1. Physical and computational motivation

1.1. **Singularities in physics simulation**

Adaptive Mesh Refinement proves particularly useful in simulations where different scales interact. For instance when an instability triggers a singularity, the large scale needed to simulate the global physical state is inadequate to catch the small scale of the singularity. And meshing right away the whole domain with a fine mesh is computationally too expansive.

This does not apply to stable situations like the Lid-driven Cavity, neither to homogeneous turbulence where small scales develop everywhere in the domain. AMR can be used to finely track a boundary or to capture the boundary layers.

1.2. **Numerical advantages and disadvantages of mesh refinement**

Here we discuss the pros end the cons of the Adaptive Mesh Refinement technique.

The use of an AMR demands specific numerical solvers and complicates the implementation and the parallelization. The loss of uniformity induces a loss of the error function regularity which allows to apply successive discrete operators still retaining the same order of accuracy. It makes the increase in the order of accuracy of the numerical schemes more difficult. For finite differences it also impacts the mass conservation. This problem may be solved using finite volume techniques. As the data structure is more complex, the computational time per point is multiplied by four or five.

Hence if a singularity occurs only in few percents of the simulation time or on few percents of the simulation domain then it may be advantageous to resort to AMR techniques. These techniques optimize the discretisation by reducing the number of degrees of freedom of the computation. In some occasions, such as the six-dimensional Vlasov equations, given the number of grid points, it is the only way to fit the simulation into the computer memory.

1.3. **Three examples in physics**

Thanks to AMR it was possible to simulate and better understand vortex singularity appearing in incompressible Euler equations [GMG98]. In this numerical experiment, the authors used a grid as fine as 2048^3 retaining only less than 1% of the points that would have been necessary for the equivalent uniform grid. In this paper they used a finite volume second-order upwinding Godunov scheme. In further works these authors preferred a central scheme which remains to Lax-Friedrichs type schemes and provides third-order accuracy in smooth regions using CWENO reconstruction.

In [KWA09], AMR applied to simulate the collision of galaxies provided the needed high dynamical ranges. A coarse grid at 128^3 can be refined up to 13 times, reaching scales of $\left(10^6\right)^3$ uniform grid equivalent which is out of reach of any existing computer.

The physical phenomenon of magnetic reconnection occurs in an ionized medium at the heart of magnetic confinement fusion plasmas but also at the interface between the solar wind and the terrestrial magnetosphere and explains the aurora borealis. This phenomenon results from a very localized magneto-hydrodynamic instability in space and time. To simulate this singularity, adaptative mesh refinement schemes allow very small scales to be reached that are inaccessible to uniform grid schemes.

However, the equations of the 2D reduced MHD models involve high order derivatives and require high order approximations, which are difficult to implement with adaptive schemes. The method presented in [DSD17] is based on finite differences, which poses conservation problems. To do this, we can rely on the wavelet approach, with interpolettes and interpolation wavelets, which are similar to the one adopted in [DSD17].

2. Finite volumes

2.1. From conservation laws to finite volumes

In this section, we briefly recall the basis of finite volumes schemes and some strategies to get a higher order of convergence. This class of numerical schemes is well suited to study conservation laws. A system of N conservation laws models the transport of N quantities of interest $\mathbf{u} = (u_1, ..., u_N)^T$ — also named conserved variables — in the space $\mathbb{R}^d$. The equation ruling the behavior of these quantities can be written in integral form:

$$\int_K \mathbf{u}(t_2, x)\,\mathrm{d}x - \int_K \mathbf{u}(t_1, x)\,\mathrm{d}x = -\int_{\partial K}\int_{t_1}^{t_2} \mathbf{f}(\mathbf{u}(t,s))\,\mathbf{n}_K(s)\,\mathrm{d}t\,\mathrm{d}s \qquad \forall K \subset \mathbb{R}^d,\ \forall (t_1, t_2) \in (0, +\infty)^2 \tag{1}$$

with $\mathbf{f} \in C^1(\mathbb{R}^N; \mathbb{R}^{N\times d})$ the analytic flux. When coupled to initial datum $\mathbf{u}^0 : \mathbb{R}^d \to \mathbb{R}^N$, equation (1) becomes equivalent to the following weak conservative formulation:

$$\begin{cases} \partial_t \mathbf{u} + \operatorname{div} \mathbf{f}(\mathbf{u}) = 0 & \text{in } \mathbb{R}_+ \times \mathbb{R}^d \\ \mathbf{u}(t=0, x) = \mathbf{u}^0(x) & \text{in } \mathbb{R}^d \end{cases} \tag{2}$$

In order to discretise this problem, we restrain ourselves to a bounded subspace $\Omega \subset \mathbb{R}^d$. We then have to complete the conservation laws with some boundary conditions. To stay in a simple setting, we consider Ω to be the unit cube and the boundary conditions are taken periodic. We introduce the uniform Cartesian mesh $\mathcal{T}$, and consider the averaged solution $\mathbf{U}_K^n$ over the cell $K \in \mathcal{T}$ at time t_n. The integral formulation (1) directly implies:

$$\mathbf{U}_K^{n+1} - \mathbf{U}_K^n = -\frac{1}{|K|}\int_{\partial K}\int_{t_1}^{t_2} \mathbf{f}(\mathbf{u}(t,s))\,\mathbf{n}_K(s)\,\mathrm{d}t\,\mathrm{d}s \tag{3}$$

The right hand side cannot be computed in the general case, and this is where an approximation has to be done. If we denote $\mathcal{E}(K)$ the set of interfaces of the cell K, and if we have an approximation $\mathbf{F}_{K,\sigma}$ of the averaged flux on each interface $\sigma \in \mathcal{E}(K)$ between times t_n and t_{n+1}, we get the so called finite volumes scheme:

$$\mathbf{U}_K^{n+1} - \mathbf{U}_K^n = -\frac{\Delta t_n}{|K|}\sum_{\sigma\in\mathcal{E}(K)} |\sigma|\,\mathbf{F}_{K,\sigma} \tag{4}$$

Thus the main stake when dealing with finite volumes schemes is to find a numerical flux that consistently approximates the right term from (3). A common approach to build such approximations is to consider the one-dimensional Riemann problem projected along the orthogonal direction to σ:

$$\begin{cases} \partial_t \mathbf{v} + \partial_x \mathbf{f}_\sigma(\mathbf{v}) = 0 & \text{in } \mathbb{R}_+ \times \mathbb{R} \\ \mathbf{v}(t=0, x) = 1_{x<0}\mathbf{U}_L + 1_{x\geq 0}\mathbf{U}_R & \text{in } \mathbb{R} \end{cases} \tag{5}$$

where $\mathbf{U}_L, \mathbf{U}_R$ are values of the solution on both sides of the interface σ, and $\mathbf{f}_\sigma(\mathbf{u}) = \mathbf{f}(\mathbf{u})\,\mathbf{n}_\sigma$. Solutions of the Riemann problem depend among other things on the eigenvalues $(\lambda_i)_{1\leq i\leq N}$ of $\operatorname{Jac}\mathbf{f}_\sigma$, which is why the numerical flux $\mathbf{F}_{K,\sigma}$ itself often depends on them (see table 1 for examples of common numerical fluxes).

