$	$\frac{569}{8192}$	$\frac{-4661}{262144}$	$\frac{49}{16384}$	$\frac{-63}{262144}$

TABLE 1. Coefficients of centered linear polynomial interpolations [20]

As far as Cartesian mesh is used, extension to multidimensional polynomial interpolations is easily obtained by a tensor product of the 1-D operator [7, 31]. If we define the polynomial interpolation Q^s as:

$$Q^s\left(j; v^l\right) = \sum_{q=1}^{s} \xi_q \left(v^l_{j+q} - v^l_{j-q}\right), \tag{12}$$

the tensor product of the polynomial interpolation in two dimensions, proposed by Bihari and Harten [7] reads:

$$\hat{v}^{l+1}_{2j+p,2k+q} = v^l_{j,k} + (-1)^p\ Q^s(j; \mathbf{v}^l_{.,k}) + (-1)^q\ Q^s(k; \mathbf{v}^l_{j,.}) - (-1)^{(p+q)}\ Q^s_2(j,k; \mathbf{v}^l), \tag{13}$$

with p and q equal to either 0 or 1 depending on the child-cell considered. The polynomial Q^s (12) is used in both directions and the operator Q^s_2, derived from a tensor product, reads:

$$Q^s_2\left(j,k; \mathbf{v}^l\right) = \sum_{a=1}^{s} \xi_a \sum_{b=1}^{s} \xi_b \left(v^l_{j+a,k+b} - v^l_{j-a,k+b} - v^l_{j+a,k-b} + v^l_{j-a,k-b}\right), \tag{14}$$

In the same way, polynomial interpolations are also extended to 3-dimensions by introducing a new operator Q^s_3, also derived from a tensor product, that reads:

$$\begin{aligned} Q^s_3\left(i,j,k; \mathbf{v}^l\right) = \sum_{a=1}^{s} \xi_a \sum_{b=1}^{s} \xi_b \sum_{c=1}^{s} \xi_c \quad & \left(v^l_{i+a,j+b,k+c} - v^l_{i-a,j+b,k+c} - v^l_{i+a,j-b,k+c}\right. \\ & -v^l_{i+a,j+b,k-c} + v^l_{i-a,j-b,k+c} + v^l_{i-a,j+b,k-c} \\ & \left. +v^l_{i+a,j-b,k-c} - v^l_{i-a,j-b,k-c}\ \right), \end{aligned} \tag{15}$$

Therefore, the 3D- polynomial interpolation reads:

$$\begin{aligned} \hat{v}^{l+1}_{2i+p,2j+q,2k+r} = \quad & v^l_{i,j,k} + (-1)^p\ Q^s(i; \mathbf{v}^l_{.,j,k}) + (-1)^q\ Q^s(j; \mathbf{v}^l_{i,.,k}) + (-1)^r\ Q^s(k; \mathbf{v}^l_{i,j,.}) \\ & -(-1)^{(p+q)}\ Q^s_2(i,j; \mathbf{v}^l_{.,.,k}) - (-1)^{(p+r)}\ Q^s_2(i,k; \mathbf{v}^l_{.,j,.}) \\ & -(-1)^{(q+r)}\ Q^s_2(j,k; \mathbf{v}^l_{i,.,.}) + (-1)^{(p+q+r)}\ Q^s_3(i,j,k; \mathbf{v}^l), \end{aligned} \tag{16}$$

with p, q, and r equal to either 0 or 1 depending on the child-cell considered.

When dealing with unstructured meshes based either on triangle or tetrahedral elements, it is much more delicate to derive multidimensional polynomial interpolations from the 1D-operator (10). Regarding unstructured meshes, detailed information can be found in Cohen *et al.* [11] and M. Postel [28].

1.2.3. *Multiresolution transform*

One can introduce a *prediction error* at a grid level l which is estimated by evaluating *"details"* (d_j^l) defined as the difference between the numerical solution v_j^l and interpolated values $\hat{v}_j^l$:

$$d_j^l = v_j^l - \hat{v}_j^l. \tag{17}$$

Following eq. (6 and 7), *details* can be interpreted as the inner product between the solution ($\mathbf{w}$) and a *dual wavelet*, noted $\widetilde{\psi}$:

$$d_j^l =< \mathbf{w}, \widetilde{\psi}^l > . \tag{18}$$

Following definitions of both the *detail* (17) and the cell average value in terms of the scaling function (7),

$$< \mathbf{w}, \widetilde{\psi}^l >=< \mathbf{w}, \widetilde{\varphi}^l > - \sum_q c_q^{l-1} < \mathbf{w}, \widetilde{\varphi}^{l-1} >, \tag{19}$$

the *dual wavelet* reads:

$$\widetilde{\psi}^l := \widetilde{\varphi}^l - \sum_q c_q^{l-1} \widetilde{\varphi}^{l-1}. \tag{20}$$

If we assume that prediction coefficients c_q^{l-1} are uniformly estimated, the *dual wavelet* is normalized in L^1: $||\widetilde{\psi}||_{L^1} \leq C$.

Thanks to the consistency assumption (8, 9, and 10), the sum of *details* (d_j^l) on *children*-cells of a *parent*-cell is equal to zero [21]:

$$\sum_{p \in \mathcal{C}_j^l} |V_p^l| \; d_p^l = 0. \tag{21}$$

Therefore, in N_{dim} dimensions, the knowledge of the $2^{N_{dim}}$ *children* cell-averages of a given *parent*-cell is equivalent to the knowledge of the *parent* cell-average and $2^{N_{dim}} - 1$ *details*.

Following A. Harten [21], if the function w has $q-1$ continuous derivatives and a jump discontinuity at its q-th derivative, then the *detail* behaves as:

$$d_j^l(v^L) \sim \begin{cases} (\delta x_j)^q \; [w^{(q)}] & \text{for} \quad 0 \leq q \leq o; \\ (\delta x_j)^o \; w^{(o)} & \text{for} \quad q > o; \end{cases} \tag{22}$$

where o is the accuracy order and [.] is the jump at the discontinuity. The main property required in the MR process is that the prediction approximation recovers o-th order of accuracy which is equivalent to say that it is exact for $(o-1)$-th order polynomials. In such way, the *details* recover null values for smooth solutions with locally bounded o-th order derivatives [12]: *i.e.* $\forall \; \mathbf{u} \in \prod_{o-1}$ ($(o-1)$-th order polynomials), then $u_j = \hat{u}_j$ and

$$< \mathbf{u}, \widetilde{\psi}^l >= \mathbf{d}^l = 0. \tag{23}$$

The $o-1$ first moments of the *dual wavelet* must then be null [11, 12, 28]. Moreover, as far as the solution is smooth, it was shown in [9, 13, 28] that *details* decay with a rate at least of 2^{-l}:

$$\left|d_j^l\right| \leq C \; 2^{-l} \left|\mathbf{v}'\right|_{L^\infty(V_j^l)}. \tag{24}$$

Therefore for smooth solutions, the higher the grid level, the smaller the *details*. On the other hand, *details* recover significantly high values in regions where singularities of the solution occur.

Let us denote by $\mathbf{D}^l$ the vector of all details at a grid level l:

$$\mathbf{D}^l = \left\{ d_j^l,\ 0 \leq j \leq N_l \right\},$$

with $N_l = (2^{N_{dim}} - 1)\ 2^{N_{dim}(l-1)}$ for dyadic nested grids, we can observe that, following eq. (17 and 21), the knowledge of $\left(\mathbf{v}^l,\ \mathbf{D}^{l+1}\right)$ is equivalent to the knowledge of $\mathbf{v}^{(l+1)}$:

$$\mathbf{v}^{(l+1)} \leftrightarrow \left(\mathbf{v}^l,\ \mathbf{D}^{l+1}\right). \tag{25}$$

Note that there is a one to one transformation between the two sets which have the same number of elements: $N_{l+1} = 2^{N_{dim}.(l+1)}$ elements at grid level $(l+1)$ decomposed in $2^{N_{dim}.l}$ values of $\mathbf{v}^l$ plus $(2^{N_{dim}} - 1)\ 2^{N_{dim}.l}$ *details*. Recursively on all L grid levels, one gets the so-called *multiresolution transform* [21] that maps the vector of the solution on the finest grid $(\mathbf{v}^L)$ to the solution on the root-cell plus all vectors of *details* from grid level $l = 1$ to the highest level (L):

$$\mathcal{M} : \mathbf{v}^L \longmapsto \left(\mathbf{v}^0, \mathbf{D}^1, \ldots, \mathbf{D}^L\right) = \mathbf{M}^L. \tag{26}$$

This multiresolution transform $(\mathcal{M})$ is a one-to-one transformation between $\mathbf{v}^L$ and $\mathbf{M}^L$, and in the case where the **prediction** operator $\mathbf{P}_{l \to l+1}$ is linear, it can be seen as a basis change, keeping the number of degrees of freedom unchanged:

$$\mathbf{v}^L = \mathcal{M}^{-1}\ \mathbf{M}^L\ ;\ \mathbf{M}^L = \mathcal{M}\ \mathbf{v}^L \tag{27}$$

Encoding the solution known by its cell-average value on the finest grid is achieved by using the following Algorithm 1.

In the opposite way, *decoding* the solution known by its cell-average values on the coarsest grid and the set of *details* is obtained through the use of the following Algorithm 2.

The new data format is however more convenient for data compression because *detail* magnitudes are zero when solutions have locally bounded o-th order derivatives and decay with the grid level (2^{-l} for smooth solutions). Nevertheless, to be competitive with computations on a unique fine grid at the highest level, the multiresolution transform should not increase the computational complexity related to the number of floating point operations required by the algorithm. In order to decrease the computational complexity, one of the solutions is to adapt locally the mesh, considering the behavior of the solution, which should yield lower CPU time and memory requirements.

1.3. **Thresholding and compression**

Indicator functions are needed to locate regions where the solution requires a mesh refinement for capturing small scale structures. Here, assuming that the multiresolution basis 26 is stable [11], this indicator is based on the *details* because they decay with the grid level for smooth solutions and recover sufficiently large values in singular regions. The idea is to retain only the *details* whose values are greater than a threshold parameter (ε_l) on each grid level (l) and set the others to zero. In practice, when we deal with systems of equations, the measure of the *details* is computed through the L^1-norm of the vector $\mathbf{d}^l$, which seems natural for conservation law problems. The threshold operation $(\mathcal{T})$ is then obtained by taking into account a measure of *details* scaled

Algorithm 1 Encoding cell-average solution $\mathbf{M}^L = \mathcal{M}\ \mathbf{v}^L$

1: Knowing the average values (v_j^L) of the solution on the finest grid control volumes (V_j^L);
2: **for** $l = L-1$ down to 0 **do**
3: **for** $j \in I_l$ **do**
4: Calculation of cell averages v_j^l at grid level l by using the projection operator $\mathbf{P}_{l+1\to l}$ (8):

$$v_j^l = \frac{1}{|V_j^l|} \sum_{p=1}^{2^{N_{dim}}} |V_p^{l+1}|\ v_p^{l+1}; \tag{28}$$

where p denotes indexes of the $2^{N_{dim}}$ *children*-cells at grid level $(l+1)$ of the current cell V_j^l. In fact, $p \in [2j, 2j+1]$ in each direction.
5: Calculation of predicted values by using linear polynomial interpolations (12, 14 or 15):

$$\hat{\mathbf{v}}^{l+1} = \mathbf{P}_{l\to l+1}\ \mathbf{v}^l$$

6: Calculation of *details* between $\mathbf{v}^{l+1}$ and $\hat{\mathbf{v}}^{l+1}$ by using (17):

$$d_{2j}^{l+1} = v_{2j}^{l+1} - \hat{v}_{2j}^{l+1}$$

Store the set of *details* in the vector $\mathbf{D}^{l+1} = \left\{d_p^{l+1},\ 0 \le p \le (2^{N_{dim}}-1)\ 2^{N_{dim}.l}\right\}$. If needed, the last *detail* can be computed by consistency relations eq. (21).
7: Encode the solution by replacing the cell-average values on the fine grid, $\mathbf{v}^{l+1}$, by the cell-average values on the coarser grid plus the *details*, $(\mathbf{v}^l, \mathbf{D}^{l+1})$;
8: **end for**
9: **end for**
10: Considering a complete tree data structure having $\dfrac{2^{(L+1).N_{dim}}-1}{2^{N_{dim}}-1}$ *nodes* among its $2^{L.N_{dim}}$ *leaves*, the cell-average function $\mathbf{v}$ is now encoded by using the new data format with one cell-average value on the *root*-cell and $2^{L.N_{dim}}-1$ *details*: $\mathbf{M}^L = \left(\mathbf{v}^0, \mathbf{D}^1, \ldots, \mathbf{D}^L\right)$.

by global maximum, following:

$$\mathcal{T}_{\varepsilon_l}(\mathbf{d}) = \begin{cases} \text{if} & \dfrac{\left|d_j^l\right|_{L_1}}{\max_j |d_j^l|} < \varepsilon_l \Longrightarrow d_j^l = 0; \\ \text{otherwise} & d_j^l \in \mathbf{D}^l. \end{cases} \tag{32}$$

Therefore, the computational mesh is formed by *leaves* such that the normalized L^1-norm of their *details* is greater than ε_l. In order to obtain data compression, *leaves* with their *details* set to zero are discarded from the tree structure. Therefore, we define the index and grid level set (Λ_ε) of *nodes* that belong to the compressed tree data structure:

$$\Lambda_{\varepsilon_l} = \left\{(j,l)\ \text{ s.t. }\ |d_j^l| \ge \varepsilon_l\right\} \tag{33}$$

We can then define the multiresolution (MR) approximation operator $(\mathcal{A}_{\Lambda_{\varepsilon_l}})$ that maps the function (the finite-volume solution for conservation law) on the unique finest grid $(\mathbf{v}^L)$ to the function evaluated on the

Algorithm 2 Decoding cell-average solution $\mathbf{v}^L = \mathcal{M}^{-1}\ \mathbf{M}^L$

We here assume that the solution $\mathbf{w}$ is known by its cell-average values (v_j^0) on the control volumes (V_j^0) of the coarsest grid and the set of *details* $\left(\mathbf{D}^1, \ldots, \mathbf{D}^L\right)$;
for $l = 0$ up to $L-1$ **do**
  for $j \in I_l$ **do**
    Knowing the cell-average value, v_j^l, of the current cell V_j^l and the local *details*, $d_p^{l+1},\ p \in \mathcal{C}_j^l$, where $\mathcal{C}_j^l$ is the index set of the $2^{N_{dim}}$ *children*-cells of V_j^l;
    Interpolate $\hat{v}_p^{l+1},\ p \in \mathcal{C}_j^l$ by using the centered linear polynomial interpolations $\mathbf{P}_{l \to l+1}$ (10, 13 or 16);
    Calculate cell-average values $v_p^{l+1},\ p \in \mathcal{C}_j^l$ at grid level $l+1$ by (17):

$$v_p^{l+1} = \hat{v}_p^{l+1} + d_p^{l+1},\ p \in \mathcal{C}_j^l; \tag{29}$$

    In fact, $2^{N_{dim}} - 1$ values of the cell-average is computed through eq.(29). The last value is computed by consistency relations eq. (9, 21). For instance, in one dimension, we have:

$$v_{2k}^{l+1} = \hat{v}_{2k}^{l+1} + d_{2k}^{l+1}; \tag{30}$$

$$v_{2k+1}^{l+1} = \frac{|V_j^l|}{|V_{2k+1}^{l+1}|}\ v_j^l - v_{2k}^{l+1}; \tag{31}$$

    Decode the solution replacing $(\mathbf{v}^l,\ \mathbf{D}^{l+1})$ by $\mathbf{v}^{l+1}$;
  end for
end for
The solution $\mathbf{w}$ is now known by its $2^{L.N_{dim}}$ cell-average values $\mathbf{v}_j^L,\ j \in I_L$ on the finest grid.

refined nested grids:

$$\mathcal{A}_{\Lambda_{\varepsilon_l}} := \mathcal{M}^{-1}\ \mathcal{T}_{\varepsilon_l}\ \mathcal{M} \tag{34}$$

Let us remark that the MR operator $\mathcal{A}_{\Lambda_{\varepsilon_l}}$ is non linear since it depends on the solution $\mathbf{v}^L$ through the threshold procedure (33). We can then compute the *perturbation error* defined as the difference between the finite-volume solution evaluated on the finest grid and the solution obtained on the refined grid by using the multiresolution algorithm. This *perturbation error* intrinsic to the MR algorithm, recovers the following form [11, 28]:

$$\|\mathbf{v}^L - \mathcal{A}_{\Lambda_{\varepsilon_l}} \mathbf{v}^L\| = C \sum_{|\mathbf{d}^l| < \varepsilon_l} |\mathbf{d}^l|\ 2^{-N_{dim} l} \tag{35}$$

1.4. **Building of a graded tree**

In order to be *graded*, the tree must verify that, for each *leaf* at a level l, the prediction operator $(P_{l-1 \to l})$ can be evaluated. Therefore, adjacent cells of level at least equal to $l-1$ must be added in each direction (see Figure 4 for $s = 1$), and also diagonally for multi-dimension configurations. The number of adjacent cells that must be added depends on the number (s) of the nearest neighbor cells of the polynomial interpolation.

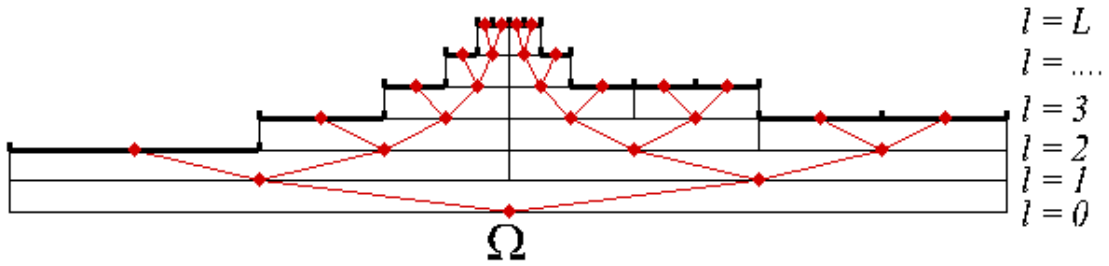


FIGURE 4. Graded tree data structure in 1D with a stencil width $s = 1$. Leaves are enhanced by bold lines.

We say that $\widetilde{\Lambda}$ is a graded tree if, for every *node*-cell (j, l) that belongs to $\widetilde{\Lambda}$, the associated prediction stencil R_j^l, defined in (11), is also in $\widetilde{\Lambda}$:

$$(j,l) \in \widetilde{\Lambda} \Rightarrow (\frac{j}{2} + q, l-1) \in \widetilde{\Lambda},\ |q| \leq s,\ q \in \mathbb{Z}. \tag{36}$$

Let us notice that, for the Haar basis ($s = 0$), a tree is always graded. Let us also remark that, for $s \neq 0$, we at most have a difference of one grid level between consecutive cells. Algorithm 3 gives the procedure to ensure that the tree is graded.

