Technische Universität München
Fakultät für Mathematik

Galilean-invariant simulations for conservation laws with a speed-density component on a moving mesh

Masterarbeit von Ezra Rozier

Themensteller : Herr Prof Dr. Oliver Junge

Betreuer : Herr Prof Dr. Oliver Junge

Abgabedatum : 15.10.2020

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of reference.

Ezra Rozier	08.10.2020 Date

Abstract

This Master's thesis is concerned with a parallel implementation in Julia of a finite volume scheme on a moving mesh to solve hyperbolic systems of conservation laws. The main focus of this thesis is to clarify the work of V.Springel in [1] as well as giving a short implementation in Julia of the AREPO code.

Table des matières

A Introduction

To compute the dynamic of structure formation in astrophysics, it has become central to use numerical methods. Thanks to the enhanced precision in these computation, and comparing it with what actually happens, we are able to predict precise characteristics of unknown physical components. For instance, we are able to be more precise in evaluating the characteristics of dark matter halos.

As we can see, even if we are interested in the dark matter halos that can be represented as collisionless particles, the only effects we are yet able to detect lie in the speed and position of classical particles, thus we have to compute hydrodynamic cosmological simulations. In this field there are several possible approaches, the two main kinds are SPH (smoothed particles hydrodynamics) which is a Lagrangian method, and mesh-based hydrodynamics which is an Eulerian method, to which you can add AMR (adaptive mesh refinement). Both these methods (moving mesh and mesh refinement by refining the number of cells) are called refinement strategies, for the moving mesh method it is called r-refinement and for the AMR h-refinement. In the following I will call r-refinement moving mesh and h-refinement refinement (or derefinement).

Both of these methods have their advantages and drawbacks. In particular we can mention the following : SPH, as a Lagrangian approach is Galilean-invariant but it has trouble computing shock solutions. On the other hand, Eulerian approach has a high order spatial accuracy but is not Galilean invariant. To cope with this, and try to combine the advantages of both methods I will, in this work, present the work of V.Springel, 2010 [1]. He mixed both Eulerian and Lagrangian approach on an ALE (arbitrary Lagrangian-Eulerian) code based on the SPH approach combined with AMR.

As I wrote before, such a method must be applicable to both classical and dark matter. Since the gravitational part of the simulation will be accounted for by a N-body simulation, I will only focus on the classical particles dynamics and not to take care of gravitationnal issues. In section B I will present the mathematical aspects of such a work and what are the areas of interest. In section C I will present my implementation of such a parralel code and how it adapts to Julia. In section D I will present my results for one specific test problem.

B Mathematical aspects

The method from Springer [1] consists in a finite volume method applied to a system of conservation laws and can be generalized to any hyperbolic conservation laws system for particles movement (as long as one has a speed and density component), but I will focus on the Euler equation for the following. To present this I will first discuss about the form of the Euler equation and its finite volume resolution and then get into the method and present all its computational aspects.
This finite volume scheme will be applied on a Voronoi moving mesh, thus I will also discuss the choice of the speed of each mesh-generating point. We will see later that the mesh movement will not be completely Lagrangian.

B - 1 The Euler equation

The Euler equation consists in a mass-momentum-energy conservation system and can be represented as follows :

$$\frac{\partial U}{\partial t} + \nabla \cdot F(U) = 0 \tag{1}$$

With $U = U(\boldsymbol{x}, t)$. As for the conservation laws we are interested in we have :

$$U = \begin{pmatrix} \rho \\ \rho \boldsymbol{v} \\ \rho e \end{pmatrix} = \begin{pmatrix} \rho \\ \rho \boldsymbol{v} \\ \rho u + \frac{1}{2}\rho \boldsymbol{v}^2 \end{pmatrix} \tag{2}$$

$$F(U) = \begin{pmatrix} \rho \boldsymbol{v} \\ \rho \boldsymbol{v}\boldsymbol{v}^T + P \\ (\rho e + P)\boldsymbol{v} \end{pmatrix} \tag{3}$$

With $P = (\gamma - 1)\rho u$.
We are going to employ a finite volume strategy integrating the fluid's state on a volume V_i of mass m_i, momentum $\boldsymbol{p}_i$ and energy E_i :

$$Q_i = \int_{V_i} U dV = \begin{pmatrix} m_i \\ \boldsymbol{p}_i \\ E_i \end{pmatrix} \tag{4}$$

Hence

$$\frac{dQ_i}{dt} = -\int_{\partial V_i} [\underbrace{F(U)}_{\text{(1) \& Green's law}} - \underbrace{U\boldsymbol{w}^T}_{\text{variation of the volume}}] d\boldsymbol{n} \tag{5}$$

with $\boldsymbol{n}$ being the normal vector of the cell surface, and $\boldsymbol{w}$ is the velocity of each point of the boundary.
Knowing that I will work on a Voronoi mesh thus in 2D (resp. 3D) ∂V_i is a reunion of segments (resp. convex polygones) $\boldsymbol{A}_{ij}$. By noting :

$$F_{ij} = -\frac{1}{A_{ij}} \int_{Aij} [F(U) - U\boldsymbol{w}^T] d\boldsymbol{A}_{ij} \tag{6}$$

$$\frac{dQ_i}{dt} = -\sum_{i \neq j} A_{ij} F_{ij} \tag{7}$$

And thus the scheme is going to be the following :

$$Q_i^{(n+1)} = Q_i^n - \Delta t \sum_j A_{ij} \hat{F}_{ij}^{(n+\frac{1}{2})} \tag{8}$$

$\hat{F}_{ij}^{(n+\frac{1}{2})}$ is here the average flux across the boundary ij predicted half a timestep further. In the following, the goal will be to put up a strategy to compute $\hat{F}_{ij}^{(n+\frac{1}{2})}$. To achieve this goal I will try to find a good time-average approximation.