2.2. High order finite volumes schemes

Before moving to the next topic, we address the issue of achieving high order convergence with finite volumes schemes. A first approach would be to draw on the formalism of finite differences schemes. In fact, in the case of smooth enough solutions the latter can reach an arbitrary high order by the mean of Taylor expansions. The idea is then to try to design stable finite differences schemes that can be rewritten under conservative form (4)

TABLE 1. Some well-known numerical fluxes.

	Numerical flux $\mathbf{F}_{K,\sigma}$	*Order*
Godunov	$\mathbf{f}_\sigma(\mathbf{u}^\star(x/t=0))$, $\mathbf{u}^\star$ self-similar solution of Riemann problem (5)	$\mathcal{O}(\Delta t, \Delta x)$
Rusanov	$[\mathbf{f}_\sigma(\mathbf{U}_L) + \mathbf{f}_\sigma(\mathbf{U}_R) - \lvert a\rvert(\mathbf{U}_R - \mathbf{U}_L)]/2$, $a = \max_i(\lvert\lambda_i\rvert)$	$\mathcal{O}(\Delta t, \Delta x)$
HLL	$[\lambda_L\lambda_R(\mathbf{U}_R - \mathbf{U}_L) + \lambda_R\mathbf{f}_\sigma(\mathbf{U}_L) - \lambda_L\mathbf{f}_\sigma(\mathbf{U}_R)]/[\lambda_R - \lambda_L]$, $\lambda_L \leq \lambda_{\min}$ and $\lambda_R \geq \lambda_{\max}$	$\mathcal{O}(\Delta t, \Delta x)$

by substituting the point-wise values with cell averages. A simple example is given by the Lax-Wendroff scheme. Consider u the solution of the 1D scalar linear transport with constant velocity c, and $(x_i)_i$ the equidistant cells centers of some domain discretisation. A first Taylor expansion in time of the solution writes:

$$u(t^{n+1}, x) = u(t^n, x) + \Delta t u_t(t^n, x) + \frac{\Delta t^2}{2} u_{tt}(t^n, x) + \mathcal{O}(\Delta t^3) \tag{6}$$

Using the relations $u_t = -cu_x$ and $u_{tt} = c^2 u_{xx}$, one can replace the time derivatives in (6) by second order Taylor expansions in space:

$$\begin{aligned} u(t^{n+1}, x_i) = u(t^n, x_i) - c\Delta t \frac{u(t^n, x_{i+1}) - u(t^n, x_{i-1})}{2\Delta x} + \frac{c^2\Delta t^2}{2} \frac{u(t^n, x_{i-1}) - 2u(t^n, x_i) + u(t^n, x_{i+1})}{\Delta x^2} \\ + \mathcal{O}(\Delta t^3, \Delta x^3) \end{aligned} \tag{7}$$

Relation (7) thus gives the second order finite differences scheme:

$$u_i^{n+1} = u_i^n - c\Delta t \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + \frac{c^2\Delta t^2}{2} \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} \tag{8}$$

where u_i^n stands for the point-wise approximation of the solution at position x_i and time t^n. Note that the last term in (8) adds numerical viscosity, and removing it would result in an unconditionally unstable centered scheme. Substituting u_i^n by the average $\bar{u}_i^n$ over cell $C_i = [x_{i-1/2}, x_{i+1/2}]$, this scheme can finally be rewritten under conservative form $\overline{u}_i^{n+1} = \overline{u}_i^n - \frac{\Delta t}{\Delta x}(F(\overline{u}_i^n, \overline{u}_{i+1}^n) - F(\overline{u}_{i-1}^n, \overline{u}_i^n))$. Here we have chosen:

$$F(U_L, U_R) = \frac{c}{2}(U_R + U_L) - \frac{c^2}{2}\frac{\Delta t}{\Delta x}(U_R - U_L)$$

for the numerical flux. Hence the Lax-Wendroff method can be interpreted at the same time as a finite volumes and a finite differences scheme, and is of order 2 both in time and space. The obvious issue with this example is that in a more general case, it is not always possible to rewrite a given finite differences scheme under conservative form. Furthermore, this approach is based on an assumption of smoothness, which is not necessarily true. More specifically, the Lax-Wendroff scheme is known to have a highly oscillatory behavior for solutions with discontinuities.

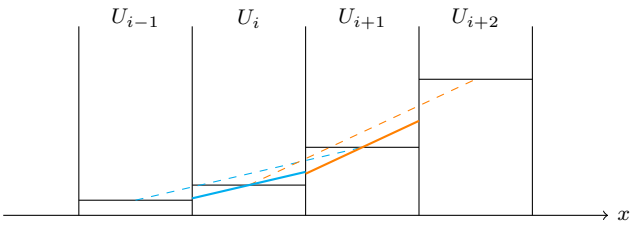


FIGURE 1. MUSCL reconstruction using centered slopes.

Alternatively, one could try to improve the order of convergence through the use of a reconstruction operator. The most celebrated one is the MUSCL reconstruction first introduced by van Leer in [VL79], and which consists in replacing the piece-wise constant solution by a piece-wise linear approximation $\hat{u}_i^n(x) := u_i^n + \sigma_i^n(x - x_i)$ with σ_i^n a slope determined by some "interface differencing" discussed below. This way, the numerical flux benefits from more accurate left and right interface limits (9) induced by the slopes σ_i^n, leading to a second order convergence in space if the original scheme is of order one.

$$\hat{u}_{i+1/2-}^n = \hat{u}_i^n(x_i + \Delta x/2), \qquad \hat{u}_{i-1/2+}^n = \hat{u}_i^n(x_i - \Delta x/2) \tag{9}$$

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(F(\hat{u}_{i+1/2-}^n, \hat{u}_{i+1/2+}^n) - F(\hat{u}_{i-1/2-}^n, \hat{u}_{i-1/2+}^n)) \tag{10}$$

The update (10) is then replaced by a Runge-Kutta time integration to get second order both in time and in space. As for the choice of the slope, it should somehow approximate the gradient of the solution, since otherwise the reconstruction wouldn't be of order two. For instance, one could pick the centered slope $\sigma_i^n = (u_{i+1}^n - u_{i-1}^n)/2\Delta x$ (see fig. 1). A clear advantage of the MUSCL method is that it allows to increase accuracy of any first order scheme without changing the numerical flux, and is thus very easy to implement.