1.5. **Harten's heuristic hypothesis**

The threshold parameter (ε_l) has a crucial influence on the error introduced by the multiresolution procedure [11], called *the perturbation error*. If one wants that the *perturbation error* (for both the L^1 and L^∞ norms) recovers the same magnitude (ε) for all grid levels, the threshold parameter (ε_l) should be defined as:

$$\varepsilon_l = 2^{N_{ndim} \cdot (l-L)}\ \varepsilon. \tag{38}$$

This has first been introduced by Harten [21] as a heuristic hypothesis which has been corroborated by theoretical studies [11, 14, 31]. The *perturbation error* in the mentioned norms is then bounded. It scales linearly with the threshold parameter ε:

$$\|\mathbf{v}^L - \mathcal{A}_{\Lambda_{\varepsilon_l}} \mathbf{v}^L\| \leq C\varepsilon, \tag{39}$$

where the constant C is independent of the number of grid-levels. In this way, the value of ε has a leading role in the accuracy and efficiency of the grid refinement. We will show later in the tutorial (see § 3) the influence of the threshold parameter, ε, on the accuracy of the solution and on gains in both the memory use and the CPU time. The procedure is presented in Algorithm 4.

1.5.1. *Foreseeing refinement*

As far as hyperbolic conservation laws are concerned, we have to consider that the solution propagates with a finite speed. Moreover, the compressible feature of these laws might lead to discontinuity formations after a finite time, even if the solution is initially smooth.

Algorithm 3 Building a graded tree $\widetilde{\Lambda}$

We here assume that the solution $\mathbf{w}$ is known by its cell-average values (v_j^0) on the control volumes (V_j^0) of the coarsest grid and the set of *details* $\left(\mathbf{D}^1,\ldots,\mathbf{D}^L\right)$;
First step: thresholding *details* $\mathcal{T}_{\varepsilon_l}(\mathbf{d})$.
we initiate a binary flag, $\widehat{t}_j^l$ that marks if a *node*-cell (j,l) belongs or not to the tree $\widetilde{\Lambda}_{\varepsilon_l}$:

- the root belongs to the tree data structure,

$$\widehat{t}_j^0 = \text{ true },\forall j \in I_0;$$

- and for all other grid levels:

$$\widehat{t}_j^l = \text{ false },\forall l \in [1,L] \text{ and } \forall j \in I_l \tag{37}$$

for $l = L$ down to 1 **do**
  for $j \in I_l$ **do**
    if $|d_j^l| \leq \varepsilon_l$ **then**
      $d_j^l = 0.$ and the *node*-cell is potentially discarded from the tree $\widehat{t}_j^l = $ false,
    else
      $(j,l) \in \widetilde{\Lambda}_{\varepsilon_l}$, and the *node*-cell is marked as a cell of the tree: $\widehat{t}_j^l = $ true ,
    end if
  end for
end for
Second step: ensuring that cells belong to a tree data structure and that the tree is graded $\widetilde{\Lambda}_{\varepsilon_l}$.
for $l = L$ down to 1 **do**
  for $j \in I_l$ **do**
    if $(j,l) \in \widetilde{\Lambda}_{\varepsilon_l}$, *i.e.* $\widehat{t}_j^l = $ true , **then**
      parent-cells that belong to the prediction stencil R_j^l (11), must belong to the tree $\widetilde{\Lambda}_{\varepsilon_l}$:
      for $q = -s$ to $+s$ **do**
        if $(j/2+q, l-1) \notin \widetilde{\Lambda}_{\varepsilon_l}$, *i.e.* $\widehat{t}_{j/2+q}^{l-1} = $ false , **then**
          this *node*-cell is potentially added to the graded tree: $\widehat{t}_{j/2+q}^{l-1} = $ true ,
        end if
      end for
    end if
  end for
end for

We consider the evolution operator (E_j) relating the solution evaluated at two consecutive time steps ($t^n = n\ \delta t$ and $t^{n+1} = (n+1)\ \delta t$): $U_J^{n+1} = E_j\ U_J^n$. During time integration, the graded tree $\widetilde{\Lambda}_\varepsilon$ must be appropriate to both solutions, *i.e.* U_J^n and U_J^{n+1}. Assuming that the solution slowly evolves from one time step to another, the Harten's idea [21] is to look for a graded tree $\widetilde{\Lambda}_\varepsilon^{(n+1)}$ that contains both trees at the two time-steps, *i.e.* $\Lambda_\varepsilon^{(n)}$ and $\Lambda_\varepsilon^{(n+1)}$. The intuitive idea of A. Harten [21] can be formulated as:

Algorithm 4 Harten's thresholding

We assume that the solution $\mathbf{w}$ is known by its cell-average value (v_j^0) on the *root*-cell (V_j^0) and the set of *details* $\left(\mathbf{D}^1, \ldots, \mathbf{D}^L\right)$ for grid-levels $l \in [1, L]$;
we also choose a unique value of the threshold parameter ε;
for $l = L$ down to 1 **do**
  evaluation of the threshold value: $\varepsilon_l = 2^{N_{ndim} \cdot (l-L)}\ \varepsilon$;
  for $j \in I_l$ **do**
    if $\dfrac{\left|d_j^l\right|_{L_1}}{\max_j \left|d_j^l\right|} < \varepsilon_l$ **then**
      $d_j^l = 0$;
      this *node* is marked to be discarded from the tree data structure;
    else
      $d_j^l \in \mathbf{D}^l$
    end if
  end for
end for

Hypothesis 1: Harten's heuristic. Considering a graded tree $\widetilde{\Lambda}_\varepsilon^{(n+1)} \supseteq \Lambda_\varepsilon^{(n)} \bigcup \Lambda_\varepsilon^{(n+1)}$ and the evolution operator $U_J^{n+1} = E_j\ U_J^n$, one would like to have:

$$\|\mathbf{v}^{L^{(n)}} - \mathcal{A}_{\widetilde{\Lambda}_{\varepsilon_l}^{(n+1)}} \mathbf{v}^{L^{(n)}}\| \leq C\varepsilon \quad \text{and} \quad \|\mathbf{v}^{L^{(n+1)}} - \mathcal{A}_{\widetilde{\Lambda}_{\varepsilon_l}^{(n+1)}} \mathbf{v}^{L^{(n+1)}}\| \leq C\varepsilon, \tag{40}$$

Therefore, $\widetilde{\Lambda}_\varepsilon^{(n+1)}$ is adapted to depict the solution at both times $t^n = n\ \delta t$ and $t^{n+1} = (n+1)\ \delta t$.

As far as $\Lambda_\varepsilon^{(n)}$ is a graded tree, the first constraint must be satisfied when equation 38 holds. The second constraint is however much more delicate to satisfy. The question which arises is how to "*enlarge*" the graded tree $\Lambda_\varepsilon^{(n)}$ that has been established through the MR algorithm, knowing the solution at time $n\ \delta t$? The Harten's strategy to enlarge the tree is a function of the *detail* magnitudes [21] and is presented in Algorithm 5.

The first rule *(i)* is related to assumption that the solution slowly propagates with a finite speed. The parameter "K" corresponds to the maximal speed of propagation. K could be chosen as the stencil width of the numerical flux. In fact, as the time step is generally limited by a CFL-condition, the choice $K = 1$ is sufficient in most cases. The second rule *(ii)* is used to foresee discontinuity formation as shock creation for instance. Assuming that the loss of accuracy can be detected on coarse grid-levels, *details* are then used to predict this accuracy loss. The parameter p is then related to the regularity analysis by assuming that *detail* magnitudes of *children*-cells ($|d_{2j}^{l+1}|$, $|d_{2j+1}^{l+1}|$) can be estimated by $2^{-p}\ |d_j^l|$:

$$\frac{\left|d_j^l\right|_{L_1}}{\max_j \left|d_j^l\right|} \geq 2^{(2.p)}\ \varepsilon_l \tag{41}$$

p must then be determined by the regularity of the solution at cell (j, l). Following Harten's heuristic assumption [21] in 1D case, the range of the parameter p in the previous algorithm (Algorithm 5) must be:

$$1 \leq p \leq o - 1,$$

Algorithm 5 Predictive Harten's thresholding

We assume that the solution $\mathbf{w}$ is known by its cell-average value (v_j^0) on the *root*-cell (V_j^0) and the set of *details* $\left(\mathbf{D}^1,\ldots,\mathbf{D}^L\right)$ for grid-levels $l\in[1,L]$;
we also choose a unique value of the threshold parameter ε;
we initiate a binary flag, $\widehat{t}_j^l$ that marks if a *node*-cell (j,l) belongs or not to the tree $\widetilde{\Lambda}_{\varepsilon_l}$:

- the root belongs to the tree data structure,

$$\widehat{t}_j^0 = \text{ true },\forall j\in I_0;$$

- and for all other grid levels:

$$\widehat{t}_j^l = \text{ false },\forall l\in[1,L] \text{ and } \forall j\in I_l$$

for $l=L-1$ down to 1 **do**
  evaluation of the threshold value: $\varepsilon_l=2^{N_{ndim}\cdot(l-L)}\ \varepsilon$;
  for $j\in I_l$ **do**
    if *(i)* $\dfrac{\left|d_j^l\right|_{L_1}}{\max_j\left|d_j^l\right|}<\varepsilon_l$ **then**
      $d_j^l=0$;
      the *children-nodes* are marked to be discarded from the tree data structure,
      i.e. $\widehat{t}_{2j}^{l+1}=$ false and $\widehat{t}_{2j+1}^{l+1}=$ false ;
    else
      $d_j^l\in\mathbf{D}^l$
      and the *children-nodes* are kept in the tree data structure,
      i.e. $\widehat{t}_{2j+q}^{l+1}=$ true with $-K\le q\le K+1$;
      if *(ii)* $\dfrac{\left|d_j^l\right|_{L_1}}{\max_j\left|d_j^l\right|}\ge 2^{(2\cdot p)}\ \varepsilon_l$ **then**
        if $l\neq L-1$ **then**
          a new grid-level is locally created and children are created to the *children-nodes* of the current cell;
          i.e. $\widehat{t}_{2q}^{l+2}=$ true and $\widehat{t}_{2q+1}^{l+2}=$ true , with $2j-K\le q\le 2j+1+K$;
        end if
      end if
    end if
  end for
end for

where o is the accuracy order of the approximation interpolation (10). For multidimensional cases, the parameter p can be adjusted and set to $o+1$ or $o+2$ [28]. Previous hypothesis are crucial in the error analysis of the adaptive MR techniques and the assumption (40) has a great impact on the criteria to enlarge the graded tree (see [14] for a rigorous study).

1.6. **Conservativity: Virtual cells**

As far as finite volume approach is concerned, numerical fluxes must be evaluated at cell interfaces. Dealing with hyperbolic conservation laws, the conservative property is then a crucial key point. In order to ensure conservativity, the numerical fluxes on a cell face must have the same value for any cell or set of cells contiguous to the face. Concerning a leaf of a locally refined grid, the adjacent cell might however not exist at the same grid level, as illustrated in figures 4 (in a 1D case) and 5 (in a 2D case). Outgoing fluxes through the right face of both cells $V^{l+1}_{2i+1,2j}$ and $V^{l+1}_{2i+1,2j+1}$ have to be balanced with the outgoing flux through the left face of cell $V^{l}_{i+1,j}$.

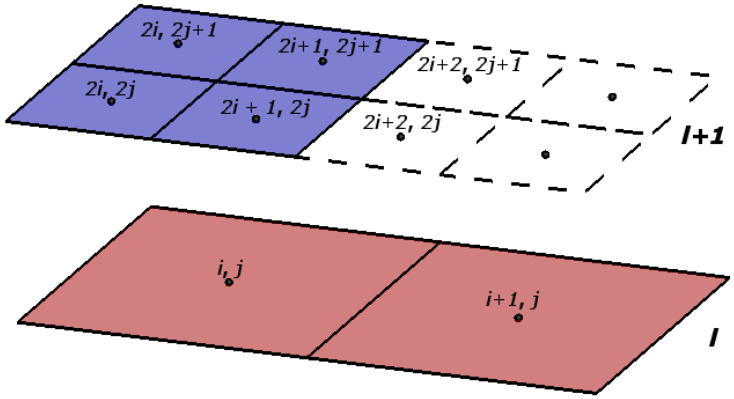


FIGURE 5. Ingoing and outgoing flux computations in 2D on faces between cells at two different grid levels.

In such a situation, the numerical fluxes are always computed at the highest grid level $(l+1)$ by means of *virtual*-cells that have been added to the tree at the grid level $l+1$. Let us note that the solution is not integrated on these *virtual*-cells but just evaluated by using the prediction operator (10, 13, 16) from the solution at time $n\ \delta t$ on the grid level l. To assure a strict conservativity in the flux computation between cells at different grid levels, without increasing significantly the number of costly flux evaluations, the ingoing flux on the leaf at the grid level l is equal to the sum of the outgoing fluxes on leaves at grid level $l+1$ (5) [31], i.e.

$$F^{l}_{i,j\to i+1,j}\ \Gamma^{l}_{i,j\to i+1,j} = \sum_{q=2j}^{2j+1} F^{l+1}_{2i+1,q\to 2i+2,q}\ \Gamma^{l+1}_{2i+1,q\to 2i+2,q}, \tag{42}$$

where Γ is the measure associated to the cell face between two adjacent cells.

2. ALGORITHMS AND CODE STRUCTURE

This section is devoted to the description of the fully adaptive multiresolution (MR) algorithm. The main idea is to let the solution evolve on a hybrid adaptive grid without reconstructing the solution up to the finest grid as it was the case in the original Harten's algorithms. A code structure is then presented based on a tree data structure.

2.1. Adaptive MR Algorithm

In what follows, we present the adaptive MR algorithm (Algorithm 6) which allows to code and let evolve a solution on a hybrid graded tree. From now on, we consider $\mathbf{v}^L$ as the solution on the locally most refined cells, not necessarily on the finest grid.

Algorithm 6 Adaptive MR algorithm

(1) **Initialization:**

1: *Initialize nested grids:* we first build nested grids for grid-levels $l = 0, 1, \cdots, L$ from the coarsest to the finest grid. See (§ 1.1) for more details. There are regular disjoint partitions (cells) V_j^l of an open subset $\Omega \subset \mathbb{R}^d$ (3), such that each V_j^l, $j \in S_l$, is the union of a finite number of cells V_k^l, $k \in S_{l+1}$, and thus, S_l and S_{l+1} are consecutive embedded grids.

2: *Initial solution:* the initial solution is given either on cells of the finest grid V_j^L, $j \in S_L$ or on leaves of an *initial graded tree*, if it exists.

3: *Threshold parameter:* we also define an unique value of the threshold parameter ε. The threshold value for each grid level is then given by: $\varepsilon_l = 2^{N_{ndim} \cdot (l-L)}\ \varepsilon$;

(2) **Beginning of the time integration:**

(3) **For** $time = 0$ up to $n\_time \times \delta t$ (where δt is the time-step prescribed), **Do**

(a) **MR algorithm:**

(i) **Encoding:** Knowing the cell-average values $(\mathbf{v}_j^L)$ of the solution on the locally most refined leaves, *encode* the solution $\mathbf{M}^L = \mathcal{M}\ \mathbf{v}^L$ using Algorithm 1.

(ii) **Predictive Harten's thresholding process:** A multiresolution transformation allows an effective data compression by means of the thresholding process because we literally discarded some cells from the tree data structure. However, we have seen before that a *graded tree* data structure must be respected.

Hence, before alleged useless cells are discarded, they are first marked by using a logical function $(\widehat{t})$ following the Algorithm 5.

(iii) **Building a graded tree:** marking cells or creating new cells that must belong to the tree to be graded $\widetilde{\Lambda}_{\varepsilon_l}$;

1: **while** new cells are created **do**
2: **for** each cell in the tree, $(j,l) \in \widetilde{\Lambda}_{\varepsilon_l}$: *i.e.* $\widehat{t}_j^l =$ true **do**
3: *parent*-cells that belong to the prediction stencil R_j^l (11), must belong to the tree $\widetilde{\Lambda}_{\varepsilon_l}$:
4: **for** $q = -s$ to $+s$ **do**
5: **if** $(j/2+q, l-1) \in \widetilde{\Lambda}_{\varepsilon_l}$ **then**
6: the cell already exists in the tree structure,
7: the cell is kept in the tree: $\widehat{t}_{j/2+q}^{l-1} =$ true .
8: **else**
9: the cell doest not exist: $(j/2+q, l-1) \notin \widetilde{\Lambda}_{\varepsilon_l}$;
10: It must be created and we turn the flag to true: $\widehat{t}_{j/2+q}^{l-1} =$ true ;
11: **end if**

12: **end for**
13: **end for**
14: **Decoding:** For new cells that have been created, cell-average solution must decoded ($\left(\mathbf{v}^{l-2},\ \mathbf{D}^{l-1}\right) \rightarrow \mathbf{v}^{(l-1)}$) following Algorithm 2.
15: **end while**

(iv) **Pruning the graded tree:** once all cells needed to form a graded tree have been marked:

1: **for** $l = L$ down to 1 **do**
2: **for** $j \in I_l$ **do**
3: **if** $\widehat{t}_j^l =$ false , **then**
4: *node* (j, l) is literally deleted from the tree data structure.
5: **end if**
6: **end for**
7: **end for**

(v) **Virtual leaves are added for conservativity constraints:**

1: **for** $l = 0$ up to $L - 1$ **do**
2: **for** $j \in I_l$ **do**
3: **if** the next cells (in each spatial direction) $(j + 1, l)$ do not exist **then**
4: create *virtual* cells $(j+1, l)$ and $(j+2, l)$ and decode the solution on these *virtual*-cells (Algorithm 2)
5: **else**
6: **if** the next cells exist, and have $2^{N_{dim}}$ *children* **then**
7: create *children* ($(2j, l + 1)$ and $(2j + 1, l + 1)$, in each direction) for the current cell (j, l).
8: **end if**
9: **end if**
10: **end for**
11: **end for**

(b) **Perform the time evolution of the solution:**

1: **while** a "true" leaf is selected **do**
2: Time evolution of the solution is then performed by solving equations (1) with a discretized approximation:

$$\mathbf{v}_j^{l\,(n+1)} = E_j\ \mathbf{v}_j^{l\,(n)}$$

3: **end while**
4: **if** $time = time_{save}$, prescribed time for saving solution is reached **then**
5: Print the solution evaluated at $time_{save}$ on each *leaf* of the tree.
6: **end if**

(4) **End For**
(5) **End of time integration**
(6) **Save the solution at the final time**
(7) **END Program**

2.2. **Data structure**

A dynamic *graded tree* structure is used to represent data in the computer memory. The adapted grid corresponds then to a set of nested dyadic grids generated by refining recursively a given cell depending on the local regularity of the solution. The chosen data structure can handle 1D, 2D and 3D Cartesian geometries. In the practice, the tree is represented by a arranged set of cells (defined as *nodes*) so that they can be easily retrieved by means of a searching process from the *root* of the tree or from a coarser grid in a general case. Then, the basic element of the structure is the cell itself, which consists of a set of geometric and physical values, plus pointers to its *parent*, their $2^{N_{dim}}$ *children* and the contiguous cells in each dimension, the *neighbors*. Figure 6 shows an example of a *graded tree* structure in 1D.

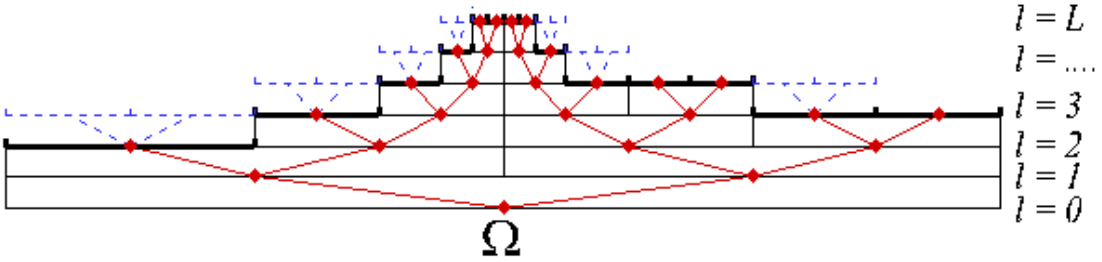


FIGURE 6. Example of a *graded tree* structure in 1D. *Nodes* and links to their corresponding *children* are indicated (red solid lines) as well as the *leaves* (enhanced by bold lines) and the *virtual*-cells (*ghost cells*, in blue dashed lines).

We have previously defined the *roots* as the first cells of the tree structure which correspond to the basis of the tree in figure 6, ($j = 0, l = 0$). Let us recall that the *nodes* are elements of the tree and the *leaves*, the upper elements with no *children* in the tree (see Figure 6). In N_{dim} dimensions, a *parent*-cell at a level l has at most $2^{N_{dim}}$ *children* cells at the level $l + 1$. When there is only one *root* in the tree structure, the maximal number of *leaves* N on which the solution might be represented is given by $N = 2^{L\ N_{dim}}$, which is exactly the number of cells on the finest grid. The maximal number of *nodes* M in the tree is given by $M = (2^{N_{dim}\ (L+1)} - 1)/(2^{N_{dim}} - 1)$.

As far as flux evaluations are concerned, to ensure conservativity, the numerical fluxes should always be computed at the highest grid level by using *virtual*-cells, also called *ghost cells*, that must be added.

We present here a basic straightforward implementation, further optimizations are possible but are not the purpose of this part. See [38], for instance, for detailed optimized implementations. As an example, the cell representation is defined as a derived type in a general FORTRAN environment with the following components stocked within.

`type cell`

- A first flag to indicate if the cell needs to be kept or removed, and an optional second flag to mark *virtual* cells in the case where conservative flux computations are necessary,

`logical :: tree` ⟶ true if it must belong to the tree and false otherwise.
`logical :: leave` ⟶ true if it is a *"true"* leaf and false if it is a *virtual*-leaf.

- The grid level of the cell and its position (index) on the corresponding grid for N_{dim} dimensions; the couple level-index (j, l) allows to define an unique position of each cell into the whole set of nested grids,
  `integer :: level`
  `integer, dimension(`N_{dim}`) :: index`

- Spatial coordinates of the center of the cell and spatial discretization; these might be also calculated at any time knowing the grid level, the index and the space boundaries,
  `double precision, dimension(`N_{dim}`) :: x`
  `double precision, dimension(`N_{dim}`) :: dx`

- Value of the local thresholding parameter; normally, it depends on the grid level and it can also be calculated,
  `double precision :: epsilon`

- Local values of the number of variables ($Nvar$) and the global detail of the cell; the details associated to each variable might be stocked instead of the actual values of the variables during the multiresolution transformation and the global detail (which takes into consideration all variables) might be then computed directly,
  `double precision, dimension(`$Nvar$`) :: u`
  `double precision :: detail`

- Right/left flux on each dimension if flux computations are necessary and need to be stocked for several computations during the same time step,
  `double precision, dimension(`$Nvar, N_{dim}$`) :: flux`

- Pointer on the father of the cell, its children and its neighbors on each dimension; these pointers are then responsible of linking the various existing cells and allow to move from one cell to another in a continuous way,
  `type(cell), pointer :: father`
  `type(child_cell), dimension(2**`N_{dim}`) :: children`
  `type(neighbor_cell), dimension(2*`N_{dim}`) :: neighbors`

`end type cell`

As a vector of pointers is not allowed in Fortran90/95, we have to define new derived type structures, called `child_cell` and `neighbor_cell`, that contains pointers:

```
type child_cell
    • type(cell), pointer :: child
end type child_cell
```

and

```
type neighbor_cell
```

- `type(cell), pointer ::  previous`
- `type(cell), pointer ::  next`

```
end type neighbor_cell
```

Local physical properties for example might also be stocked into the cell-type as well as other computing parameters. However, it is always important to measure the potential benefits of whether stocking or computing the various parameters or variables; therefore, a compromise must be made considering the availability of computing resources and the type of application.

2.3. **Code Structure**

As it was previously stated, this code represents directly the tree-structured data as a set of linked cells by means of pointers. Notice that in FORTRAN 90/95 a pointer is just an alias to the target; nevertheless, we can take advantage of the fact that each pointer has a different state, depending on whether it is associated or not to another object, as we will see in the following.

In the context of multiresolution techniques, we work usually with cells that are at different grids and that are not necessarily arranged in a contiguous way. Hence, we must conceive the mechanisms to navigate through the tree structure. In this illustrating example, we adopt a recursive strategy in which one moves from one cell to another passing through the child of the first, and through the consecutive children, until we get to the desired cell. At each step, the state of the child-pointer tells us whether the target exists or not. As a consequence, any cell can be retrieved by this simple scheme. The knowledge of the neighbors as well as other flags or indicators is not strictly necessary but eases considerably the searching process for certain routines. Notice that this is a fully local approach because we never consider more than one cell at the same time.

In what follows, we consider a number of routines which are executed successively in the main program. As an example, let us consider the routine: `evaluation`, that is called from the main program and in which we perform some computations at the leaves. Previously, we have constructed from the roots, the rest of the tree structure by similar navigating procedures. Therefore, into the Main Program, each step will be performed as :

```
do k = 1, number_root(3)
 do j = 1, number_root(2)
  do i = 1, number_root(1)


   current => root(i, j, k)
   call evaluation(current)


  enddo
 enddo
enddo
```

where

```
type(child_cell), dimension(number_root(1),number_root(2),number_root(3)) ::     root
type(cell) ::  current
```

Into a general routine, in this case `evaluation` as an example, the recursive scheme is presented in the following for evaluating a function only on leaves:

```
recursive subroutine evaluation(current)
if (.not.associated(current%children(1))) then
   !  we are on a leaf,
   current%u = ....
else
  do n = 1, 2**N_dim
    call evaluation(current%children(n))
  enddo
endif
end subroutine evaluation
```

In this recursive way, we are able to locate the leaves considering the states of the pointers that link cells at different levels. The same kind of procedure is conducted in the opposite direction, from the leaves towards the roots, or when the prediction stencil of a given cell needs to be located, for example. This could be achieved using the following example where the value of a function is known on leaves and we would like to propagate the function down to the root:

```
recursive subroutine evaluation(current)
if (.not.associated(current%children(1))) then
   !  we are on a leaf, nothing is done
else
  do n = 1, 2**N_dim
    call evaluation(current%children(n))
  enddo
   current%u = ....
endif
end subroutine evaluation
```

Possibly, other parameters as the index of the cell or the pointers to the neighbors, are also taken into consideration during the research. It is clear that more performing navigation schemes yield certainly more efficient strategies.

3. TEST-CASES

In this part, we apply the MR procedure to several classical test-cases to illustrate the influence of the MR parameters on the performance of the method. Hence, the first example is devoted to check the influence of the threshold value (ε) and the stencil width (s) on the accuracy as well as on the compression rate in the memory

use. This first test-case can be handled in 1D, 2D or 3D Cartesian geometries. Next examples are devoted to the solution of PDE, either for the solution of hyperbolic equations or nonlinear reaction-diffusion problems.

3.1. **Multiresolution on 1D-functions: some examples**

This section illustrates the compression of functions on dyadic grids. Four functions with different smoothness are tested:

$$f_1(x) = \exp\left(-50\ x^2\right) \tag{43}$$

$$f_2(x) = 1. - \sqrt{\left|\sin\left(\frac{\pi}{2}\ x\right)\right|} \tag{44}$$

$$f_3(x) = \tanh\left(50.\ x\right) \tag{45}$$

$$f_4(x) = 0.5 - |x| \tag{46}$$

where $x \in [-1, 1]$.

A code written in Fortran95 is provided to check the influence of some MR parameters, namely the influence of the threshold value (ε) and the stencil width (s) of the prediction operator which gives the accuracy order ($o = 2s + 1$) as we have already seen in (10). This code, whose main program is named `MR_function.f90`, is based on a dyadic tree structure and derived type variables.

Figures 7, 8, 9, and 10 illustrate the MR coding of the four functions (equations 43, 44, 45, and 46). We ran the code using a domain extent $x \in [-1, +1]$ with a width of the polynomial approximation $s = 1$ and a threshold value $\varepsilon = 10^{-2}$.

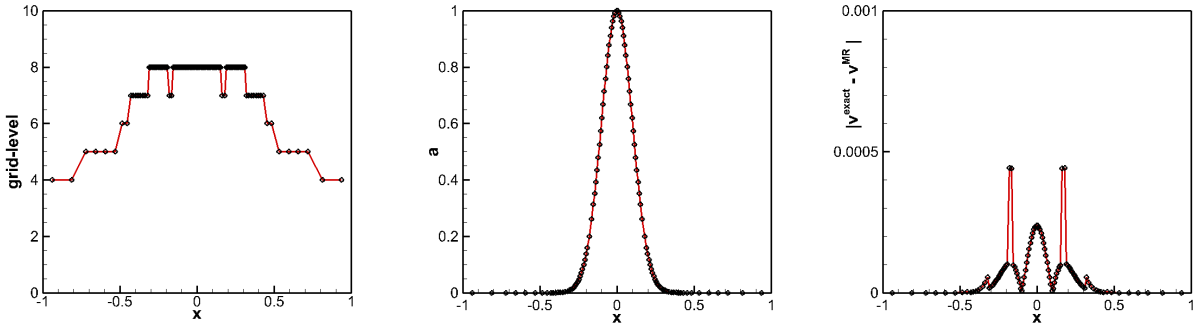


FIGURE 7. Analysis of the function $y = f_1(x)$, with a reconstruction accuracy $o = 3$ and a threshold value $\varepsilon = 10^{-2}$: on the left, grid level and locations of leaves; in the center, the reconstructed function on leaves, and on the right, error magnitudes of the reconstruction compared to the exact solution.

The threshold parameter (ε) has a crucial influence on the error introduced by the MR procedure [11]; therefore, it is important to check how MR solutions converge with ε. On figure 11, we plotted *the perturbation error* in L^∞, L^1 and L^2-norms, calculated as the difference between the exact solution and the MR solution for several ε values, obtained on 10 grid levels. The perturbation error almost recovers a linear fit versus ε whatever the width of the polynomial approximation is. This is in complete agreement with the heuristic MR approach

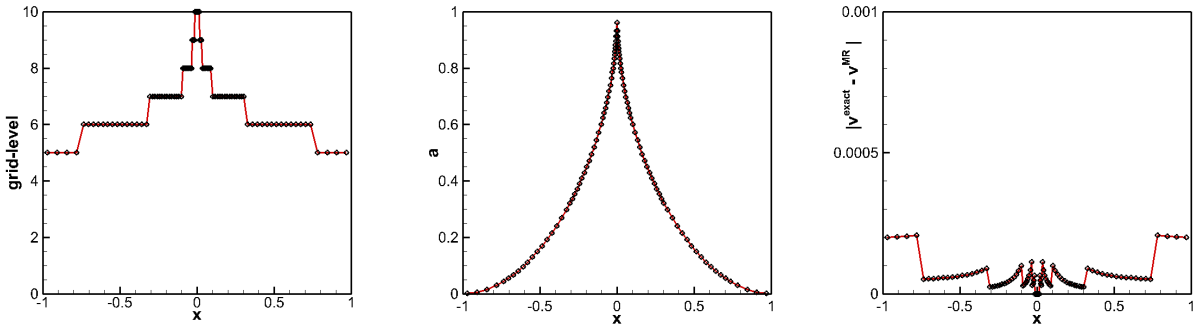


FIGURE 8. Analysis of the function $y = f_2(x)$, with a reconstruction accuracy $o = 3$ and a threshold value $\varepsilon = 10^{-2}$: on the left, grid level and locations of leaves; in the center, the reconstructed function on leaves, and on the right, error magnitudes of the reconstruction compared to the exact solution.

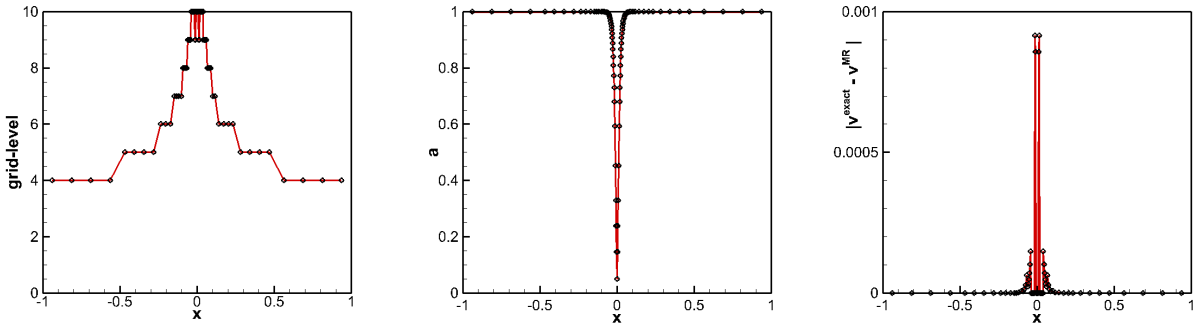


FIGURE 9. Analysis of the function $y = f_3(x)$, with a reconstruction accuracy $o = 3$ and a threshold value $\varepsilon = 10^{-2}$: on the left, grid level and locations of leaves; in the center, the reconstructed function on leaves, and on the right, error magnitudes of the reconstruction compared to the exact solution.

(39) from Harten's work [21]. Error levels between several values of the s parameter cannot be compared to each other since solutions are not obtained on the same refined grid. However, as we can see on Fig. 11-right, the memory usage decreases when s increases, meaning that the compression rate increases with s. Let us note that the memory usage is calculated as the ratio between the number of grid points used to calculate the solution on the adapted grid and the total number of grid points of the finest grid. This is related to the accuracy of the prediction operator that needs coarser grids for higher approximation orders to recover the same error level.

3.1.1. *Implementing adaptive grid*

The structure of a grid cell, as previously mentioned (see § 2.2), is defined in a module, named `mod_structure.f90`. Basic parameters of the MR algorithm are defined in a common file, named `mod_common.f90`.

All these routines are included in a module named `mod_arbre.f90`.

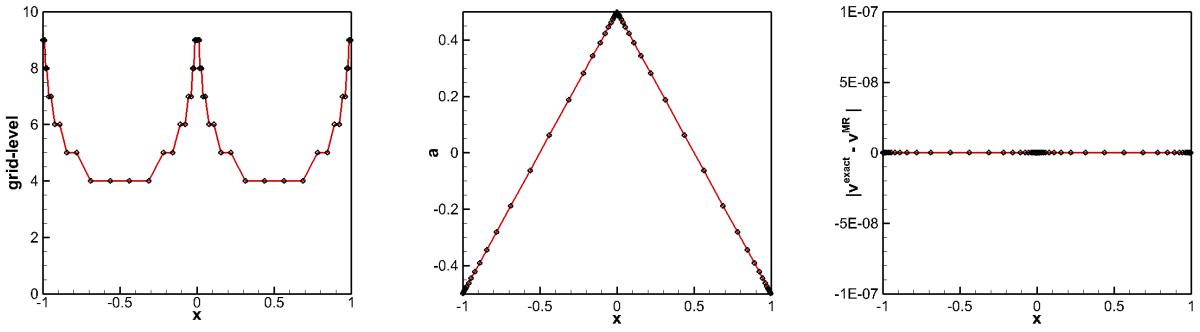


FIGURE 10. Analysis of the function $y = f_4(x)$, with a reconstruction accuracy $o = 3$ and a threshold value $\varepsilon = 10^{-2}$: on the left, grid level and locations of leaves; in the center, the reconstructed function on leaves, and on the right, error magnitudes of the reconstruction compared to the exact solution.

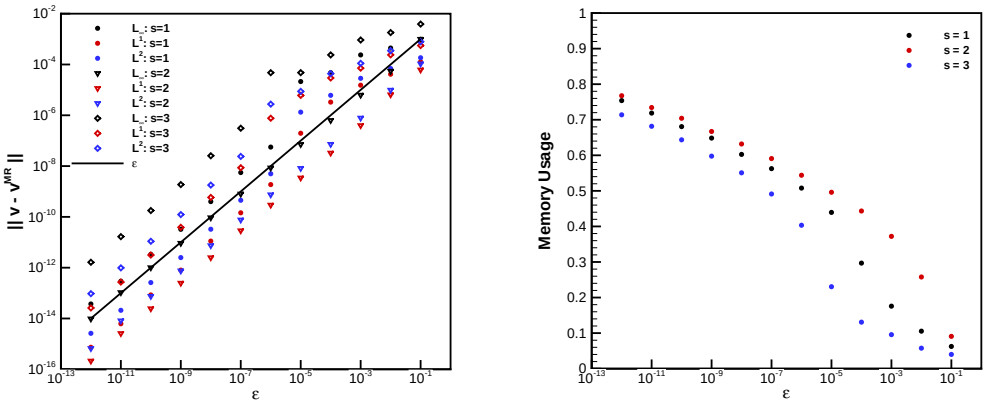


FIGURE 11. Capability of the MR procedure: On the left, L^∞, L^1 and L^2-norms of the *perturbation error* versus the threshold parameter (ε) obtained on 10 grid levels (*i.e.* the finest grid has 1024 grid points); on the right, the memory usage compared to the finest grid.

By means of the *make* command, which reads the `Makefile` provided, the compilation builds an executable, named `mr_function.x`.

Input parameters are read in a data file, named `don_function.dat`, that gathers all parameters that can be modified in order to check their influence. This file is structured as follow:

- *num_funct* $\in [1, 4]$: `the function number, corresponding to functions (43), (44), (45), and (46), respectively`;
- *lim_left, lim_right*: `Left and right limits of the domain;`
- *niv_max*: `the maximum grid level;`
- *niv_min*: `the minimum grid level;`

Algorithm 7 MR_functions

1: Input parameters (geometrical and MR parameters) are read from a data file (named `don_function.dat`) with routine: `lect_don.f90`.
2: The tree structure is built and linked, in recursive procedures:
- creating and linking nodes in the tree: `chainage.f90`;
- linking roots if we employ a tree forest: `support_racine.f90`;
- and linking support corresponding to the approximation stencil: `chainage_support.f90`.

3: The initial solution is prescribed on the leaves of the tree: `init_sol.f90`
This routine is in the module `mod_fonction.f90`, where the various functions are defined.
4: **The MR algorithm is based on several recursive routines devoted to:**
- Encoding the average value at cell centers, from leaves down to the root: `encoding_moy.f90`;
- Encoding details, from leaves down to the root: `encoding_det.f90`;
- Thresholding the solution with respect to ε value: `seuillage.f90`;
- Building the graduate tree: `graduation.f90` and `graduation_local.f90`;
- Decoding the average value at the cell centers, from the root up to leaves: `decoding_moy.f90`;
- Discarding undesirable leaves: `elagage.f90`;
- Linking cells for the approximation stencil in the tree structure: `chainage_support.f90`;

5: Listing and printing leaves: `liste_feuille.f90` and `print_leaves.f90`, respectively.