Going even further, one can replace the piece-wise constant solution by a piece-wise second order polynomial $\hat{u}_i^n(x) = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2$, as studied in [vT09]. This reconstruction generates a third order scheme in space if $\hat{u}_i^n$ preserves the averaged solution u_i^n, and if it approximates space derivatives of the solution at $x_{i-1/2}$ and $x_{i+1/2}$ with an error of at most $\mathcal{O}(\Delta x^2)$. This translates into the following system:

$$\left\{\begin{array}{l} \dfrac{1}{\Delta x}\displaystyle\int_{x_{i-1/2}}^{x_{i+1/2}} \hat{u}_i^n(x)\,\mathrm{d}x = u_i^n \\ (\hat{u}_i^n)'(x_{i-1/2}) = \dfrac{u_{i+1}^n - u_i^n}{\Delta x} = \partial_x u(t^n, x_{i-1/2}) + \mathcal{O}(\Delta x^2) \\ (\hat{u}_i^n)'(x_{i+1/2}) = \dfrac{u_i^n - u_{i-1}^n}{\Delta x} = \partial_x u(t^n, x_{i+1/2}) + \mathcal{O}(\Delta x^2) \end{array}\right. \quad \Longrightarrow \quad \left\{\begin{array}{l} a_i + \dfrac{\Delta x^2}{24}c_i = u_i^n \\ b_i - \dfrac{\Delta x}{2}c_i = \dfrac{u_i^n - u_{i-1}^n}{\Delta x} \\ b_i + \dfrac{\Delta x}{2}c_i = \dfrac{u_{i+1}^n - u_i^n}{\Delta x} \end{array}\right.$$

Solving this system, we get:

$$a_i = \frac{26u_i^n - u_{i-1}^n - u_{i+1}^n}{24}, \qquad b_i = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}, \qquad c_i = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

Finally we have :

$$\left\{\begin{array}{l} \hat{u}_{i+1/2-}^n = \hat{u}_i^n(x_{i+1/2}) = \dfrac{5}{6}u_i^n - \dfrac{1}{6}u_{i-1}^n + \dfrac{1}{3}u_{i+1}^n \\ \hat{u}_{i-1/2+}^n = \hat{u}_i^n(x_{i-1/2}) = \dfrac{5}{6}u_i^n - \dfrac{1}{6}u_{i+1}^n + \dfrac{1}{3}u_{i-1}^n \end{array}\right. \tag{11}$$

Interestingly, the stencil of the third order reconstruction operator (11) is of size three, which is the same as for the MUSCL reconstruction, yet the former is more precise.

3. Adaptive mesh refinement framework

As mentioned in the opening of this paper, there is need for a fine enough grid in order to accurately catch fine details of the solution. Given that the scale of the simulation is likely to be too large in comparison to the "scales of interest", one cannot afford to use a uniform mesh. In this section, we present some issues concerning non-uniform grids from the algorithmic point of view.

3.1. **Representing a non-uniform grid**

The main goal is to discretise a space domain of dimension $d \in \mathbb{N}$ into a non-uniform mesh made of superimposed grids. For the sake of simplicity, we limit ourselves to the unit cube of dimension d, and consider an initially uniform grid. Each cell of the initial grid can be subdivided into 2^d equally sized sub-cells which contribute to a first level of refinement. Of course those sub-cells themselves can be split in a recursive manner, leading to greater levels of refinements. All these levels can be seen as "layers" applied to the initial grid, each added layer being twice as refined as the previous one. A convenient way of representing this

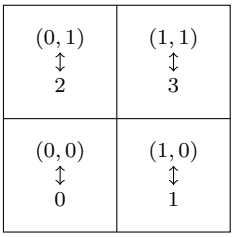


FIGURE 2. Illustration of a node and its 2^d cells for $d = 2$. Note that the cells can be referenced inside the node by their coordinates or the associated binary dyadic combination.

structure is to use a tree-based representation. The tree connects nodes from consecutive levels between each other, every node resulting from the gathering of 2^d cells. Inside a node, cells can be referenced either using their coordinates $\mathbf{i} := (i_0, i_1, ..., i_{d-1}) \in \{0, 1\}^d$, or using the binary dyadic combination of their coordinates $\alpha(\mathbf{i}) := i_0 2^0 + i_1 2^1 + ... + i_{d-1} 2^{d-1}$ (see fig. 2 for an example in dimension 2). When a cell is divided for refinement, it becomes a node with its proper cells playing the role of leaves.

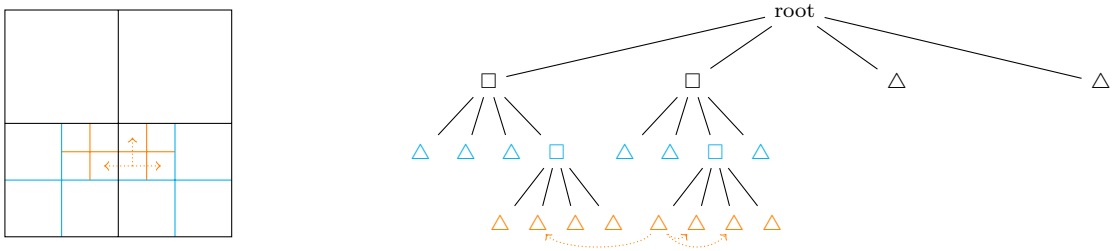


FIGURE 3. Tree-based representation (right) of superimposed grids (left) in dimension $d = 2$. In the tree, nodes are represented by squares, while leaves (non-divided cells) are represented by triangles. The dotted arrows give an example of neighborhood for a specific cell.

The advantage of the tree structure lies in the fact that it gives a straightforward way to navigate in the node hierarchy: from a given node, it is easy to visit its parent or its children. However, what is less trivial is to access nodes at the same level, for instance if one wants to iterate over a cell's neighbors (see fig. 3). Let us show an example of how it can be achieved. For instance, suppose that we have a cell $C_{\mathbf{i}}$ belonging to the node $\mathcal{N}$ and that we want to access its neighbor at relative position $l := 2k + j$ where $k \in \{0, 1, ..., d-1\}$ is the direction and $j \in \{0, 1\}$ is the sense along that direction (positive if $j = 0$, negative otherwise). First question that one may ask: is the neighbor cell we are looking for in the same node or in a neighbor node ? The answer is given by the quantity $|j - p_k(\mathbf{i})|$ with $p_k(\mathbf{i}) := (\alpha(\mathbf{i})/2^k) \bmod 2$. The function p_k can be interpreted as a "parity" argument taking the value 0 if the cell $C_{\mathbf{i}}$ is at the left along direction k inside the node, and 1 if it is at the right along the same direction inside the node. Thus we see that the neighbor sought after is in the same node if and only if $|j - p_k(\mathbf{i})|$ is equal to 0. In that case, the neighbor cell is $C_{\mathbf{i}'} \in \mathcal{N}$ with $\mathbf{i}'$ such that

$\alpha(\mathbf{i}') = \alpha(\mathbf{i}) + 2^k(1-2j)$. In the opposite case, we have to recursively apply the algorithm to the node $\mathcal{N}$ itself in order to find its own neighbor $\mathcal{N}'$ in the relative position $l = 2k + j$. Once this is done, the neighbor cell $C_{\mathbf{i}'} \in \mathcal{N}'$ is given by the coordinates $\alpha(\mathbf{i}') = \alpha(\mathbf{i}) - 2^k(1-2j)$.