- s: `the stencil width of the prediction approximation;`
- ε: `the threshold value.`

Let us notice that the maximum number of points used on the finest grid is: $N = 2^{niv\_max}$.
At the exit, two files are generated:

- a file containing the solution evaluated on leaves of the tree. It is named:
  `Tree_funct(`*num_funct*`)_niv(`*niv_max*`)_L1s(`s`)_eps(`ε`).dat`,
  where (*num_funct*) is the value of the function number, (*niv_max*) is the value of the maximum grid level, (s) is the value of the stencil width, and (ε) is the threshold value. The five columns of this file correspond to: the leaf coordinate, the grid spacing, the leaf level, the value of the function on the leaf and the error compared to the exact solution evaluated on the leaf.
- a file containing errors with respect to the exact solution. It is named:
  `Err_funct(`*num_funct*`)_niv(`*niv_max*`)_L1s(`s`).dat`.
  Results are saved in columns: Four columns corresponding to: the threshold value, ε; the L_∞ norm, the L_1 norm, the L^2 norm of errors compared to the exact solution, and compression rate $N_{leaves}/2^{niv\_max}$.

Finally, in order to run the program after compiling, we just need to type:
`./mr_function.x`

However, a script shell (named `function.sh`) is also provided to run the executable several times for different threshold values (ε). The threshold value starts at $\varepsilon = 10^{-1}$ and ends at $\varepsilon = 10^{-12}$ with a step of 10^{-1}. Errors for these different ε values are collected in the file `Err_funct(`*num_funct*`)_niv(`*niv_max*`)_L1s(`s`).dat` so that

they could be plotted versus the threshold parameter (with "gnuplot", for instance) to check the order of the method.

Using this executable, your can:

(1) Check the influence of the threshold parameter (ε) for all the functions prescribed in the code.
- Prescribe the number of a function you would like to check;
- Prescribe the number of grid levels, 10 for instance;
- Prescribe the width of the approximation stencil, $s = 1$, for instance;
- Run the code (`./mr_function.x`) with the script shell `./function.sh` to let evolve ε from 10^{-1} to 10^{-12};
- Plot errors compared to the exact solution for several threshold values, versus ε by using gnuplot, for instance.
  (Errors are recorded in `Err_funct(`*num_funct*`)_niv(`*niv_max*`)_L1s(`*s*`).dat`
- You could also plot the compression rate of memory use versus ε.

(2) Check the influence of the width of the approximation stencil (s) for the above described functions.
- Prescribe the width of the approximation stencil;
- Run the code (`./mr_function.x`) with the script shell `./function.sh` to let evolve ε from 10^{-1} to 10^{-12};
- Plot errors compared to the exact solution versus ε.
- You could also plot the compression rate of memory use versus ε.

(3) Check the influence of the constraint (41 in Algorithm-5) to foresee discontinuity by modifying the variable `precis` (which is originally set to $2\ s - 1$) in the routine `seuillage.f90`.
Recompile the executable and proceed as above to check the influence on the accuracy or the memory compression.

(4) Prescribe your favorite functions in the module `mod_fonction.f90`.
Recompile and proceed as above to check the influence of the MR parameters on the accuracy and memory use.

(5) Modify the number of dimensions of the problem (N_{dim}) in the module file `mod_common.f90`.
Be sure that functions are well defined for multi-dimension evaluation.
Recompile and proceed as above to check the influence of the MR parameters on the accuracy and memory use.

3.2. **Hyperbolic equations: scalar advection**

We here focus on the solution $u(\mathbf{x}, t)$ of the linear scalar transport equation:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \nabla \cdot \mathbf{f}(u) = 0, \quad \text{in } \Omega, \\ u(0,t) = u(1,t), \quad \text{on } \partial\Omega, \\ u(\mathbf{x},0) = u_0(\mathbf{x}), \end{cases} \tag{47}$$

with $\mathbf{f}(u) = \mathbf{a}\ u$, $\mathbf{a}$ being the velocity which is independent of the solution $u(\mathbf{x}, t)$. We propose to solve this transport equation by using MR algorithm in both 1D and 2D Cartesian grids.

3.2.1. *1D scalar advection*

Assuming that the speed is constant, the equation reads:

$$\frac{\partial u}{\partial t} + a\ \frac{\partial u}{\partial x} = 0. \tag{48}$$

with periodic boundary conditions.

The analytical solution of this equation is the convection at the speed a of the initial solution:

$$u(x,t) = u_0(x - a\ t). \tag{49}$$

In this example, we prescribe the initial solution at:

$$u(x,0) = -V_{left}\ \sin(2.\ \pi\ x)\,;\ x \in [0,1], \tag{50}$$

where V_{left} is an input value.

3.2.1.1. *Numerical method:* This equation is solved using a One-Step scheme developed in Daru & Tenaud [15] based on a Lax-Wendroff approximation [24]. This scheme is constructed in the scalar case by correcting the error terms of the successive modified equations, to yield one additional order of accuracy at each level. For stability reasons, the error term, involving highest degree derivatives at each level, is discretized using upwind formulae for odd derivatives and centered formulae for even derivatives. In this way, one obtains a recurrence relation to construct schemes with arbitrary order of accuracy in time and space. This scheme (named OS7) is time-space coupled, with 7th-order time and space accuracy. It was shown Daru & Tenaud [16] that OS7 is at least six times more efficient in CPU time performance than the well known method of lines using a Runge-Kutta time integration and WENO schemes for space discretizations that recover the same accuracy.

To illustrate this scheme, let us introduce the discrete form of equation 49:

$$u_j^{n+1} = E_j\ u_j^n = u_j^n - \frac{\delta t}{\delta x}(F_{j+1/2}^{lw} - F_{j-1/2}^{lw}), \tag{51}$$

where $F_{j+1/2}^{lw}$ is the Lax-Wendroff numerical flux:

$$F_{j+1/2}^{lw} = f_j^n + \frac{(1-\nu)}{2}(f_{j+1}^n - f_j^n), \tag{52}$$

and ν is the local CFL number $\nu = a\frac{\delta t}{\delta x}$. The modified equation for this scheme reads:

$$u_t + f(u)_x = a\frac{\delta x^2}{6}(\nu^2 - 1)u_{xxx}. \tag{53}$$

By subtracting from the Lax-Wendroff scheme an upwind term formed by discretizing the right hand side of (53), one obtains the classical third order upwind-biased scheme with a numerical flux which can be written:

$$F_{j+1/2}^{3} = f_j^n + \frac{(1-\nu)}{2}(f_{j+1}^n - f_j^n - \frac{1+\nu}{3}(f_{j+1}^n - 2f_j^n + f_{j-1}^n)). \tag{54}$$

For convenience, this numerical flux can be recast in the generic form:

$$F^o_{j+1/2} = f^n_j + \Phi^o_{j+1/2}\frac{(1-\nu)}{2}(f^n_{j+1} - f^n_j). \tag{55}$$

So, the discretization of equation 49 that defines the operator (E_j) is the following:

$$u^{n+1}_j = E_j\ u^n_j = u^n_j - \frac{\delta t}{\delta x}(F^o_{j+1/2} - F^o_{j-1/2}), \tag{56}$$

Thus, the o-th order scheme is expressed in the usual form of a second order flux limiter scheme, and the $\Phi^o_{j+1/2}$ function plays the role of an accuracy function that drives the o-th order of accuracy of the scheme. Following such successive corrections of the higher order error terms, one can construct schemes of arbitrarily high o-th order of accuracy. In this way we can obtain the function Φ^o corresponding to a o-th order (in time and space) scheme: We performed the successive derivations of the modified equations up to sixth order. By defining $r_{j+1/2} = \frac{u^n_j - u^n_{j-1}}{u^n_{j+1} - u^n_j}$, Φ functions up the the seventh order of accuracy are provided here:

(57)

$$\begin{aligned}
\Phi^3_{j+1/2} &= 1 - \frac{1+\nu}{3}(1 - r_{j+1/2}),\\
\Phi^4_{j+1/2} &= \Phi^3_{j+1/2} + \frac{1+\nu}{3}\cdot\frac{\nu-2}{4}(1 - 2\ r_{j+1/2} + r_{j+1/2}\ r_{j-1/2})\\
\Phi^5_{j+1/2} &= \Phi^4_{j+1/2} - \frac{1+\nu}{3}\cdot\frac{\nu-2}{4}\cdot\frac{\nu-3}{5}\cdot\\
&\quad\left(\frac{1}{r_{j+3/2}} - 3 + 3\ r_{j+1/2} - r_{j+1/2}\ r_{j-1/2}\right)
\end{aligned}$$

$$\begin{aligned}
\Phi^6_{j+1/2} &= \Phi^5_{j+1/2} + \frac{1+\nu}{3}\cdot\frac{\nu-2}{4}\cdot\frac{\nu-3}{5}\cdot\frac{\nu+2}{6}\cdot\\
&\quad\left(\frac{1}{r_{j+3/2}\ r_{j+5/2}} - \frac{4}{r_{j+3/2}} + 6 - 4\ r_{j+1/2} + r_{j+1/2}\ r_{j-1/2}\right)
\end{aligned} \tag{58}$$

$$\begin{aligned}
\Phi^7_{j+1/2} &= \Phi^6_{j+1/2} - \frac{1+\nu}{3}\cdot\frac{\nu-2}{4}\cdot\frac{\nu-3}{5}\cdot\frac{\nu+2}{6}\cdot\frac{\nu+3}{7}\cdot\\
&\quad\left(\frac{1}{r_{j+3/2}\ r_{j+5/2}} - \frac{5}{r_{j+3/2}} + 10 - 10\ r_{j+1/2} + 5\ r_{j+1/2}\ r_{j-1/2}\right.\\
&\quad\left. - r_{j+1/2}\ r_{j-1/2}\ r_{j-3/2}\right)
\end{aligned}$$

Let us emphasize that the numerical schemes constructed in this way always have the same order of accuracy in time and space. Here we use a stencil of only nine points to get a seventh order scheme relative to both time and space. This kind of scheme has the property, which seems desirable, of giving the exact solution if the CFL number is equal to 1, though the scheme has a classical CFL stability condition; $0 \leq \nu \leq 1$.

3.2.1.2. *Implementing adaptive grid:* We consider here the propagation of an initial function with a constant speed, using a grid adaptivity based on a MR technique. The structure of the code is the one described in the

Algorithm-6 by using the evolution operator defined in equation (56) based on a one-step 7-th order accurate scheme in time and space [15, 16].

To run the code that solves this problem using MR technique, we just need to type:
`./mr_scalar1D.x`

Input parameters are read in a data file, named `don_scalar1D.dat`, that gathers all parameters that can be modified in order to check their influence. This file is structured as follow:

- *lim_left, lim_right*: `Left and right limits of the domain;`
- *caltype_left, caltype_right*: `Left and right boundary conditions at limits of the domain;` `caltype` must take one of the following values:
  - `caltype = -1` $\Rightarrow$ `symmetry condition`
  - `caltype = 0` $\Rightarrow$ `homogeneous Neumann condition`
  - `caltype = 1` $\Rightarrow$ `homogeneous Dirichlet condition`
  - `caltype = 2` $\Rightarrow$ `periodic condition`
- V_{left}: `magnitude of the initial condition as defined in (50);`
- V_{right}: `magnitude of the initial condition , if needed;`
- *CFL*: `CFL number,` $0 \leq CFL \leq 1$;
- N_{max}: `number of time steps;`
- *niv_max*: `the maximum grid level;`
- *niv_min*: `the minimum grid level;`
- s: `the stencil width of the prediction approximation;`
- ε: `the threshold value.`
- *t_min*: `initial recorded time;`
- δt_{save}: `time step for the recorded time;`
- *t_final*: `final time at which the simulation stops.`

Let us notice that the maximum number of points used on the finest grid is: $N = 2^{niv\_max}$.

At the exit, the solution evaluated on the leaves of the tree is saved in a file named:
`Tree_t(`*time_save*`)_niv(`*niv_max*`)_L1s(`*s*`)_eps(`ε`).dat`,

where (*time_save*) is the time value at which the solution is saved, (*niv_max*) is the value of the maximum grid level, (s) is the value of the stencil width, and (ε) is the threshold value. The five columns of this file correspond to: the leaf coordinate, the grid spacing, the leaf level, the value of the solution on the leaf and the error compared to the exact solution evaluated on the leaf.

Using the executable `./mr_scalar1D.x`, your can:

(1) Check the influence of the maximum grid level (*niv_max*).

(2) Check the influence of the threshold parameter (ε).

(3) Check the influence of the width of the approximation stencil (s) for the above described functions.

We know that the *perturbation* (MR) error is linear versus the threshold parameter ε. To recover this behavior, you must first run the code with $\varepsilon = 0.$ to get the solution on the finest grid everywhere. You must secondly save the solution for different ε values. The *perturbation* error is defined as the difference between the finite-volume solution evaluated everywhere on the finest grid and the solution obtained on the refined grid for different ε values by using the MR algorithm. Since the solution on an adapted grid is not evaluated at the same locations than on the finest grid, one needs to propagate the solution evaluated on the adapted grid toward the finest grid, assuming that $details = 0.$ and using the approximation stencil with the prescribed width (s). This could

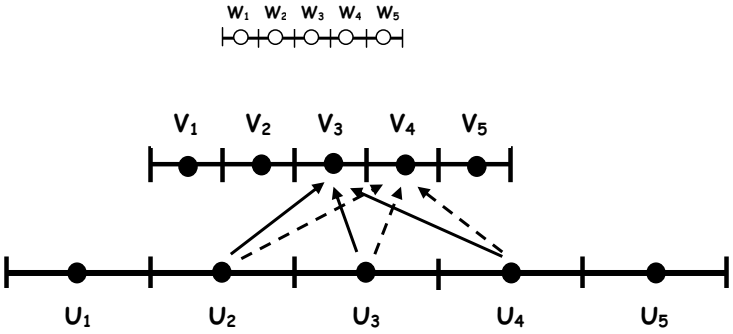


FIGURE 12. Local reconstruction by mean value using approximation polynomial based on $s = 1$ nearest points.

be accomplished by means of a linear function applied on 5 centered values, as it is illustrated in the scheme (Fig. 12):

$$\mathbf{V} = A_v \mathbf{U} \quad \text{with} \quad A_v = \begin{pmatrix} -\frac{1}{8} & 1 & \frac{1}{8} & 0 & 0 \\ +\frac{1}{8} & 1 & -\frac{1}{8} & 0 & 0 \\ 0 & -\frac{1}{8} & 1 & \frac{1}{8} & 0 \\ 0 & +\frac{1}{8} & 1 & -\frac{1}{8} & 0 \\ 0 & 0 & -\frac{1}{8} & 1 & \frac{1}{8} \end{pmatrix} \tag{59}$$

- Could you write down a routine that propagates the solution known on the adapted grid towards the finest grid;
- Calculate the *perturbation* errors in the L^1-norm;
- Then, plot the *perturbation* errors versus the threshold parameter (ε) to show that this error recovers a linear fit with ε.
- To study the efficiency of the algorithm in terms of memory compression, you could also plot the compression rate of memory use versus ε.

3.2.1.3. *Numerical results:* We performed integration of equation (48) by using the OS7 scheme, assuming that the characteristic speed $a = 1$. Figure 13 illustrates the mesh adaptation by the MR algorithm for the solution

computed at several times on a mesh with 7 grid-levels. These solutions are obtained by using a domain extent $x \in [-1, +1]$ with a width of the approximation polynomial $s = 1$ and a threshold value $\varepsilon = 10^{-2}$.

Solutions are also obtained for several grid-levels, namely 7 and 10 grid-levels corresponding respectively to 128 and 1024 grid-points on the finest grids, with several values of the polynomial approximation ($s = 1, 2$ or 3), and for several threshold values (ε, ranging from 10^{-8} up to 10^{-1}). Perturbation errors have been calculated using these parameter values. They are in agreement with the heuristic Harten's theory [21] (39) since they recover a linear fit versus ε (Fig. 14). Though error levels between several parameter values cannot be compared because the solutions are not obtained on the same grid, this behavior is recorded whatever the s value is. The influence of the polynomial approximation width (s) is more pronounced on the memory usage and the CPU time consumption as we can see on Fig. 15. Let us recall that the memory usage is calculated as the ratio between the number of grid points used to calculate the solution on the adapted grid and the total number of grid points of the finest grid. The memory usage decreases or equivalently the memory compression increases, when the order of approximation increases (*i.e.* s increases). In fact, we can see that the larger s value, locally the coarser the grid. This explains why the number of grid points needed to evaluate the solution decreases when we increase the width of the polynomial approximation. Besides, the CPU time needed to operate the multiresolution is approximately two times the integration time of the equations. Therefore, the MR algorithm becomes competitive when about 50 % of the grid points are erased from the memory, as illustrated on the figure (15-right) for $s = 1$ for instance. As far as $\varepsilon > 10^{-5}$, less than 60 % of the CPU time used to compute the solution on the finest grid everywhere ($\varepsilon = 0.$) is needed for a stencil with $s = 1$. Increasing the width of the polynomial approximation, increases the CPU time; for instance compared to $s = 1$, the CPU time is increased of about 15 % when $s = 3$ if the mesh is not sufficiently coarsened (Fig. 15-right). However as the number of grid points needed to evaluate the solution is lowered when s or equivalently the order of approximation increases, the CPU time with $s = 3$ is lower than (or at least equivalent to) the one required to compute the solution with $s = 1$, demonstrating that it is relevant to chose a higher approximation order for this test-case.

3.2.2. *2D scalar advection*

In that case, the equation reads:

$$\frac{\partial u}{\partial t} + \mathbf{a} \cdot \nabla u = 0. \tag{60}$$

with periodic boundary conditions and the initial condition $u(\mathbf{x}, t) = u_0(\mathbf{x})$. Here $\mathbf{a}$ is a vector with two components that are independent of the solution $u(\mathbf{x}, t)$

In the present example, the computational domain is $(x \times y) \in [-1, 1] \times [-1, 1]$. We consider the transport of a circular spot that is initially located at $x_0 = 0.5$, $y_0 = 0.$:

$$u_0(x, y) = (1. - C)\ V_{right} + C\ V_{left}, \tag{61}$$

where

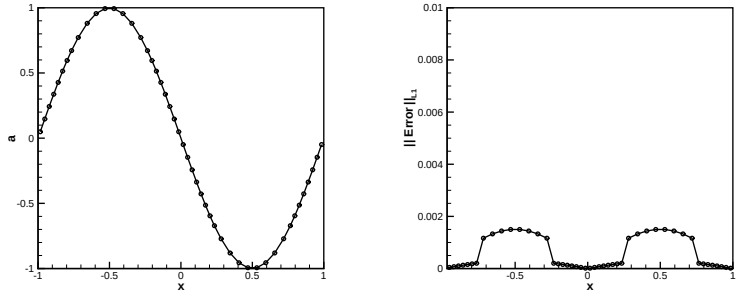
$$C = \begin{cases} 1. & \text{if } \sqrt{(x - x_0)^2 + (y - y_0)^2} \leq r_0^2 \\ 0. & \text{otherwise} \end{cases},$$

with $r_0 = 0.25$

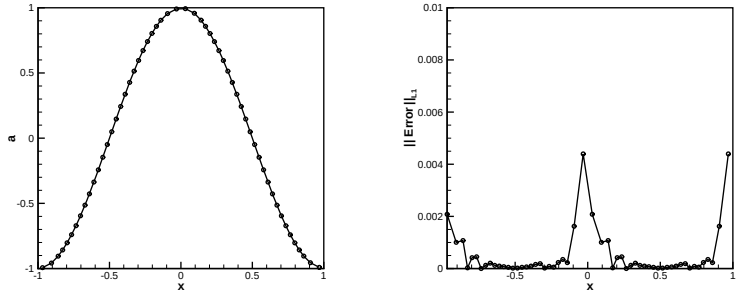
The transport velocity vector $\mathbf{a}$ is prescribed by using the flow of a solid body-type forced vortex:

$$\begin{aligned} a(1) &= -y, \\ a(2) &= +x. \end{aligned} \tag{62}$$

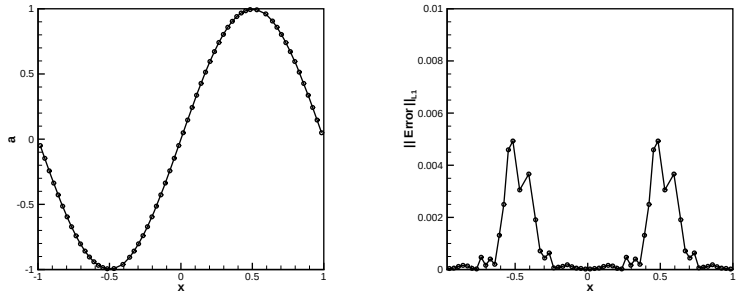
$t = 0.$



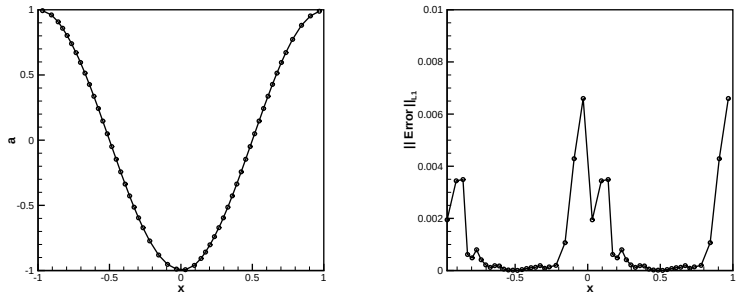
$t = 0.5$



$t = 1.$



$t = 1.5$



$t = 2$.

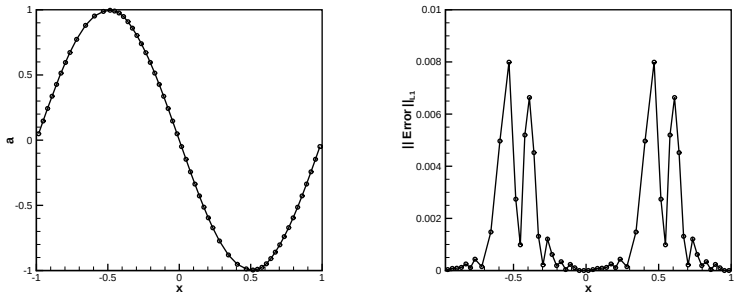


FIGURE 13. Scalar advection in 1D, with a reconstruction accuracy $o = 3$, ($s = 1$) and a threshold value $\varepsilon = 10^{-2}$: the initial solution obtained on 7 grid levels (N = 128 grid points for the finest grid), is convected with a unit velocity $a = 1$. From the top to the bottom, the solution is plotted at the initial time ($t = 0$) and at several recorded times ($t = 0.5$, $t = 1$, $t = 1.5$, and $t = 2$).

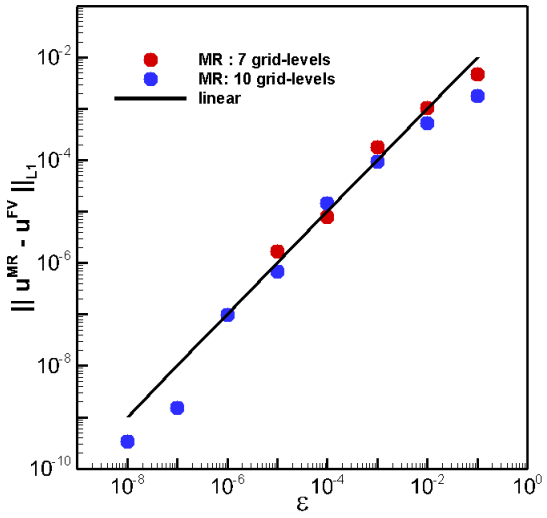


FIGURE 14. L^1-norms of the *pertubation error* versus the threshold parameter (ε) obtained on both 7 and 10 grid levels (*i.e.* the finest grid has 128 and 1024 grid points, respectively).

3.2.2.1. *Numerical method:* This equation is solved using a One-Step scheme (OS7). The extension in the multidimensional case is delicate as far as a one-step approach is used. In fact, we need to consider cross derivative terms that appear in the second and higher order terms. While one could consider higher order splitting methods (see [18], for instance), the simplest way to avoid the problem of cross derivatives and to

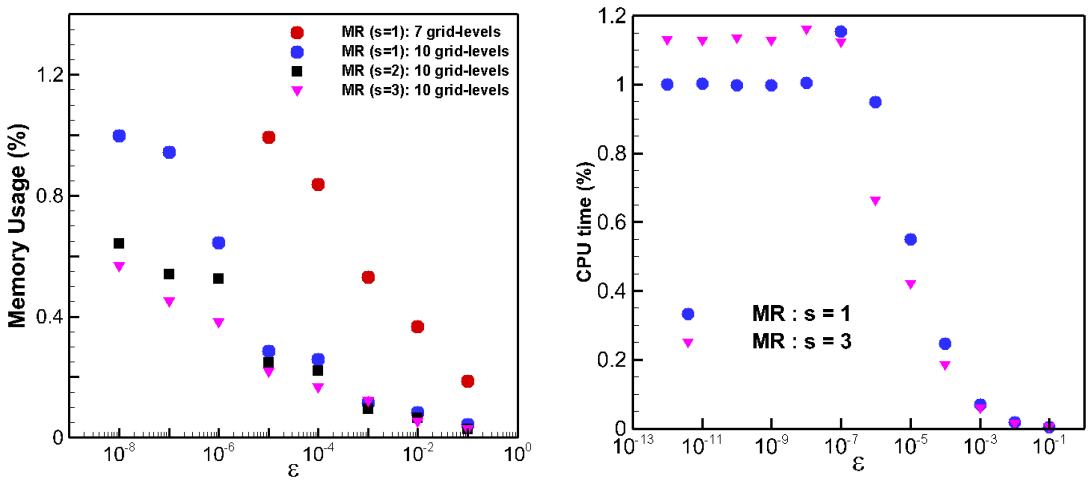


FIGURE 15. Capability of the MR procedure: on the left, the MR memory usage compared to the computation of the solution on a unique finest grid obtained for several ε values and different s values, on both 7 and 10 grid levels (*i.e.* the finest grid has 128 and 1024 grid points, respectively); on the right, the CPU time consumption to compute the MR solutions with 10 grid levels (1024 grid points) for $s = 1$ and $s = 3$, compared to the CPU time used to compute the solution with $s = 1$ and $\varepsilon = 0$ (on the finest grid) versus the threshold parameter (ε).

recover the good properties of the one-dimensional scheme is to use a Strang directional splitting strategy [25,35], which however is only second order accurate. While the order of accuracy is lowered compared to the tensorial multistage approach, the OS scheme with the Strang algorithm gives results with very small errors at lower cost [15].

In two dimensions, the splitting of the system of equations (60) is written:

$$w_j^{n+2} = L_{\delta x} L_{\delta y} L_{\delta y} L_{\delta x} \; w_j^n. \tag{63}$$

Here $L_{\delta x}$ and $L_{\delta y}$ are discrete approximations of operators in each space direction. When directional operators do not commute, the second order accuracy is recovered every two time steps with however the symmetric property of the solution [15].

3.2.2.2. *Implementing adaptive grid:* We consider here the propagation of an initial circular spot with constant speed, using a grid adaptivity based on a MR technique. The structure of the code is the one described in the Algorithm-6 by using the evolution operator defined in equation(63) based on a one-step 7-th order accurate scheme in time and space and a directional splitting procedure [15, 16].

To run the code that solves this problem using MR technique, we just need to type:

```
./mr_scalar2D.x
```

Input parameters are read in a data file, named `don_scalar2D.dat`, that gathers all parameters that can be modified in order to check their influence. This file is structured as follow:

- *lim_left(1), lim_right(1)*: `Left and right limits of the domain;`
- *caltype_left(1), caltype_right(1)*: `Left and right boundary conditions at limits of the domain;`
- *lim_left(2), lim_right(2)*: `Bottom and top limits of the domain;`
- *caltype_left(2), caltype_right(2)*: `Bottom and top boundary conditions at limits of the domain;` `caltype` must take one of the following values:
  - `caltype = -1` $\Rightarrow$ `symmetry condition`
  - `caltype = 0` $\Rightarrow$ `homogeneous Neumann condition`
  - `caltype = 1` $\Rightarrow$ `homogeneous Dirichlet condition`
  - `caltype = 2` $\Rightarrow$ `periodic condition`
- V_{left}: `magnitude of the initial solution inside the circular spot as defined in (61);`
- V_{right}: `magnitude of the initial solution outside the circular spot as defined in (61);`
- *CFL*: `CFL number,` $0 \leq CFL \leq 1$;
- N_{max}: `number of time steps;`
- *niv_max*: `the maximum grid level;`
- *niv_min*: `the minimum grid level;`
- s: `the stencil width of the prediction approximation;`
- ε: `the threshold value.`
- *t_min*: `initial recorded time;`
- δt_save: `time step for the recorded time;`
- *t_final*: `final time at which the simulation stops.`

Let us notice that the maximum number of points used on the finest grid is: $N = 2^{niv\_max} \times 2^{niv\_max}$.

At the exit, the solution evaluated on the leaves of the tree is saved in a file named:

`Tree_t(`*time_save*`)_niv(`*niv_max*`)_L1s(`*s*`)_eps(`ε`).dat`,

where (*time_save*) is the time value at which the solution is saved, (*niv_max*) is the value of the maximum grid level, (s) is the value of the stencil width, and (ε) is the threshold value. The solution is written in a finite-element format:

- Corner coordinates of leaves are first listed;
- For each leaf, a list of connectivity between leaf corners is secondly printed;
- For each leaf, values of the solution at center of the leaf are finally printed.

Using the executable `./mr_scalar2D.x`, your can:

(1) Check the influence of the maximum grid level (*niv_max*).

(2) Check the influence of the threshold parameter (ε).

(3) Check the influence of the width of the approximation stencil (s) for the above described functions.

If you would like to compute the *perturbation* error, you need to propagate the solution evaluated on leaves of the adapted grid toward the finest grid and calculate the difference between the finite-volume solution with $\varepsilon = 0$ and the solution obtained on the adapted grid for different ε values.

3.2.2.3. *Numerical results:* We then performed the integration of equation (60) using the OS7 scheme developed in Daru & Tenaud [15], already presented above. Figure 16 illustrates the mesh adaptation by the MR algorithm for the solution computed at several times with 7 grid levels ($N = 128 \times 128$ for the finest grid). These solutions are obtained by using a width of the polynomial approximation $s = 1$ and a threshold value $\varepsilon = 10^{-3}$.

Perturbation errors have been calculated at the final simulated time, $t = 6$, for several ε values ranging from 10^{-6} up to 10^{-2}. For the two different s values ($s = 1$ and 2), the perturbation errors linearly decrease with ε (Fig. 17) that agrees with the heuristic Harten's theory [21] (39). On the one hand, though the perturbation errors decrease with ε, the memory usage does not increase very much (Fig. 18-left) when ε increases. Let us recall that the memory usage is calculated as the ratio between the number of grid points used to calculate the solution on the adapted grid and the total number of grid points of a unique finest grid. In fact, as the discontinuity is well localized in space, the highest grid level of the adapted grid is only tightened in its vicinity and rather low grid levels are used elsewhere, leading to a rather constant low memory usage whatever the ε used is, between 15 and 25 % of the memory required for a unique finest grid. This however explains why the memory usage slightly increases when we increase the width (s) of the polynomial approximation (for a specific ε value). On the other hand, when the s value is increased from $s = 1$ to 2, the MR algorithm mainly adds grid points at the highest grid-level in the vicinity of the discontinuity enlarging the region where the highest grid level occurs, while the mesh stays comparable elsewhere. On the contrary of previous test-cases, as the adapted grid is not dramatically modified when s goes from $s = 1$ to 2 except very close to the discontinuity, the accuracy of the solution increases as we can see for a specific ε value on the perturbation error levels (Fig. 17). CPU time compressions obtained by the MR procedure with $s = 1$ and $s = 2$, are presented on the figure (18-right). They are calculated with respect to the CPU time used for calculating the solution of the finest grid everywhere, with $\varepsilon = 0$ and $s = 1$. The CPU time compression is very high whatever the ε value is since CPU times need less than 40 % of the finest grid computation. As we can see, the CPU time compression is mainly influenced by the memory usage since it rather follows the same trend.

3.3. **Hyperbolic equations: non-linear cases**

We here focus on the solution $u(x,t)$ of the 1D burger equation:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \nabla \cdot f(u) = 0, \quad \text{in } \Omega, \\ u(0,t) = u(1,t), \quad \text{on } \partial\Omega, \\ u(x,0) = u_0(x), \end{cases} \tag{64}$$

with $f(u) = \dfrac{1}{2}\, u^2$. We propose to solve this non linear transport equation by using MR algorithm.

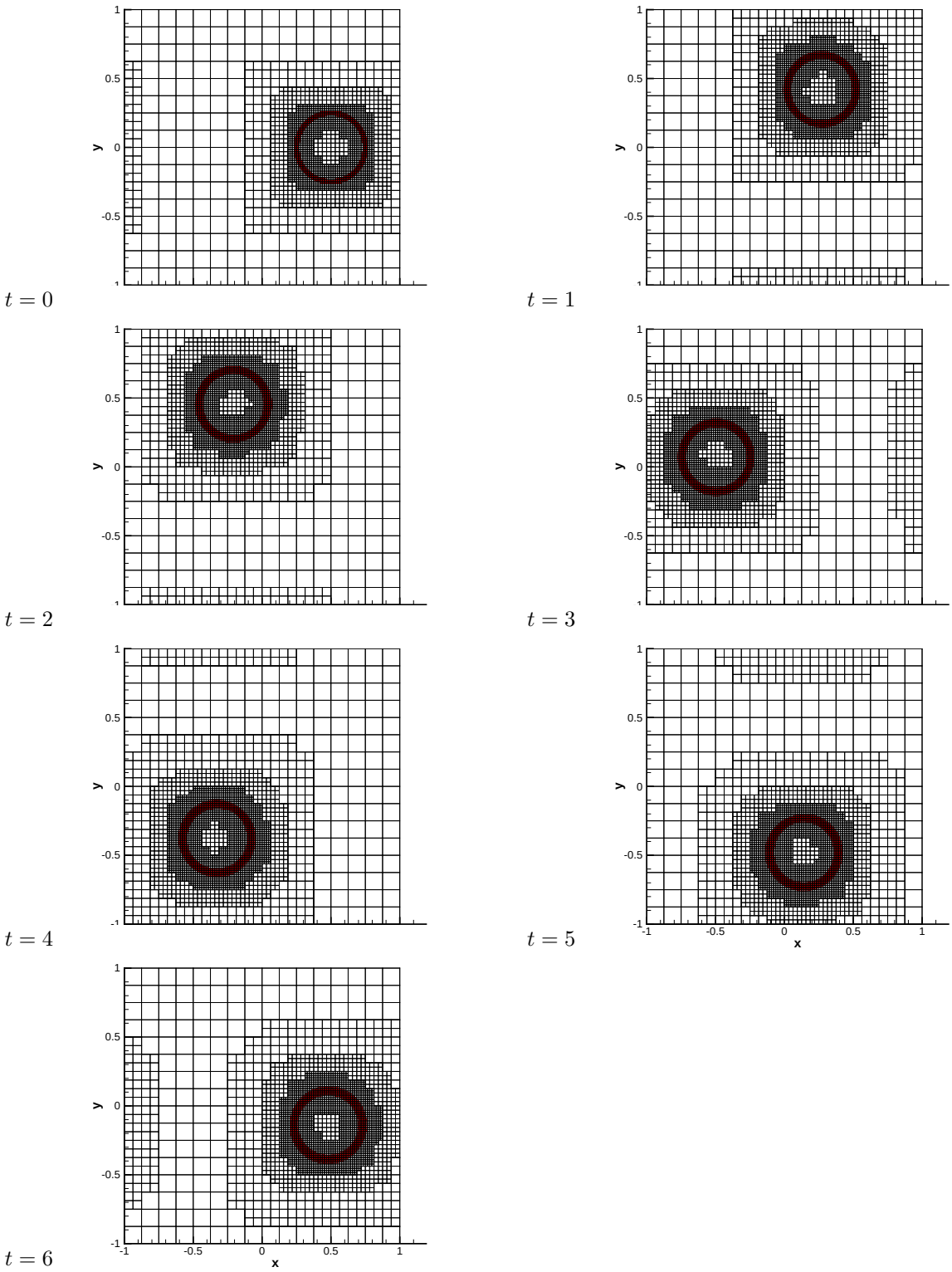


FIGURE 16. Scalar advection in 2D, with a reconstruction accuracy $o = 3$, $(s = 1)$ and a threshold value $\varepsilon = 10^{-3}$: the initial solution is convected with a cylindrical vortex with a speed magnitude $\sqrt{a(1)^2 + a(2)^2} = 0.5$. The solution obtained by using 7 grid-levels, is plotted at the initial time and every dimensionless time unit up to $t = 6$.

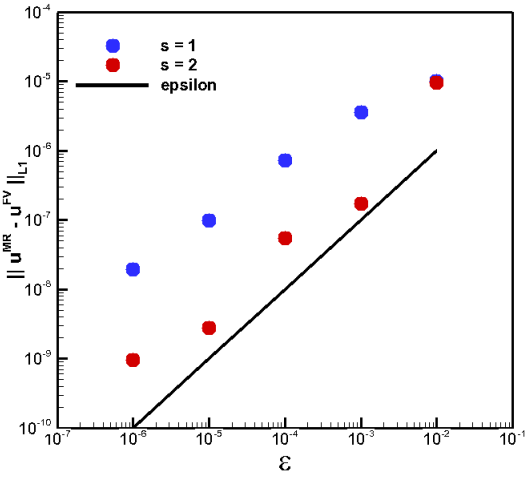


FIGURE 17. L^1-norms of the *pertubation error* versus the threshold parameter (ε) obtained for $s = 1$ and 2 with 7 grid levels (*i.e.* the finest grid has 128×128 grid points).

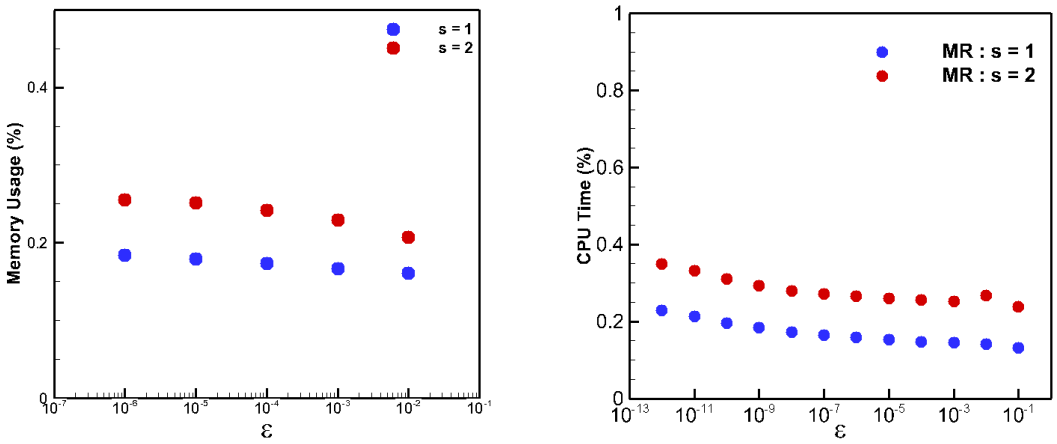


FIGURE 18. Capability of the MR procedure on a 2D scalar advection problem using 7 grid levels (*i.e.* the finest grid has 128×128 grid points): on the left, the MR memory usage compared to the computation of the solution on a unique finest grid for several ε values and for $s = 1$ and 2; on the right, the CPU time consumption to compute the MR solutions for $s = 1$ and $s = 2$, compared to the CPU time used to compute the solution with $s = 1$ and $\varepsilon = 0$ (on the finest grid) versus the threshold parameter (ε).