A constraint that will be applied from now on is that there can be no direct transitions between non-consecutive levels of refinement. As a consequence, the example given through figure 3 will in reality not be authorized. Moreover in the case of periodic boundary conditions, we shall consider the root node to be its own neighbor.

3.2. **Volume interpolation**

When refined, a coarse cell becomes a node and it is necessary to attribute accurate values to its children. Knowing the averaged values of the solution in coarser cells next to the refined one, we can approximate the average of the solution in the new sub-cells via volume interpolation. In what follows we give an example of third order volume interpolation in 1D. Let $\mu_{[a,b]} : \mathbb{R}[X] \mapsto \mathbb{R}$ be the application mapping a polynomial to its average over the real interval $[a, b]$. We are looking for values $(\alpha, \beta, \gamma) \in \mathbb{R}^3$ such that:

$$\forall P \in \mathbb{R}_2[X], \quad \mu_{[0,1]}(P) = \alpha\mu_{[-2,0]}(P) + \beta\mu_{[0,2]}(P) + \gamma\mu_{[2,4]}(P) \tag{12}$$

Evaluating this equality for polynomials $1, X, X^2$, we get the following system:

$$\begin{cases} \alpha + \beta + \gamma = 1 \\ -\alpha + \beta + 3\gamma = 1/2 \\ 4\alpha + 4\beta + 28\gamma = 1 \end{cases} \Longrightarrow \begin{cases} \alpha = 1/8 \\ \beta = 1 \\ \gamma = -1/8 \end{cases}$$

If instead we wanted to interpolate $\mu_{[1,2]}(P)$, we would have considered the values $(\alpha'\beta', \gamma') \in \mathbb{R}^3$ such that:

$$\forall P \in \mathbb{R}_2[X], \quad \mu_{[1,2]}(P) = \alpha'\mu_{[-2,0]}(P) + \beta'\mu_{[0,2]}(P) + \gamma'\mu_{[2,4]}(P) \tag{13}$$

By symmetry with the previous interpolation, we get $\alpha' = -1/8$, $\beta' = 1$ and $\gamma' = 1/8$.

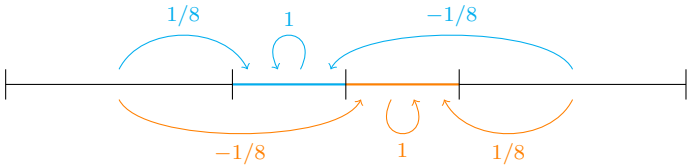


FIGURE 4. Interpolation computed from the point of view of the sub-cells.

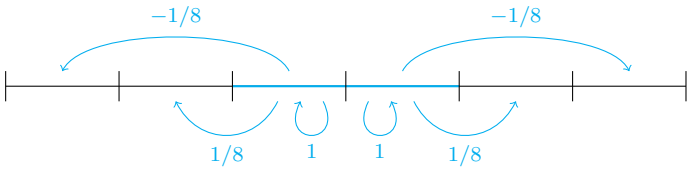


FIGURE 5. Interpolation computed in terms of a coarse cell sending its contributions to the sub-cells.

This volume interpolation is computed from the point of view of the new sub-cells: for each such cell, we gather the values of the coarse cells around to compute either (12) or (13). This might be inefficient since

we have to retrieve several times the same values at the coarse level, as shown in fig. 4. Alternatively, we could adopt another strategy illustrated through fig. 5 by iterating over the coarse cells and "diffusing" their contributions to the new sub-cells. In 1D it is equivalent, however this approach becomes more efficient than the former one at higher dimensions. In fact, suppose that we have a 2D mesh where several cells have been refined in sub-cells. A third order volume interpolation can be derived from the 1D one by performing a tensor product of the stencil presented in figure 5.

$-1/8$	$1/8$	1	1	$1/8$	$-1/8$	
$1/64$	$-1/64$	$-1/8$	$-1/8$	$-1/64$	$1/64$	$-1/8$
$-1/64$	$1/64$	$1/8$	$1/8$	$1/64$	$-1/64$	$1/8$
$-1/8$	$1/8$	1	1	$1/8$	$-1/8$	1
$-1/8$	$1/8$	1	1	$1/8$	$-1/8$	1
$-1/64$	$1/64$	$1/8$	$1/8$	$1/64$	$-1/64$	$1/8$
$1/64$	$-1/64$	$-1/8$	$-1/8$	$-1/64$	$1/64$	$-1/8$

FIGURE 6. Weights of the third order volume interpolation in 2D. The bold square at the center represent a coarse cell sending its contributions. First, we take into account the central contributions (in gray). Then we send the contributions to the blue sub-cells, then to the orange ones (or the other way around).

An efficient way of completing the interpolation is to first send the contributions with weights 1 to the children. Once this is done for all coarse nodes, we start over again by sending the contributions horizontally (those with weights $\pm 1/8$) and accumulate them with the previous contributions. Finally, we proceed similarly in the vertical direction. Note that exchanging the last two steps would lead to the same result. Here the diagonal sub-cells have been omitted because the contributions with weights $\pm 1/64$ are indirectly handled. In fact contributions are accumulated when combining both horizontal and vertical directions (see fig. 6). The general procedure in dimension d is given in algorithm 1.

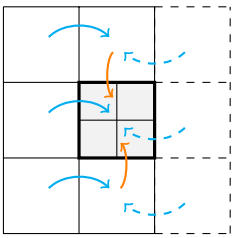


FIGURE 7. At the border some coarse nodes are missing: instead there are undivided cells represented by the dashed squares. One has to replace them by ghost nodes in order to take into account all contributions.

Algorithm 1: Third order volume interpolation of the level of refinement $n+1$ in dimension d

Each cell has an attribute `rc` (received contribution) and `rdc` (received directional contribution) initialized to zero;
for *node* $\mathcal{N}$ *in* n^{th} *level of refinement* **do**
  for *cell* $C_{\mathbf{i}}$ *in node* $\mathcal{N}$ **do**
    $C_{\mathbf{i}}^{\texttt{rc}} \leftarrow$ contribution of node $\mathcal{N}$ (weight 1);
for *direction* $k \in \{0, ..., d-1\}$ **do**
  for *node* $\mathcal{N}$ *in* n^{th} *level of refinement* **do**
    $\mathcal{N}_{\text{left}} \leftarrow$ neighbor of $\mathcal{N}$ at relative position $2k+1$;
    $\mathcal{N}_{\text{right}} \leftarrow$ neighbor of $\mathcal{N}$ at relative position $2k$;
    for *cell* $C_{\mathbf{i}}$ *in node* $\mathcal{N}$ **do**
      $K_{\mathbf{i}} \leftarrow$ cell of coordinate $\mathbf{i}$ in node $\mathcal{N}_{\text{left}}$, $L_{\mathbf{i}} \leftarrow$ cell of coordinate $\mathbf{i}$ in node $\mathcal{N}_{\text{right}}$;
      if $p_k(\mathbf{i}) = 0$ **then**
        $K_{\mathbf{i}}^{\texttt{rdc}} \leftarrow K_{\mathbf{i}}^{\texttt{rdc}} - C_{\mathbf{i}}^{\texttt{rc}}/8, \quad L_{\mathbf{i}}^{\texttt{rdc}} \leftarrow L_{\mathbf{i}}^{\texttt{rdc}} + C_{\mathbf{i}}^{\texttt{rc}}/8;$
      else
        $K_{\mathbf{i}}^{\texttt{rdc}} \leftarrow K_{\mathbf{i}}^{\texttt{rdc}} + C_{\mathbf{i}}^{\texttt{rc}}/8, \quad L_{\mathbf{i}}^{\texttt{rdc}} \leftarrow L_{\mathbf{i}}^{\texttt{rdc}} - C_{\mathbf{i}}^{\texttt{rc}}/8;$
  For each cell, add the value of `rdc` to `rc` and initialize `rdc` to zero;