Here, we consider the initial solution:

$$u(x,0) = -V_{left}\ \sin(2.\ \pi\ x)\,;\ x \in [0,1], \tag{65}$$

where V_{left} is an input value.

3.3.1. *Numerical method*

This equation is solved using a One-Step scheme leading to a coupled time and space algorithm. We know that non linearity can lead to discontinuity creations after a finite time, even if the solution is initially smooth. To prevent spurious oscillations due to Gibbs phenomenon in the vicinity of strong gradient regions, an *ad hoc* limiting process must be employed. Total Variation Diminishing (TVD) schemes are generally considered to be well suited for capturing sharp discontinuities without oscillations. Nevertheless, TVD constraints are known to clip extrema, which appears as a serious drawback in a limiting procedure. To avoid this loss of accuracy near extrema, it is necessary to satisfy the *Monotonicity Preserving* (MP) criteria, introduced by Suresh and Huynh [36], that enlarge the TVD intervals to provide room for the numerical flux to maintain an accurate value. Daru and Tenaud [15] derived a Monotonicity Preserving version of the present scheme (named OSMP scheme) that preserves accuracy near extrema. In particular, the original MP constraints of Suresh and Huynh [36] has been recast in the TVD framework without CFL restriction to generalize the MP conditions in terms of flux limitation. The MP conditions that preserve accuracy can be expressed directly as constraints on Φ^o functions [15] (see equations 55 and 56). The MP constraints can be found in [15, 16].

By taking the limiting function Φ_k^{o-MP} in (55), the resulting scheme will be high-order accurate almost everywhere, except around discontinuities where it becomes first order accurate, which is the case for all TVD schemes. Let us note that the essential difference is that MP constraints act for non-monotone data and so far are TVD for monotone data such that the scheme is not oscillatory around discontinuities.

3.3.1.1. *Implementing adaptive grid:* The structure of the code is the one described in the Algorithm-6 by using the evolution operator E_j based on a one-step 7-th order Monotonicity-Preserving (OSMP7) accurate scheme [15, 16].

To run the code that solves this problem using MR technique, we just need to type:
`./mr_burger1D.x`

Input parameters are read in a data file, named `don_burger1D.dat`, that gathers all parameters that can be modified in order to check their influence. This file is structured as follow:

- *lim_left(1), lim_right(1)*: `Left and right limits of the domain;`
- *caltype_left(1), caltype_right(1)*: `Left and right boundary conditions at limits of the domain;` `caltype` must take one of the following values:
  - `caltype = -1` ⇒ `symmetry condition`
  - `caltype = 0` ⇒ `homogeneous Neumann condition`

- `caltype = 1` $\Rightarrow$ `homogeneous Dirichlet condition`
- `caltype = 2` $\Rightarrow$ `periodic condition`

- V_{left}: `magnitude of the initial solution as defined in (65);`
- V_{right}: `magnitude of the initial solution if needed;`
- CFL: `CFL number,` $0 \leq CFL \leq 1$;
- N_{max}: `number of time steps;`
- *niv_max*: `the maximum grid level;`
- *niv_min*: `the minimum grid level;`
- s: `the stencil width of the prediction approximation;`
- ε: `the threshold value.`
- *t_min*: `initial recorded time;`
- δt_save: `time step for the recorded time;`
- *t_final*: `final time at which the simulation stops.`

Let us notice that the maximum number of points used on the finest grid is: $N = 2^{niv\_max}$.

At the exit, the solution evaluated on the leaves of the tree is saved in a file named:

`Tree_t(`*time_save*`)_niv(`*niv_max*`)_L1s(`*s*`)_eps(`ε`).dat`,

where (*time_save*) is the time value at which the solution is saved, (*niv_max*) is the value of the maximum grid level, (s) is the value of the stencil width, and (ε) is the threshold value. The four columns of this file correspond to: the leaf coordinate, the grid spacing, the leaf grid-level, and the value of the solution on the leaf.

Using the executable `./mr_burger1D.x`, your can:

(1) Check the influence of the maximum grid level (*niv_max*).

(2) Check the influence of the threshold parameter (ε).

(3) Check the influence of the width of the approximation stencil (s) for the above described functions.

To compute the *perturbation* error, you need to propagate the solution evaluated on leaves of the adapted grid toward the finest grid and calculate the difference between the finite-volume solution with $\varepsilon = 0$ and the solution obtained on the adapted grid for different ε values. The linear application 59 and the routine you wrote down previously (See §3.3.1.1) can be used for propagating the solution on the adapted grid toward the finest grid. Hence,

- Calculate the *perturbation* errors in the L^1-norm;
- Then, plot the *perturbation* errors versus the threshold parameter (ε) to show that this error recovers a linear fit with ε.
- To study the efficiency of the algorithm in terms of memory compression, you could also plot the compression rate of memory use versus ε.

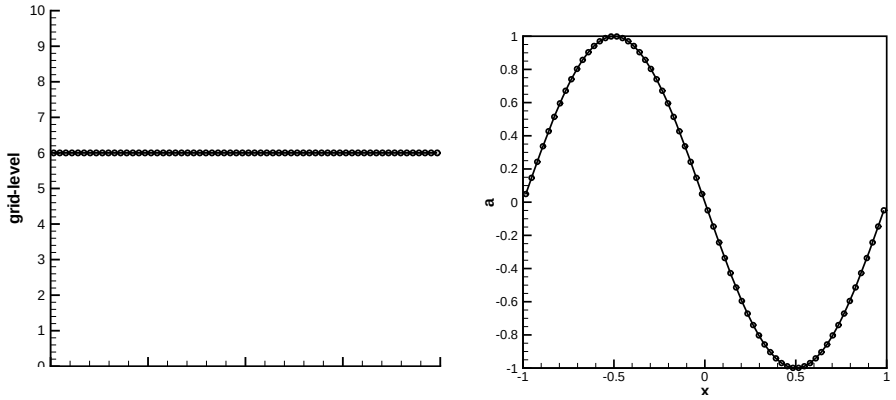
3.3.2. *Numerical results*

We then performed the integration of equation (64) using the limited OSMP7 scheme developed in Daru & Tenaud [15]. Figure 19 illustrates the mesh adaptation by the MR algorithm for the solution computed at

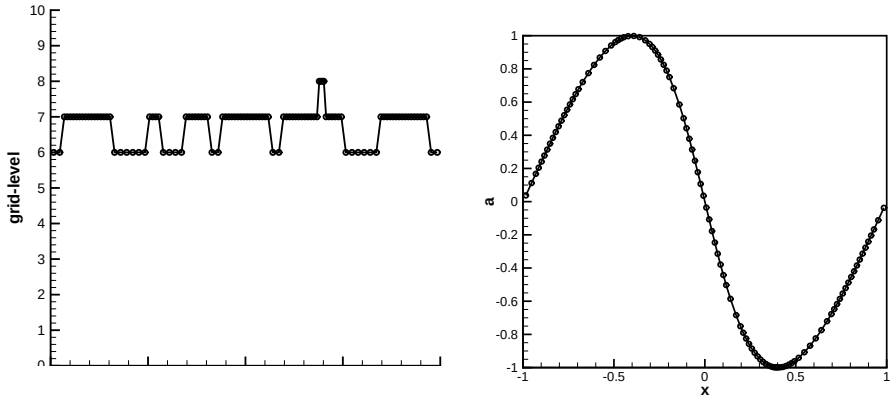
several times with 10 grid levels ($N = 1024$ grid points on the finest grid). These solutions are obtained by using a width of the approximation polynomial $s = 1$ and a threshold value $\varepsilon = 10^{-2}$. We can see (Fig. 19) that a shock like structure is created from a dimensionless time close to $t = 0.4$. In that region, thanks to the predictive Harten's thresholding algorithm (5) based on the additional constraint (41), the MR method foresees the discontinuity by refining the mesh up to the highest grid level ($l = 10$). The solution is finally represented over 5 grid-levels (from $l = 6$ to 10).

Solutions are also obtained for several grid-levels, namely 7 and 10 grid-levels corresponding respectively to 128 and 1024 grid-points on the finest grid, and with several threshold values (ε, ranging from 10^{-8} up to 10^{-1}). Perturbation errors have been calculated at the final simulated time, $t = 0.5$, for these parameter values. One more time, the heuristic Harten's theory [21] (39) holds since perturbation errors linearly decrease with ε (Fig. 20). The memory usage, calculated as the ratio between the number of grid points of the adapted grid and the number of grid points of a unique finest grid, is presented on Figure 21-left for several ε values. The memory compression is much more efficient when using a large number of grid levels; for instance when $\varepsilon = 10^{-3}$ the adapted grid needs 96 grid points for 7 grid-levels while 220 grid points are only needed to compute the solution with 10 grid-levels which is less than the number of grid points of a unique grid with 8 grid-levels (*i.e.* 256 grid-points). In fact, very fine grid points are more localized in the vicinity of discontinuities and grid points are efficiently arranged elsewhere in smooth regions to ensure the accuracy required. Using high number of grid-levels allows us to recover more accurate solutions than on a coarser unique grid without extra memory usage. CPU times per grid point and per number of time steps needed to compute the MR solutions for several threshold parameter values ($10^{-12} \leq \varepsilon \leq 10^{-1}$) are presented on the figure (21-right). Computations are performed on a processor Intel(R)-Xeon(R) CPU 5130 at 2.00 GHz with a cache size of 4096 KB. High compression rates on the CPU time are achieved for 10 grid level solutions as far as $\varepsilon \geq 10^{-5}$. One more time it is exhibited that the CPU time compression is mainly influenced by the memory usage since it rather follows the same trend and the hierarchy between two maximum grid levels are respected. This demonstrates that it is relevant to chose a higher resolution to compute solutions since the right grid level is locally chosen to satisfy the accuracy prescribed.

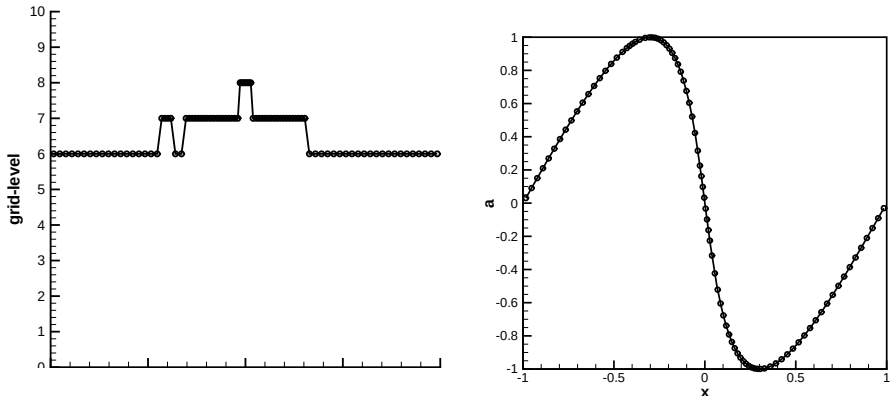
3.4. **Nonlinear reaction-diffusion equation: the KPP equation**



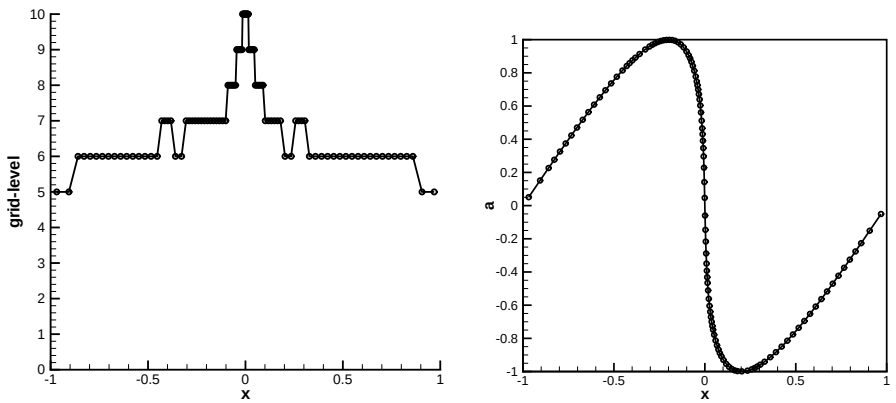
$t = 0$



$t = 0.1$



$t = 0.2$



$t = 0.3$

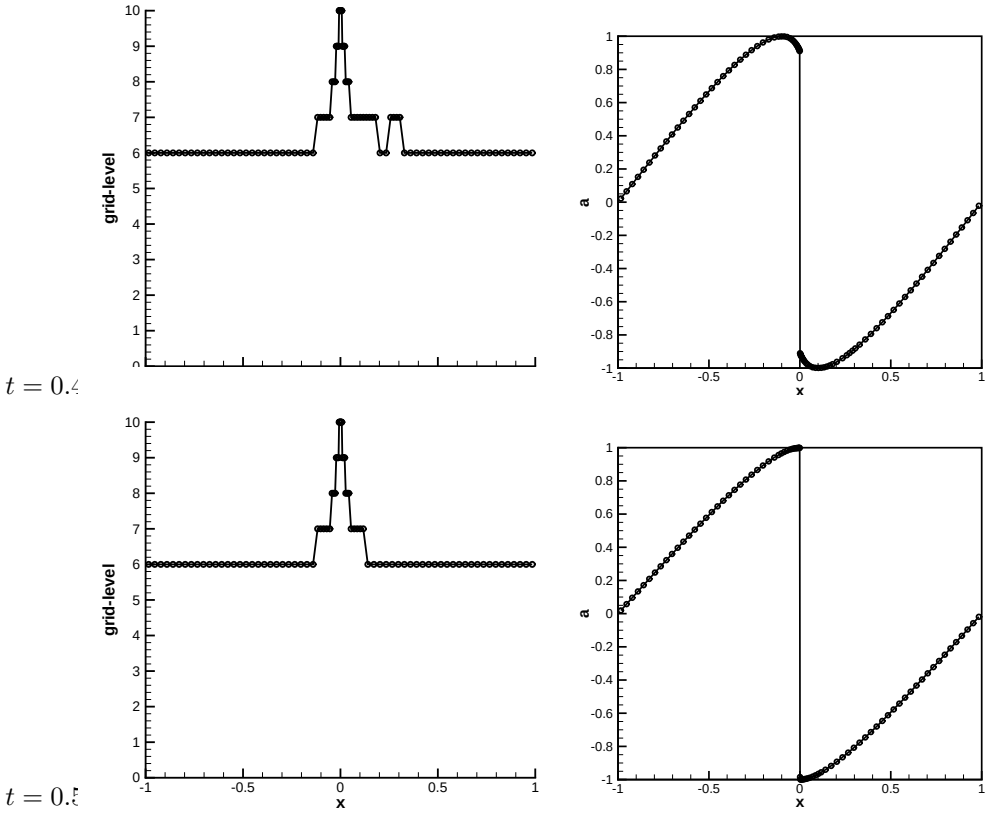


FIGURE 19. Burger equation in 1D, with a reconstruction accuracy $o = 3$, $(s = 1)$ and a threshold value $\varepsilon = 10^{-2}$. The solution obtained by using 10 grid-levels, is plotted at the initial time and every dimensionless time-step of 0.1. On the left, grid levels and locations of leaves; on the right, the solution evaluated on leaves are plotted.

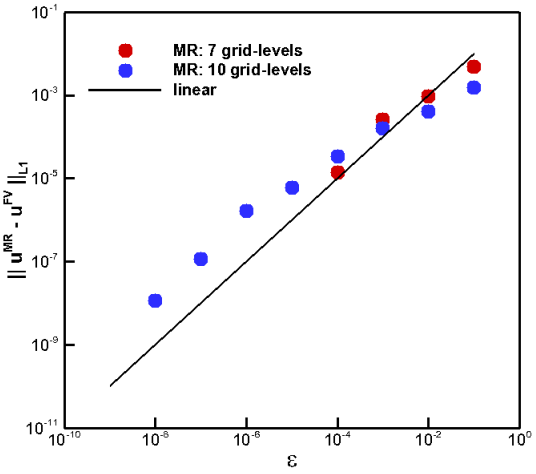


FIGURE 20. L^1-norms of the *pertubation error* versus the threshold parameter (ε) obtained on both 7 and 10 grid levels (*i.e.* the finest grid has 128×128, respectively 1024×1024 grid points).

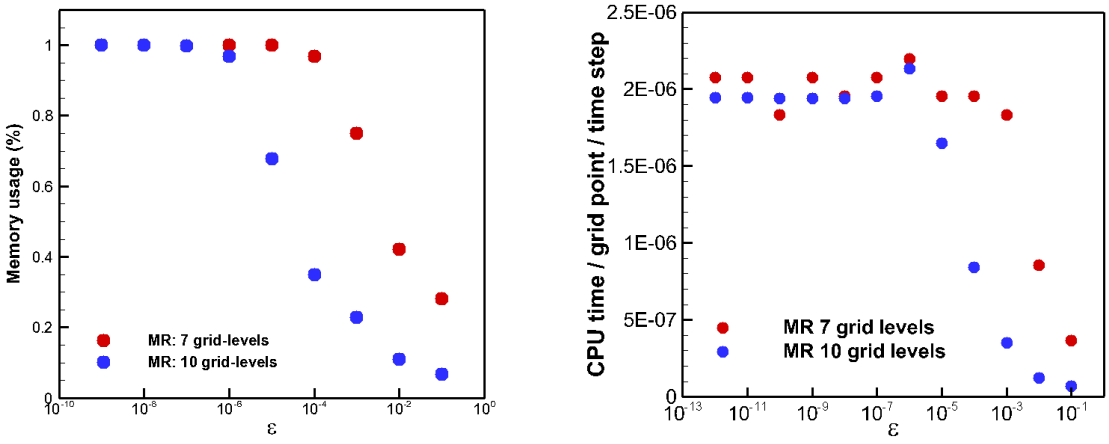


FIGURE 21. Capability of the MR procedure to compute the Burger solution in 1D: on the left, the MR memory usage obtained on both 7 and 10 grid levels (*i.e.* the finest grid has 128×128, respectively 1024×1024 grid points) for several ε, are compared to the memory used for the solution on a unique finest grid; on the right, the CPU time consumption to compute these MR solutions versus the threshold parameter (ε).

In this application, we consider the Kolmogorov-Petrovskii-Piskunov model. In their original paper dated in 1937, these authors introduced a model describing the propagation of a virus and the first rigorous analysis of a stable traveling wave solution of a nonlinear reaction-diffusion equation. The KPP equation reads

$$\frac{\partial u}{\partial t} - D\,\frac{\partial^2 u}{\partial x^2} = k\,u^2(1-u), \tag{66}$$

and we consider homogeneous Neumann boundary conditions.