A last detail we want to discuss relatively to interpolation is the handling of border cells. For them to be correctly interpolated, in the case of the third order interpolation we would have to iterate at the coarser level over 3^d nodes (the interpolation stencil). However, since the considered sub-cells are at the border of their level of refinement, it means that some of these nodes do not exist, in place of what there are undivided coarse cells (leaves at the coarser level). This is problematic because it prevents us from sending the accumulated contributions to the sub-cells as in algorithm 1. A simple solution is to virtually refine the coarse leaves to get the missing nodes: we call them ghost nodes.

3.3. **Brief overview of adaptivity criteria**

The adaptivity criteria directly derives from the interpolation process. Subtracting the interpolated value of a cell to its actual value, we are able to quantify exactly the information that it contains.

For instance applying the interpolation presented in Section 3.2, the detail stored in $\mu_{[1,2]}(f)$ is given by

$$d_{[1,2]}(f) = \mu_{[1,2]}(f) - \frac{1}{8}\left(-\mu_{[-2,0]}(f) + 8\mu_{[0,2]}(f) + \mu_{[2,4]}(f)\right).$$

If this detail is small enough, it can be removed. If it is large enough, we trigger a refinement process redividing the cell into new smaller cells.

3.4. **Finite volumes schemes on a non-uniform mesh**

In this section, we propose an adaptation of finite volumes schemes to non-uniform grids. In order to do so, we first concentrate our attention on the main difference with the uniform case, which is the handling of fluxes at the boundary of a level of refinement. Support nodes are introduced to this end, after what we describe the communication process between levels of refinement allowing to compute the fluxes.

Given a refined mesh, we want to update the solution using a finite volumes scheme. As such, we need to iterate over the interfaces of each level of refinement to compute the fluxes. The problem we are confronted to is that some interfaces are located at the border of a level of refinement, meaning that they are not separating

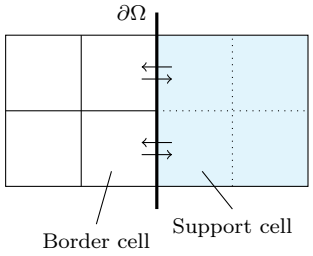


FIGURE 8. To be computed, some numerical fluxes require support cells. Here $\partial\Omega$ is the border of the refinement level. The blue area corresponds to a coarse cell that is being divided to get support cells at the finer level. After interpolation of those new cells, we are able to compute the numerical fluxes at the fine borders.

two fine cells but a fine cell and a coarse one (see fig. 8). A way to avoid this issue is to split the coarse cell into support cells that are then interpolated. As a result, we are able to compute the flux associated to this interface as usually. Note that this process is not properly speaking a refinement, since the support cells will not be updated — their only goal being to make it possible to compute the flux — and are due to be removed at the end of every iteration. As described in section 3.2, interpolating the support cells might require the introduction of ghost nodes according to the interpolation stencil.

Algorithm 2: Updating the solution over an adaptive non-uniform mesh

```
According to some criterion, refine and coarsen the cells requiring it;
Generate support cells at the border of each level of refinement;
Generate ghost nodes;
Interpolate the newly refined cells and support cells;
Compute a time step Δt verifying the CFL condition at every level of refinement to ensure stability;
Initialize current level of refinement to the finest one;
while coarsest level of refinement not reached do
    I ← gathering of current level's interfaces involving at least one "classic" cell (not two support cells);
    for interface in I do
        if interface has no sub-interfaces then neighboring cells are leaves
            Compute numerical flux F(U_L^n, U_R^n) using an approximate Riemann solver;
        else at least one of the neighboring cells is a node
            Using (14), set the numerical flux as the average of finer fluxes;
    for cell in current level of refinement do
        Update solution inside the cell using a time integrator;
    Go to next level of refinement;
Remove support cells and ghost nodes;
Increment current time by Δt;
```

The next step is to effectively compute the fluxes while taking into account the links between consecutive levels of refinement. In fact, what is an interface at a coarse level will be the collection of 2^{d-1} sub-interfaces at the finer level. In order to ensure the conservativity of the scheme, a necessary constraint is that the total flux contribution from each of the sub-interfaces should be equal to the flux contribution of the coarse interface. More precisely, we show that the coarse flux needs to be equal to the average of the sub-fluxes. Let K be a node

(refined cell) with children $(C_{\mathbf{i}})_{\mathbf{i}\in\{0,1\}^d}$ and L a non-refined cell such that K and L are neighbors. Let $\mathcal{J}(K,L)$ denote the set of indices corresponding to the children of K sharing an interface with L: for all $\mathbf{j}\in\mathcal{J}(K,L)$, the interface $C_{\mathbf{j}}\,|\,L$ exists and is at the border of a level of refinement. According to the previous paragraph, the fluxes $\mathbf{F}_{C_{\mathbf{j}}|L}$ can be determined through support cells derived from L. The coarse flux $\mathbf{F}_{K|L}$ can then be related to these finer fluxes by considering the case where K and L form a confined system (no exchange with the exterior). In fact the discrete conservativity of total mass reads:

$$\begin{aligned}
&\mathrm{vol}(\Delta x)[\mathbf{U}_K^{n+1}-\mathbf{U}_K^n] = -\mathrm{vol}(\Delta x)[\mathbf{U}_L^{n+1}-\mathbf{U}_L^n]\\
\Longrightarrow\ &\mathrm{vol}(\Delta x)\,2^{-d}\sum_{\mathbf{i}\in\{0,1\}^d}\left[\mathbf{U}_{C_{\mathbf{i}}}^{n+1}-\mathbf{U}_{C_{\mathbf{i}}}^n\right] = -\mathrm{vol}(\Delta x)[\mathbf{U}_L^{n+1}-\mathbf{U}_L^n]\\
\Longrightarrow\ &\Big(\frac{\Delta x}{2}\Big)^d\frac{\Delta t}{\Delta x/2}\sum_{\mathbf{j}\in\mathcal{J}(K,L)}\mathbf{F}_{C_{\mathbf{j}}|L} = \Delta x^d\,\frac{\Delta t}{\Delta x}\,\mathbf{F}_{K|L}\\
\Longrightarrow\ &\mathbf{F}_{K|L} = 2^{1-d}\sum_{\mathbf{j}\in\mathcal{J}(K,L)}\mathbf{F}_{C_{\mathbf{j}}|L}
\end{aligned}\tag{14}$$

The strategy we adopt here is to always compute the fluxes from the finest level to the coarsest level, since we might have to compute the average flux of sub-interfaces to get the coarse flux. The communication process occurs while iterating over interfaces which admit sub-interfaces, and the overall procedure is summarized in algorithm 2.