The analytical solution of this equation is easily obtained,

$$u(x,t) = \frac{\exp\left(-\frac{1}{\sqrt{2}}(\frac{k}{D})^{\frac{1}{2}}(x - x_0 - vt)\right)}{1 + \exp\left(-\frac{1}{\sqrt{2}}(\frac{k}{D})^{\frac{1}{2}}(x - x_0 - vt)\right)}, \tag{67}$$

after considering a steady reaction wave as solution of (66) with steady wave speed v, which is given by

$$v = \frac{1}{\sqrt{2}}(kD)^{\frac{1}{2}}. \tag{68}$$

Finally, the solutions are traveling waves at constant speed with maximal spatial gradient of

$$\left(\frac{\partial u}{\partial x}\right)_{\max} = -\frac{1}{\sqrt{32}}\left(\frac{k}{D}\right)^{\frac{1}{2}}. \tag{69}$$

3.4.1. *Numerical Configuration*

Application of the method of lines with a finite difference second order discretization in space implies a discretization of the Laplacian operator in (66) and thus, leads to a system of nonlinear ODEs. The time integration of the resulting system is then performed using a Strang operator splitting scheme. The latter considers on the one hand, a high order method like Radau5 [19], based on implicit Runge-Kutta schemes for stiff ODEs, that solves the reaction term using adaptive time integration tools and highly optimized linear systems solvers. And on the other hand, another high order method like ROCK4 [1], based on explicit stabilized Runge-Kutta schemes, that solves the diffusion problem. Finally, the Strang scheme is constructed with a half-step reaction time integration performed by Radau5 before and after an entire diffusion integration step, calculated by means of ROCK4.

In the case of $D = 1$ and $k = 1$, the velocity of the self-similar traveling wave is $v = 1/\sqrt{2}$ and the maximal gradient value yields $1/\sqrt{32}$. The structure of the wave can be observed in Figure 22 for eight time intervals into $[0, 30]$ with an uniform discretization of 4096 points of the region $[-70, 70]$.

The key point of this illustration is that the velocity of the traveling wave is proportional to $(k\,D)^{1/2}$, whereas the maximal gradient is proportional to $(k/D)^{1/2}$. Thus, switching to values $k = 10.0$ and $D = 0.1$, the velocity is preserved, but the maximal gradient is multiplied by a factor of 10 and introduces stiffness in the equation, as presented in Figure 22 for the "stiff" KPP wave. For the spatial discretizations considered, the wave, however "stiff", is always well solved on the considered grid.

3.4.2. *Implementing Adaptive Grid*

As we have seen in the previous figures, we are mainly concerned with the simulation of propagating reaction fronts, spatially very localized. Hence, as soon as we consider higher spatial gradients, the precision of the

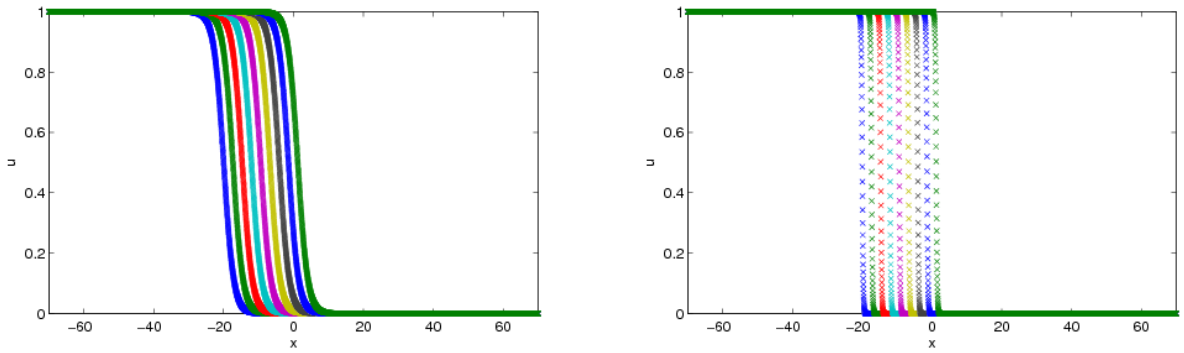


FIGURE 22. Standard (left) and "stiff" (right) KPP traveling waves, discretization with 4096 points on the $[-70, 70]$ region. Self-similar solutions for eight time intervals after the initial condition.

numerical approximations will be guaranteed only if very fine spatial discretizations are taken into consideration. In this context, uniform grids not only need important memory requirements in order to stock data, but also imply a very inefficient time consumption strategy, considering that important chemical activity takes only place over a very low percentage of the spatial domain, in the neighborhood of the wavefront. Therefore, let us consider in this study, the resolution of propagating wavefronts using a grid adaptation technique based on the multiresolution (MR) technique. In particular, we consider the reaction-diffusion system, KPP (66).

The structure of the code that solves this problem using MR is the same described in the Algorithm-6, and uses the dedicated Strang splitting technique, described in the previous part § 3.4.1, as evolution operator E_j. Figures 23 and 24 illustrate the mesh adaptation by the MR algorithm for the initial condition given by the analytical solution (67) and the solution computed at time $t = 30$, with 12 grid levels ($N = 4096$ grid points on the finest grid). These solutions were obtained using as width of the approximation polynomial, $s = 1$, and as threshold value, $\varepsilon = 10^{-8}$. The solution is finally represented over 8 grid-levels (from $l = 5$ to 12), while the minimum grid level taken into consideration during computations was 4.

In order to run the program, we just need to type the executable:
`./calcul_mr.x`
obtained after compilation into a terminal window. However, we need first to define the computing parameters included in the file `input_mr`. This file must be placed in the same directory where we have put the executable. `input_mr` gathers all the parameters that we are able to modify in order to conduct a complete case study. They are detailed in the following:

`input_mr`

- *lim_deb, lim_fin, num_root*: `Left and right limits of the domain, number of roots;`
- N_{max}: `number of splitting time steps;`
- *niv_max*: `the maximum grid level;`
- *niv_min*: `the minimum grid level;`
- s: `the stencil width of the prediction approximation;`
- ε: `the threshold value.`
- *t_min*: `initial saving time;`

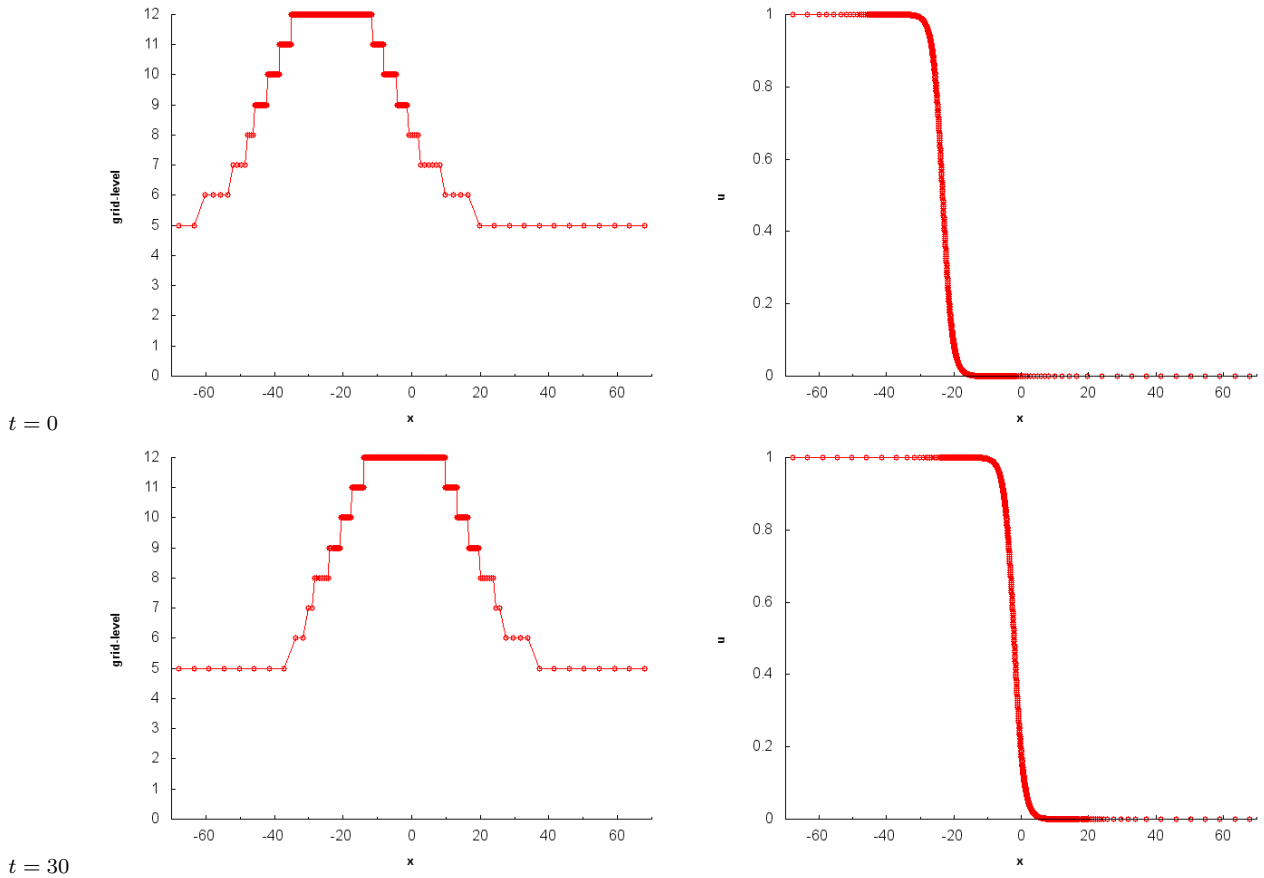


FIGURE 23. Standard KPP equation, with a reconstruction accuracy $o = 3$, $(s = 1)$ and a threshold value $\varepsilon = 10^{-8}$. Solutions were obtained using 12 grid-levels. On the left, grid levels and locations of leaves; on the right, the solution evaluated on leaves are plotted.

- δt_{save}: `time step for saving time;`
- *t_final*: `final time at which the simulation stop;`
- D: `diffusion coefficient,` $kD = 1$`;`
- *Initial*: `initial condition, 0:continuous, 1:discontinuous.`

As in the previous applications, the output files generated are called:
`Tree_t(`$time * 100$`)_niv(`niv_max`)_s(`s`)_eps(`ε`).dat`
and for each leaf, the position of the cell center, the grid level and the local values of the solution are saved in a column-fashioned way.

3.4.3. *Numerical Illustration*

In what follows, we will show and discuss some numerical results in which we change the MR parameters. The spatial domain $[-70, 70]$ is taken as an example, considering that the only restriction is that the wavefront should remain far enough from the boundaries in order to be coherent with the homogeneous Neumann boundary

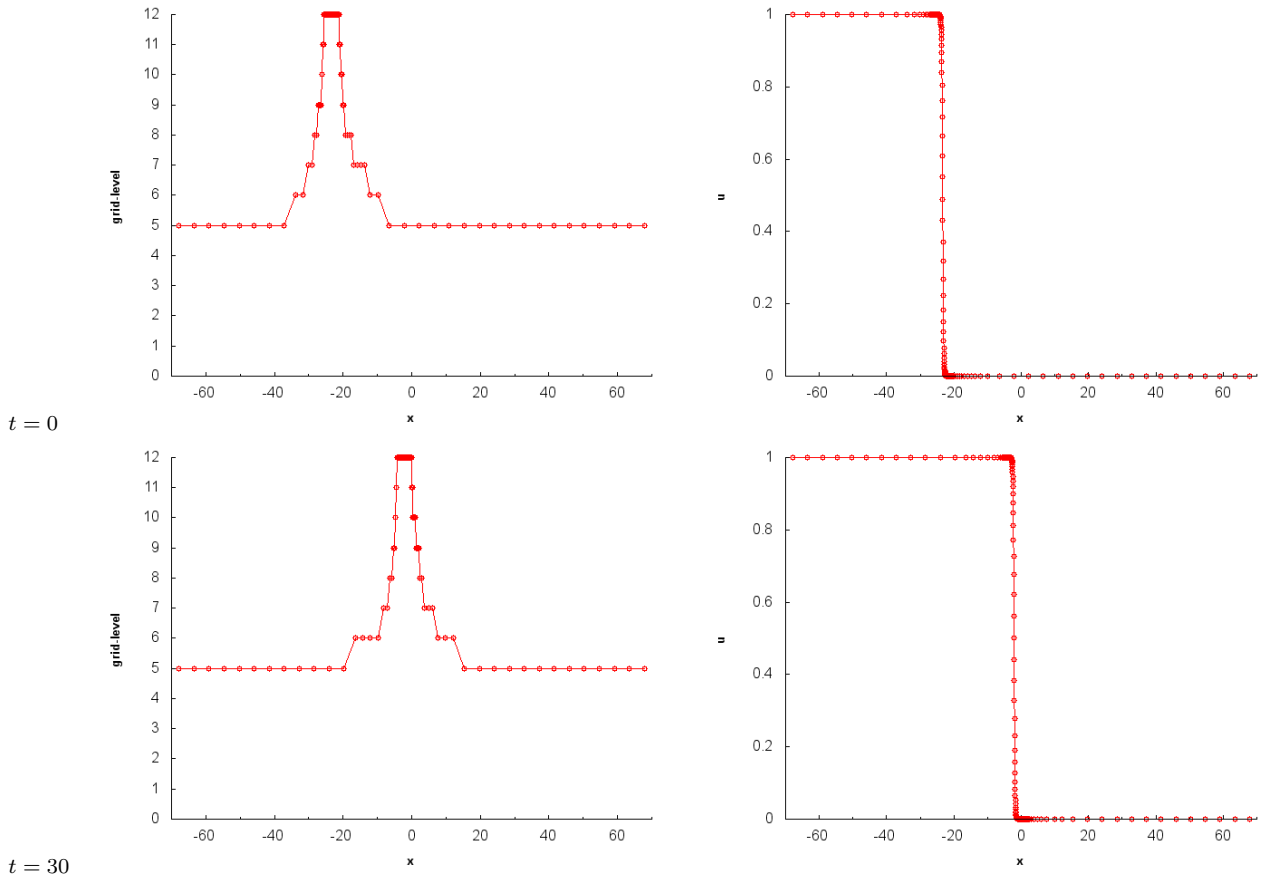


FIGURE 24. "Stiff" KPP equation, with a reconstruction accuracy $o = 3$, $(s = 1)$ and a threshold value $\varepsilon = 10^{-8}$. Solutions were obtained using 12 grid-levels. On the left, grid levels and locations of leaves; on the right, the solution evaluated on leaves are plotted.

conditions. In this application, we are dealing with propagating waves modeled by parabolic PDEs; as a consequence, the adapted grid should take into account the effective speed of propagation of the wave which does not correspond to the speed of propagation of the information in the numerical scheme, as in the hyperbolic case. Nevertheless, we might think of a CFL-like condition for the time step, which considers the fact that the time evolution is performed on a fixed grid and that the wave front should be evaluated on the finest region not because of stability issues but in order to guarantee an appropriate accuracy of the approximation. Anyway in this case, with the adopted refinement criteria and considering the propagating nature of the phenomenon, the splitting time steps are set by the desired level of accuracy in the computation of the wave speed or the wave profile, and thus, becomes a simulation parameter to be fixed practically independent of the MR procedure. As an illustration, in the following examples, we have considered $N_{max} = 1024$, which sets the splitting time step to $30/1024 \approx 3 \times 10^{-2}$ with a resulting difference of $\sim 1.4 \times 10^{-6}$ in the speed approximation for the standard case, and $\sim 1.4 \times 10^{-4}$ for the "stiff" one.

In Figure 25, we show the results for which we consider 8 roots, that is, 8 cells at level $l = 0$. In order to be consistent with the previous results (Fig. 23), we consider $niv\_max = 9$ in order to have the same fine discretization $N = num\_root \times 2^{niv\_max} = 4096$, and for the same reason, $niv\_min = 1$. As it was expected, both approaches give identical results; nevertheless, considering several roots yields a straightforward and simple parallelization technique in which each processor may work with the tree associated to one root. However, optimizing the load balance of processors is a challenging task since the roots are equally distributed throughout the spatial domain, whereas the phenomenon is spatially very localized.

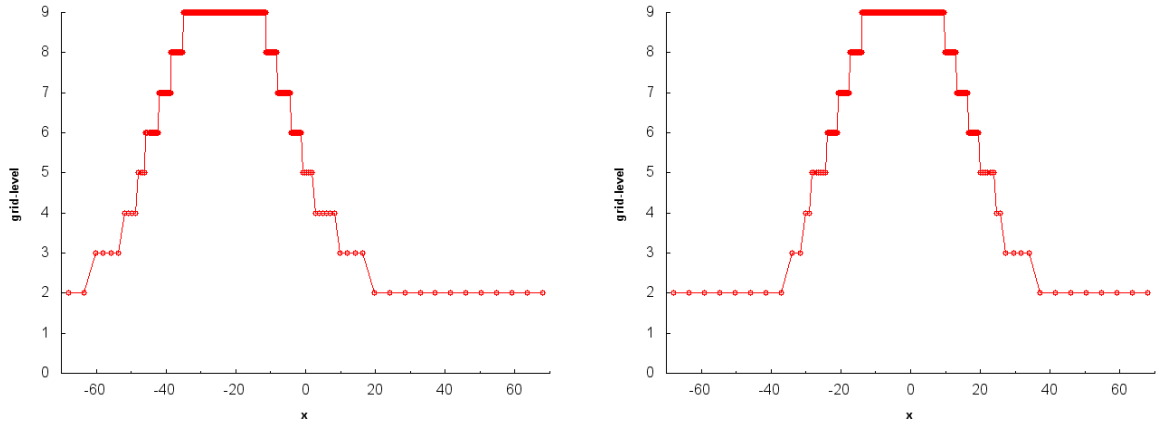


FIGURE 25. Standard KPP equation, with a reconstruction accuracy $o = 3$, $(s = 1)$ and a threshold value $\varepsilon = 10^{-8}$. Solutions were obtained using 9 grid-levels and 8 roots. Grid levels and locations of leaves at $t = 0$ (left) and $t = 30$ (right).

The finest spatial discretization (niv_max) is set by the level of spatial resolution required. From a practical point of view, this parameter fixes the finest grid that can be considered. However, depending on the threshold value as well as on the order of the prediction approximation, there might not be cells at the finest grids as shown in Figures 26 and 27.

As we can see, details are directly related to the order of the prediction approximation (22), specially at smooth regions. Therefore, for a given s, if all the details lie beneath the threshold value ε, it is almost sure that none of the cells will be refined up to the finest scale. Otherwise, new layers of grids will be necessary if one wants to have a MR representation that filters all the spatial scales present in the phenomenon. The threshold values per level ε_l are computed according to

$$\varepsilon_l = 2^{\frac{N_{ndim}}{2}\cdot(l-L)}\ \varepsilon, \tag{70}$$

in order to guarantee (39) in the L^2 norm.