4. Implementation and results

4.1. Visualization on unstructured grids

Previously discussed algorithms have been implemented in `C`. The whole implementation is based on a structure named `Node` presented in Table 2. This structure gathers all the physical values of the simulation in a field `**tab` and connects to the other nodes in order to form the tree structure. The node at the top of the tree is called the root. Following the parent-to-children links it gives access to the whole tree.

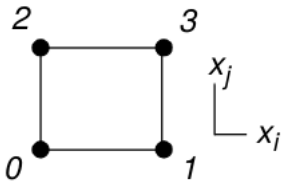
Table 2. The Node: basic element of the tree structure

```
struct node {
    double **tab;       /* 2^dim cells referenced by dyadic writing */
    struct noeud *prt;  /* parent */
    struct noeud **chd; /* children (0,1,...,2^dim - 1) */
    struct noeud **ngb; /* neighbors (0,1,...,2*dim - 1) */
    int frk;            /* filiation rank */
    int flag;           /* flag indicating which nodes are active/support/ghost */
};
```

We use VTK-File-Format for visualization purposes. Unstructured grids are defined by points, cells, and cell types. The `CELLS` keyword requires two parameters: the number of cells n and the size of the cell list size. The cell list size is the total number of integer values required to represent the list (i.e., sum of *numPoints* and connectivity indices over each cell). The `CELLS_TYPES` keyword requires a single parameter: the number of cells n. This value should match the value specified by the `CELLS` keyword. The cell types data is a single integer value per cell that specifies cell type. In 2D, we choose for all cells `VTK_PIXEL`, see figure 9.

DATASET UNSTRUCTURED_GRID
POINTS n *dataType*
p_{0x}, p_{0y}, p_{0z},
p_{1x}, p_{1y}, p_{1z},
...
$p_{(n-1)x}$, $p_{(n-1)y}$, $p_{(n-1)z}$.

CELLS n *size*
$numPointss_0$, i, j, k, l, ...,
$numPointss_1$, i, j, k, l, ...,
...
$numPointss_{n-1}$, i, j, k, l, ...



VTK_PIXEL (=8)

FIGURE 9. Cell type used for all cells in 2D.

TABLE 3. Points and cells of the unstructured grid

```
struct points_coord {
 double x_coord;
 double y_coord;
 double z_coord;
};
```

```
struct cells_connectivity {
  int total;
  int a;
  int b;
  int c;
  int d;
};
```

The implementation is based on two structures named `points_coord` and `cells_connectivity`. From the representation in Figure 10, we implement the following:

```
/* inside a loop on all nodes */
/* (xpos,ypos): coordinates of bottom left corner of the four cells associated to */
/* current node */
double dx; // dx : size of one cell
j=0;
for (i=0; i<9; ++i) {
  if (i%3 == 0) {  /* every three integer we go one row up */
    j+=1;
  }
  point_matrix[point_counter+i].x_coord = xpos+(i%3)*dx;
  point_matrix[point_counter+i].y_coord = ypos+(j-1)*dx;
  point_matrix[point_counter+i].z_coord = 0.;
}
j=0;
int aa = point_counter;   /* bottom left  */
int bb = point_counter+1; /* bottom right */
int cc = point_counter+4; /* top    right */
int dd = point_counter+3; /* top    left  */

for (i=0; i<4; ++i) {
  if (i%2 == 0){ /*every two integer we go one row up */
    j+=1;
  }
  cell_matrix[cell_counter+i].total=4;
```

```
  cell_matrix[cell_counter+i].a = aa + i%2 + (j-1)*3;
  cell_matrix[cell_counter+i].b = bb + i%2 + (j-1)*3;
  cell_matrix[cell_counter+i].c = cc + i%2 + (j-1)*3;
  cell_matrix[cell_counter+i].d = dd + i%2 + (j-1)*3;
}
```

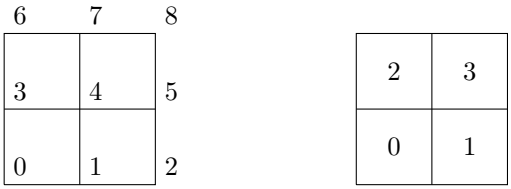


FIGURE 10. Numbering of points (left) and cells (right) for each node.

4.2. **Numerical results**

Numerical results based on our program are now presented. A first test-case is implemented to assess the order of convergence of the MUSCL and third order reconstructions. To this end, we consider the 1D scalar linear transport equation associated to the C^∞ initial datum $u^0(x) := 1_{[1/4,3/4]}(x) \exp\left(-\frac{1}{1-16(x-1/2)^2}\right)$:

$$\begin{cases} \partial_t u(t,x) + v\,\partial_x u(t,x) = 0 \\ u(t=0,x) = u^0(x) \end{cases}$$

The velocity v is taken equal to $3/4$. The mesh we use is made of an initial grid and two additional levels of refinement, and stays fixed throughout the simulation (no adaptivity). The initial condition is interpolated from the initial grid towards the finer levels. We also compare the results to the case of uniform meshes. We choose the Rusanov numerical flux, and the Runge-Kutta 4 method is used for time integration, the stability being ensured by taking $\Delta t = \text{CFL} \times \Delta x_f/(2v)$ with Δx_f the cell size of the finest level of refinement and $\text{CFL} = 0.45$. The simulation is stopped at final time $T = 73/15$. We compute the L^2-error at the coarsest level at time T, and we do so for initial grids ranging from depth 4 to depth 10. Results are gathered in tables 4 to 6.

TABLE 4. RK4 + no reconstruction

Base depth	$\log(L^2\text{-ERR})/\log(2)$	
	Nonuniform mesh	*Uniform mesh*
4	-5.712382	5.415895
5	-6.626516	-6.197763
6	-7.634907	-7.110404
7	-8.767312	-8.136759
8	-9.950684	-9.261226
9	-11.179305	-10.442954
10	–	-11.669745

TABLE 5. RK4 + MUSCL reconstruction

Base depth	$\log(L^2\text{-ERR})/\log(2)$	
	Nonuniform mesh	*Uniform mesh*
4	-8.022995	-7.151959
5	-9.579418	-8.637820
6	-11.244609	-10.203497
7	-13.167167	-11.954795
8	-15.284343	-13.917454
9	-17.591171	-16.094520
10	–	-18.447992

The order of convergence can be easily computed thanks to this table. In fact, a method of order at least $k \in \mathbb{R}_+$ is characterized by the inequality $e_{n+1} \leq 2^{-k} \times e_n$, where e_n denotes the error obtained on an initial grid at depth n (also called "base depth") with possible levels of refinement on top of it, assuming n is big enough.