Let us define $N_{leaves}/(num\_root \times 2^{niv\_max})$ as the compression rate, that is, the ratio between the number of leaves and the cells on the finest grid. Figure 28 shows that higher order prediction approximations yield higher compression rates, but also higher errors with respect to the solution computed on the finest grid without MR but with the same time integration method. However, this is a problem dependent feature because even though higher orders imply lower values of details (higher compression rates), they also need larger supports as stencils

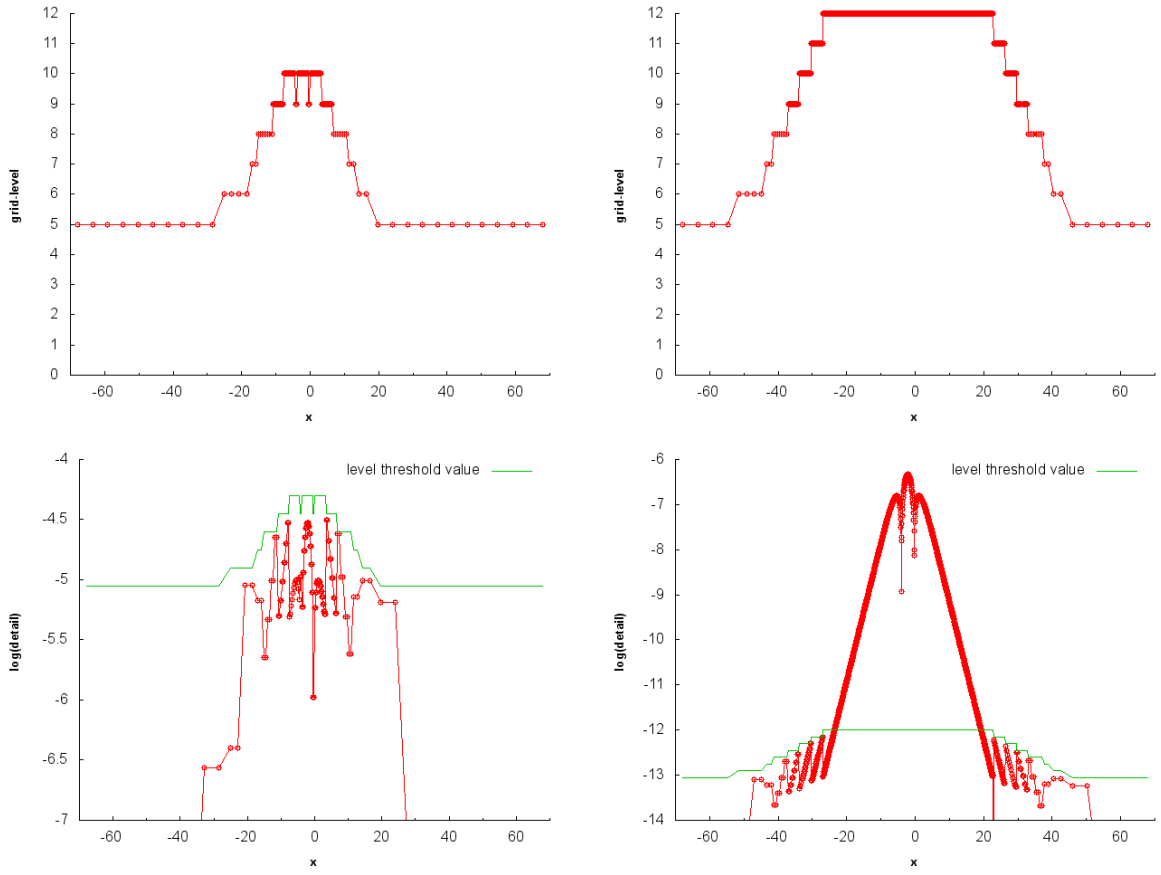


FIGURE 26. Standard KPP equation with a reconstruction accuracy $o = 3$, $(s = 1)$. Top: grid levels and locations of leaves for a threshold value of $\varepsilon = 10^{-4}$ (left) and 10^{-12} (right). Bottom: the corresponding details of the leaves and the threshold values per level.

for the prediction approximations (lower compression rates). Finally, the resulting suitable adapted grid will directly depend on the problem configuration itself. Errors in Figure 28 are computed at the finest level, the MR solutions have been then reconstructed at this level by the prediction operator.

Figure 29 shows MR solutions when one starts simulations from a discontinuous initial configuration, $Initial = 1$. The MR parameters are identical to those illustrated in Figure 23. The wave evolves clearly from the initial condition to the stable self-similar wave previously seen; in consequence, the space scales of the problem also change in time.

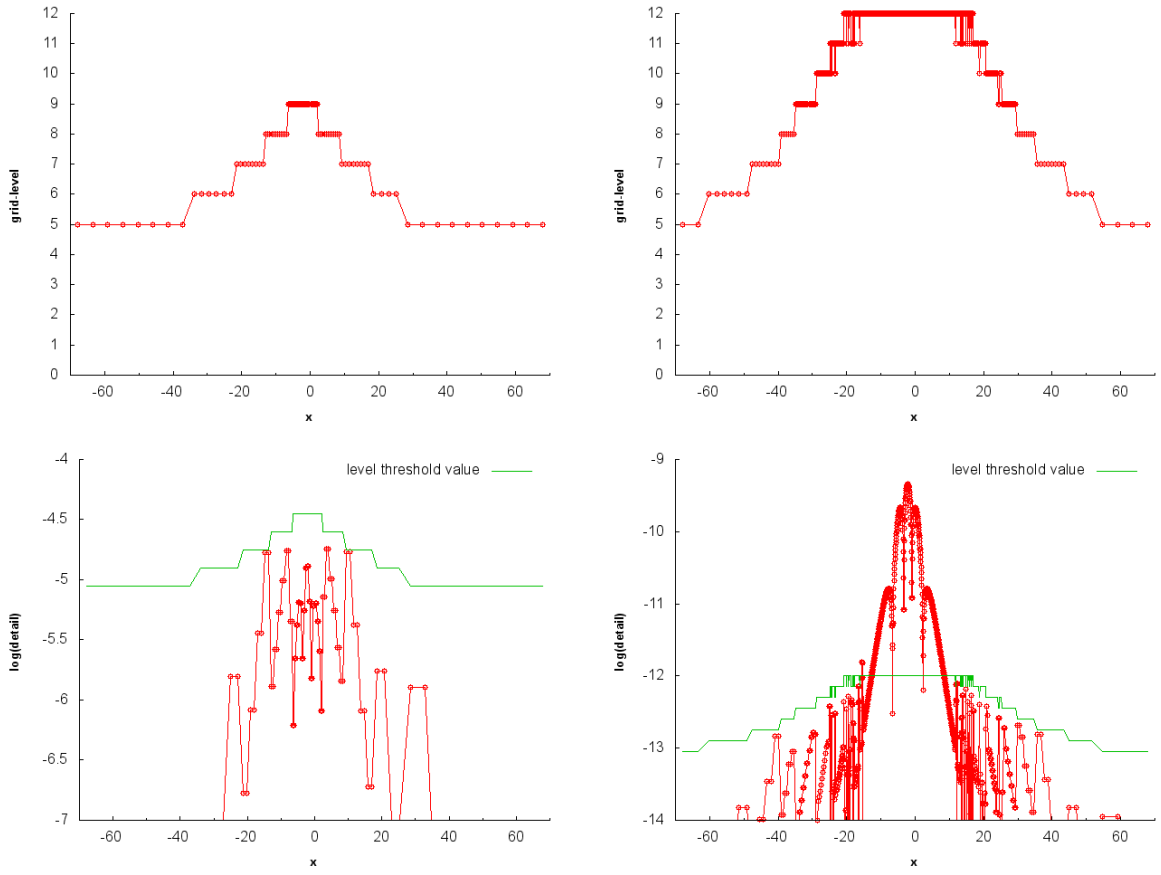


FIGURE 27. Standard KPP equation with a reconstruction accuracy $o = 5$, $(s = 2)$. Top: grid levels and locations of leaves for a threshold value of $\varepsilon = 10^{-4}$ (left) and 10^{-12} (right). Bottom: the corresponding details of the leaves and the threshold values per level.

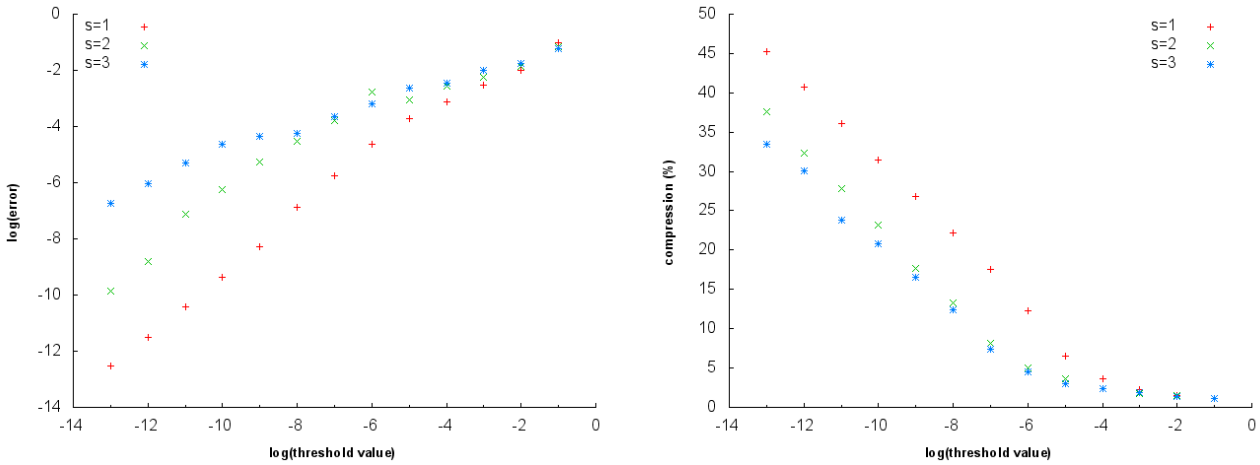
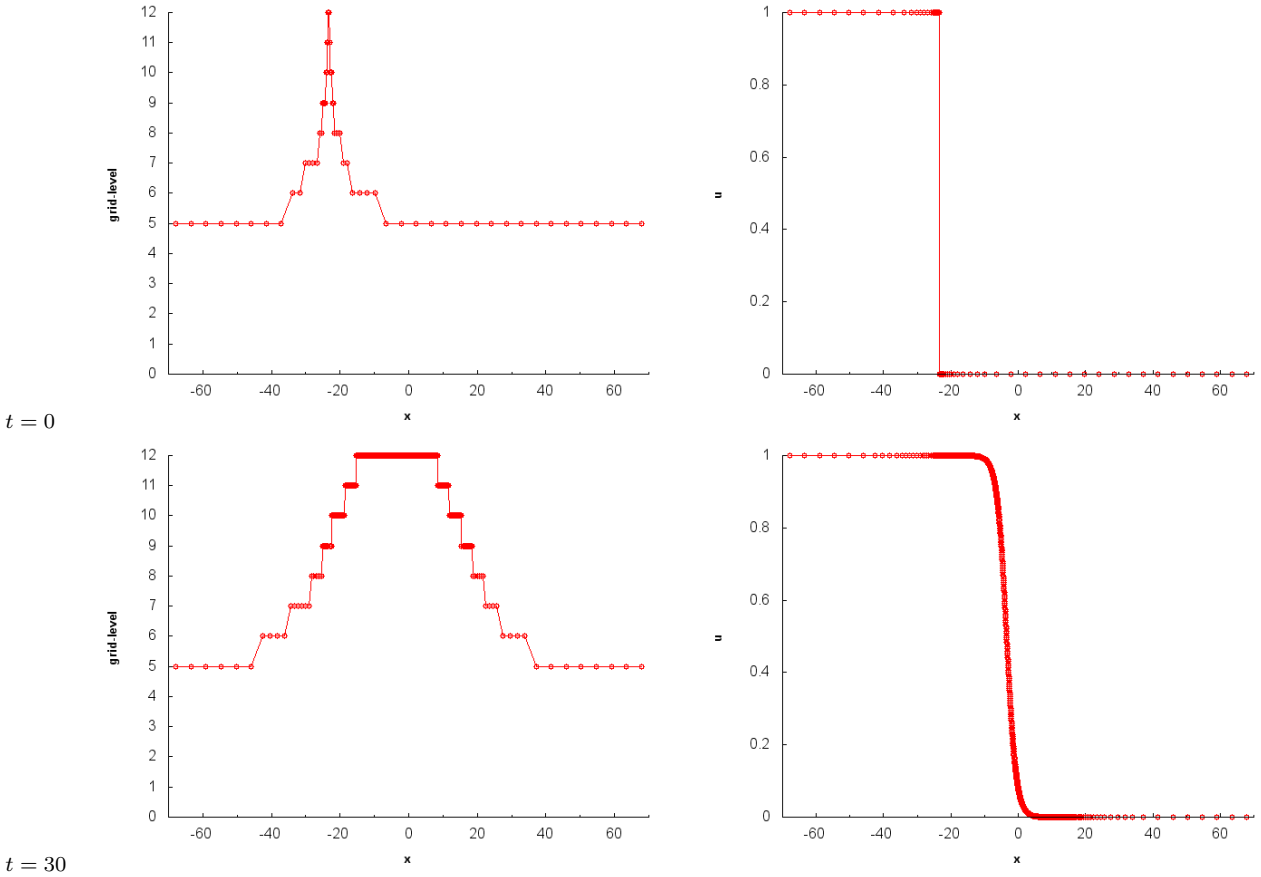


FIGURE 28. Standard KPP equation. L^2 errors (left) and compression rates (right) for different reconstruction accuracy and threshold values.



$t = 0$

$t = 30$

FIGURE 29. Standard KPP equation with discontinuous initial condition, with a reconstruction accuracy $o = 3$, $(s = 1)$ and a threshold value $\varepsilon = 10^{-8}$. Solutions were obtained using 12 grid-levels. On the left, grid levels and locations of leaves; on the right, the solution evaluated on leaves are plotted.

References

[1] A. Abdulle. Fourth order Chebyshev methods with recurrence relation. *SIAM J. Sci. Comput.*, 23(6):2041–2054 (electronic), 2002.

[2] N. A. Adams and K. Shariff. A high-resolution hybrid compact-eno scheme for shock-turbulence interaction problems. *J. Comp. Phys.*, 127:27–51, 1996.

[3] F. Aràndiga, R. Donat, and A. Harten. Multiresolution based on weighted averages of the hat function II: Non-linear reconstruction techniques. *SIAM J. Sci. Comput.*, 20(3):1053–1093, 1999.

[4] J. Bell, M. J. Berger, J. Saltzman, and M. Welcome. Three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Comput.*, 15:127–138, 1994.

[5] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:67–84, 1989.

[6] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.

[7] B. L. Bihari and A. Harten. Multiresolution schemes for the numerical solution of 2-D conservation laws I. *SIAM J. Sci. Comput.*, 18(2):315–354, 1997.

[8] F. Bramkamp, B. Gottschlich-Müller, M. Hesse, P. Lamby, S. Müller, J. Ballmann, K. H. Brakhage, and W. Dahmen. H-adaptive multiscale schemes for compressible Navier-Stokes equations - polyhedral discretization, data compression and mesh generation. In J. Ballmann, editor, *Flow modulation and fluid-structure-interaction at airplane wings*, volume 84 of *Numerical notes on Fluid Mechanics*, pages 125–204. Springer, 2003.

[9] F. Bramkamp, P. Lamby, and S. Müller. An adaptive multiscale finite volume solver for unsteady and steady flow computations. *J. Comp. Phys.*, 197(2):460–490, 2004.

[10] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Math. Comp.*, 31:333–390, 1977.

[11] A. Cohen. *Wavelet methods in numerical analysis*, volume 7 of *Handbook of Numerical Analysis*. P.G. Ciarlet and J.L. Lions, editors, Elsevier, Amsterdam, 2000.

[12] A. Cohen, I. Daubechies, and J. C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Comm. Pure Appl. Math.*, 45:485–560, 1992.

[13] A. Cohen, N. Dyn, S. M. Kaber, and M. Postel. Multiresolution finite volume schemes on triangles. *J. Comput. Phys.*, 161:264–286, 2000.

[14] A. Cohen, S. M. Kaber, S. Müller, and M. Postel. Fully adaptive multiresolution finite volume schemes for conservation laws. *Math. Comp.*, 72:183–225, 2003.

[15] V. Daru and C. Tenaud. High order one-step monotonicity preserving schemes for unsteady flow calculations. *Journal of Computational Physics*, 193:563–594, 2004.

[16] V. Daru and C. Tenaud. Numerical simulation of the viscous shock tube problem by using a high resolution monotonicity-preserving scheme. *Computers & Fluids*, 38:664–676, 2009.

[17] B. Gottschlich-Müller and S. Müller. Adaptive finite volume schemes for conservation laws based on local multiresolution techniques. In M. Fey and R. Jeltsch, editors, *Hyperbolic problems: Theory, numerics, applications*, pages 385–394. Birkhäuser, 1999.

[18] E. Hairer, C. Lubich, G. Wanner, and Lubich. *Geometric Numerical Integration:*. Springer Series in Computational Mathematics, Springer-Verlag, Berlin, second edition, 2006. Structure-Preserving Algorithms for Ordinary Differential Equations.

[19] E. Hairer and G. Wanner. *Solving ordinary differential equations II*. Springer-Verlag, Berlin, second edition, 1996. Stiff and differential-algebraic problems.

[20] A. Harten. Adaptive multiresolution schemes for shock computations. *Journal of Computational Physics*, 115:319–338, 1994.

[21] A. Harten. Multiresolution algorithms for the numerical solutions of hyperbolic conservation laws. *Communications on Pure and Applied Mathematics*, 48:1305–1342, 1995.

[22] D. J. Hill and D. I. Pullin. Hybrid tuned center-difference-weno method for large eddy simulations in the presence of strong shocks. *J. Comp. Phys.*, 194:435–450, 2004.

[23] G.-S. Jiang and C.-W. Shu. Efficient implementation of weighted eno schemes. *Journal of Computational Physics*, 126:202–228, 1996.

[24] D. Lax and B. Wendroff. Systems of conservation laws. *Communications on Pure and Applied Mathematics*, 13:217–237, 1960.

[25] R. J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhäuser, 2nd edition, 1992.

[26] P. Moin and K. Mahesh. Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30:539–578, 1998.

[27] S. Müller. *Adaptive multiscale schemes for conservation laws*, volume 27 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Heidelberg, 2003.

[28] M. Postel. Approximations multiéchelles. Ecole de printemps de mécanique des fluides numérique, Aussois, 2001.

[29] J. Qiu and C. W. Shu. Hermite weno schemes and their application as limiters for runge-kutta discontinuous galerkin method: one-dimensional case. *J. Comp. Phys.*, 193(1):115–135, 2004.

[30] Y. X. Ren, M. Liu, and H. Zhang. A characteristic-wise hybrid compact-weno scheme for solving hyperbolic conservation laws. *J. Comp. Phys.*, 192(2):365–386, 2003.

[31] O. Roussel, K. Schneider, A. Tsigulin, and H. Bockhorn. A conservative fully adaptive multiresolution algorithm for parabolic PDEs. *J. Comput. Phys.*, 188(2):493–523, 2003.

[32] C. W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservations laws. *NASA/CR-97-206253 and ICASE Report 97-65*, 1997.

[33] C. W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77:439–471, 1988.

[34] C. W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, II. *Journal of Computational Physics*, 83:32–78, 1989.

[35] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 5:506–517, 1968.

[36] A. Suresh and H. T. Huynh. Accurate monotonicity-preserving schemes with runge-kutta time stepping. *Journal of Computational Physics*, 136:83–99, 1997.

[37] V. A. Titarev and E. F. Toro. Ader: Arbitrary high order godunov approach. *Journal of Scientific Computing*, 17(1–4):609–618, 2002.

[38] A. Voss and S. Müller. A manual for the template class library *igpm_t_lib*. Technical Report 197, 1999.