TABLE 6. RK4 + third order reconstruction

Base depth	$\log(L^2\text{-ERR})/\log(2)$	
	Nonuniform mesh	*Uniform mesh*
4	-8.632480	-7.971418
5	-10.506254	-9.581725
6	-12.548737	-11.507401
7	-14.970157	-13.711773
8	-17.708034	-16.237445
9	-20.774341	-19.103689
10	-24.089038	-22.278377
11	–	-25.661788

Thus, we see that k verifies $k \leq (\log e_n - \log e_{n+1})/\log 2$. As expected, finite volumes without reconstruction are of order at least one, whereas with MUSCL reconstruction (resp. third order reconstruction) they are of order at least two (resp. at least three) no matter if the mesh is uniform or not. Without much surprise, the order three reconstruction is the most accurate and has a compact stencil of size three.

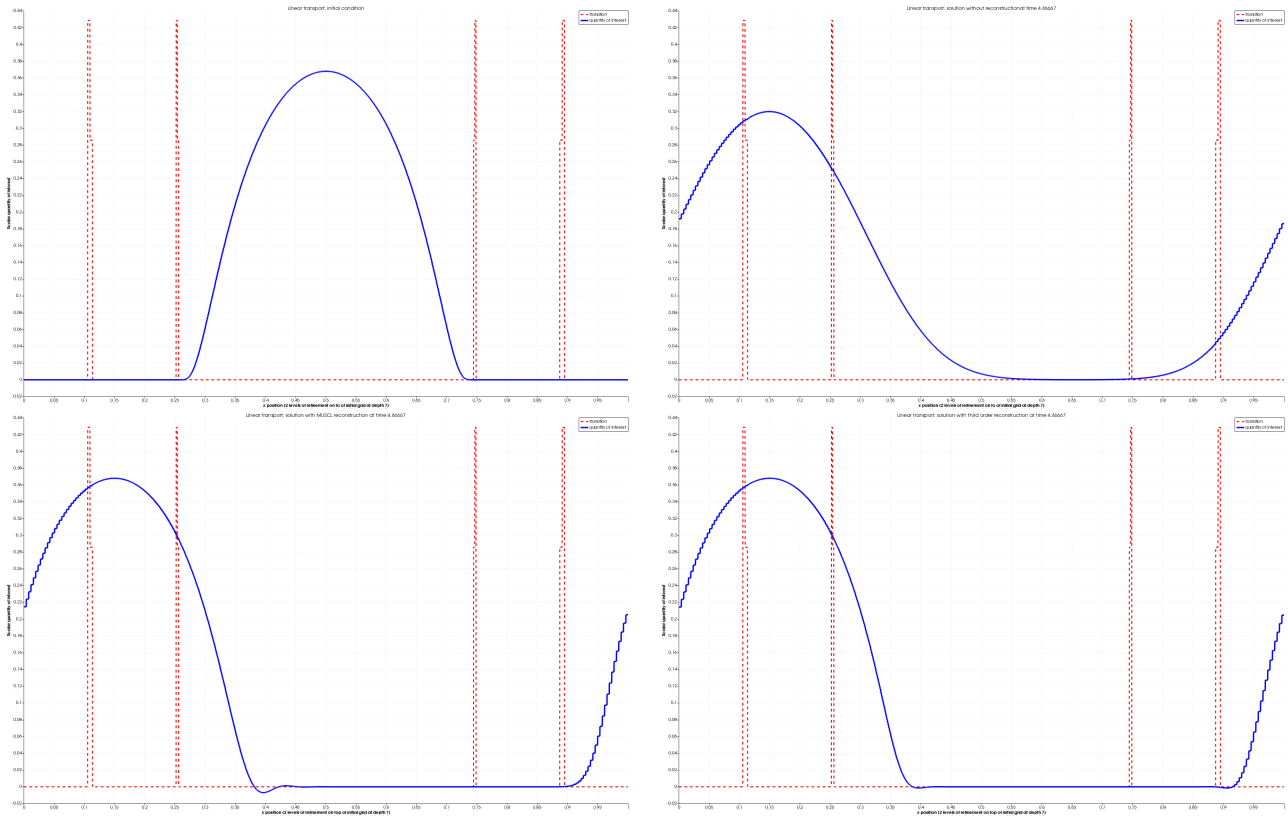


FIGURE 11. Scalar linear transport in 1D. From left to right, top to bottom: initial condition, solution without reconstruction, solution with MUSCL reconstruction, solution with third order reconstruction. The dashed pikes in red refer to the transitions between consecutive levels of refinement.

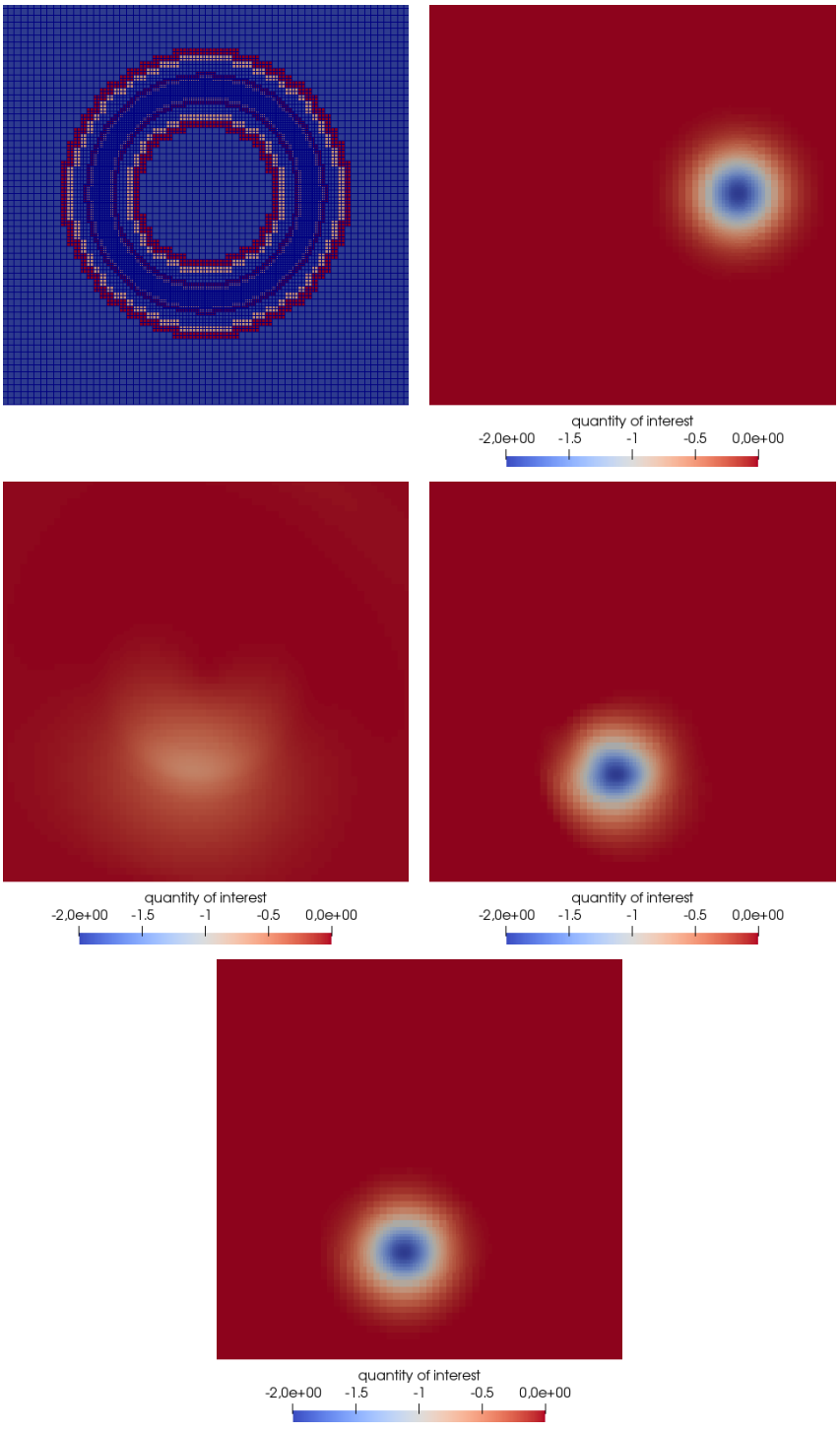


FIGURE 12. Scalar linear transport in 2D. From left to right, top to bottom: refined mesh transitions, initial condition, solution without reconstruction, solution with MUSCL reconstruction, solution with third order reconstruction.

Figure 11 helps us to describe the qualitative behavior of these schemes. When no reconstruction is performed, the numerical solution exhibits a strongly diffusive behavior with a loss of approximately 13.5% in amplitude at the end of the simulation. This is not the case anymore when reconstructing the solution near cell interfaces. However the MUSCL and third order reconstructions tend to favor spurious oscillations, which is especially noticeable for the MUSCL reconstruction. To address this issue, a fix would consist to use slope limiters ensuring the Total Variation Diminishing property (TVD).

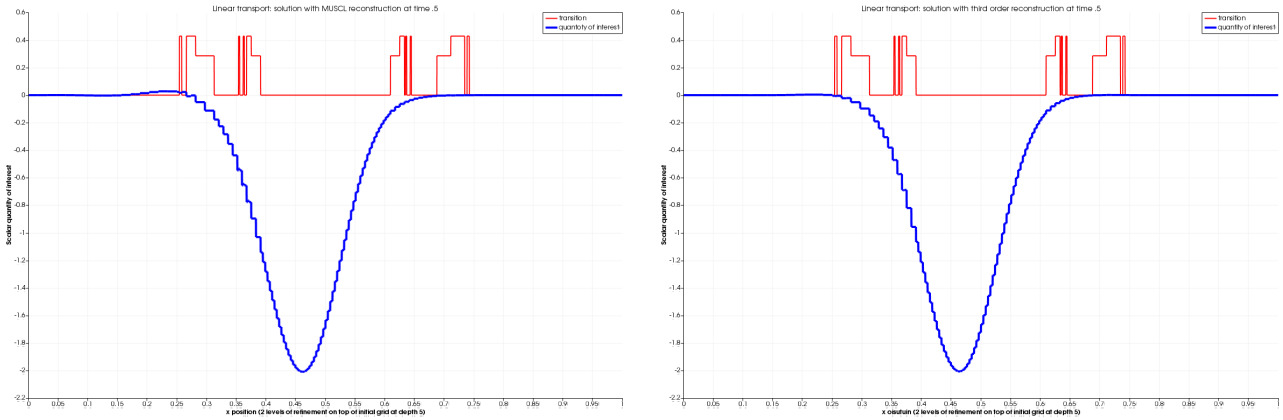


FIGURE 13. Plot over the line $y = 1/4$ at final time. Left: MUSCL reconstruction, right: third order reconstruction.

We then test our program on a 2D transport problem with non-constant velocity across the domain:

$$\partial_t u + \partial_x[v(x,y)u] + \partial_y[w(x,y)u] = 0$$

where $v(x,y) = (y - 1/2)/T$, $w(x,y) = -(x - 1/2)/T$, and $T = 1/2$ the final time. Again, the mesh is refined at the beginning and remains as such: there is no adaptivity involved here. More precisely, the mesh is made of two levels of refinement on top of the coarse grid initialized at depth 5. The initial condition is taken equal to a truncated Gaussian and is only C^0. Results can be seen in figure 12. The overall shape of the Gaussian is better preserved by the third order reconstruction, closely followed by the MUSCL reconstruction. As for the 1D test case, the latter reconstruction adds more downwind oscillations. This can be seen in figure 13 which consists in a plot of the solution at final time over the line $y = 1/4$.

5. CONCLUSION

This paper was devoted to the combination of high order finite volumes and Adaptive Mesh Refinement. These topics are of great interest in computational physics, for they are an efficient and accurate way to deal with interacting scales, while conserving the total mass of quantities of interest. After quickly recalling the state of the art concerning high order finite volumes, we have concentrated our attention towards AMR. Algorithms for third order multidimensional volume interpolation and communication between levels of refinement have been proposed. It turns out that reconstruction methods work well through transitions between levels of refinement, and do not require any specific treatment during the communication process. Of course, one has to make sure the mass conservation holds during this process to make the use of finite volumes relevant. In the end, numerical simulations allow to validate our approach, although truly adaptive meshes remains to be tested.

References

[DSD17] D Del Sarto and Erwan Deriaz. A multigrid amr algorithm for the study of magnetic reconnection. *Journal of Computational Physics*, 351:511–533, 2017.

[GMG98] R. Grauer, C. Marliani, and K. Germaschewski. Adaptive Mesh Refinement for Singular Solutions of the Incompressible Euler Equations. *Phys. Rev. Lett.*, 80:4177, 1998.

[KWA09] J.-H. Kim, J. H. Wise, and T. Abel. Galaxy mergers with adaptive mesh refinement: star formation and hot gas outflow. *ApJ*, 694, 2009.

[VL79] Bram Van Leer. Towards the Ultimate Conservative Difference Scheme. V. A Second-Order Sequel to Godunov's Method. *Journal of Computational Physics*, 32, 1979.

[vT09] Miroslav Čada and Manuel Torrilhon. Compact third-order limiter functions for finite volume methods. *Journal of Computational Physics*, 228:4118–4145, 2009.