B - 2 The Method

Before starting to be more precise about the mathematical or computational aspects I will briefly introduce the different steps of the method to give a frame

to what follows. This is the core idea of the method, we will see later that one can bring small remoted variations to this method :
Suppose that from the previous timestep you get the following informations :

- $\boldsymbol{r}_i$ the position of each mesh generating point
- $Q_i = (m_i, \boldsymbol{p}_i, E_i)^T$ the conservative variables of the cell i.

The algorithm
(i) Create the Voronoi mesh thanks to the r_i, this gives :
- $\boldsymbol{s}_i$ centre of mass of the cell i
- V_i the volume of the cell i
- A_{ij} and $\boldsymbol{f}_{ij}$ respectively the area and centre of the interfaces between cell i and j.

(ii) Compute $\boldsymbol{W}_i = (\rho_i, \boldsymbol{v}_i, P_i)^T$ thanks to the Q_i.
(iii) Evaluate the gradient of the density, of the speed components and of the pressure in each cell and apply a slope-limiter to avoid new extrema and overshoots.
(iv) Compute $\boldsymbol{w}_i$ the speed of the mesh-generating point for each $\boldsymbol{r}_i$.
(v) Use the Courant criterion to determine a timestep Δt_i.
(vi) For each face of the mesh, by linear extrapolation in space and prediction in time determine the state of the fluid on both sides of each face (the state of the face will be approximated by the state on the center of the face) and then compute $\hat{F}_{ij}^{(n+\frac{1}{2})}$ by solving a 1D Riemann problem.
(vii) Use (8) to compute Q_i^{n+1}.
(viii) Move the $\boldsymbol{r}_i$ with the $\boldsymbol{w}_i$

In this section I will follow the frame of the method except for the geometrical aspects of the Voronoi construction that I prefer to treat in the discussion about the implementation.

B - 3 Evaluation of the gradients

(iii) Evaluate the gradient of the density, of the speed components and of the pressure in each cell and apply a slope-limiter to avoid new extrema and overshoots.

We will evaluate the gradient from :

$$\{\int_{\partial V} \phi d\boldsymbol{n} = \int_V \nabla\phi dV\} \Rightarrow \{\langle\nabla\phi\rangle_i \simeq -\frac{1}{V_i}\sum_j \phi(f_{ij})\boldsymbol{A}_{ij}\} \tag{9}$$

⚠Here $\boldsymbol{A}_{ij}$ is the normal vector to the interface between cell i and cell j, of norm equal to the area and pointing from j to i.
N.B. : one could easily compute $\phi(f_{ij}) = \frac{1}{2}(\phi_i + \phi_j)$ but thanks to the geometry of the Voronoi tessellation it is not too expensive to be more accurate.
Suppose that in the neighbourhood of $\boldsymbol{r}_i$ one has a good linear approximation at a point $\boldsymbol{r}$ of $\phi : \phi(\boldsymbol{r}) = \phi_i + \boldsymbol{b} \cdot (\boldsymbol{r} - \boldsymbol{r}_i)$, $\boldsymbol{b}$ is the local gradient.
Then :

$$V_i \langle \nabla \phi \rangle = \int_{\partial V_i} \phi d\boldsymbol{n} = \sum_{i \neq j} \int_{A_{ij}} [\phi_i + \boldsymbol{b} \cdot (\boldsymbol{r} - \boldsymbol{r}_i)] \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{r_{ij}} dA \tag{10}$$

⚠$\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$
Let

$$\boldsymbol{c}_{ij} = \frac{1}{A_{ij}} \int_{A_{ij}} \boldsymbol{r} dA - \frac{\boldsymbol{r}_i + \boldsymbol{r}_j}{2} \tag{11}$$

be the vector going from the middle of the segment $[\boldsymbol{r}_i; \boldsymbol{r}_j]$ to the centre of the face.
Knowing that $\phi_j = \phi_i + \boldsymbol{b} \cdot (\boldsymbol{r}_j - \boldsymbol{r}_i)$:

$$V_i \langle \nabla \phi \rangle_i = -\sum_{i \neq j} [\frac{\phi_i + \phi_j}{2} + \boldsymbol{b} \cdot \boldsymbol{c}_{ij}] \frac{\boldsymbol{r}_{ij}}{r_{ij}} A_{ij} \tag{12}$$

With the following relation : $(\boldsymbol{b} \cdot \boldsymbol{c}_{ij}) \boldsymbol{r}_{ij} = \underbrace{(\boldsymbol{b} \cdot \boldsymbol{r}_{ij})}_{=\phi_i - \phi_j} \boldsymbol{c}_{ij} + \boldsymbol{b} \times (\boldsymbol{r}_{ij} \times \boldsymbol{c}_{ij})$.

And

$$\boldsymbol{b} \times \sum_{i \neq j} \frac{\boldsymbol{r}_{ij} \times \boldsymbol{c}_{ij}}{r_{ij}} A_{ij} = \boldsymbol{b} \times \sum_{i \neq j} \frac{\boldsymbol{r}_{ij}}{r_{ij}} \times \int_{A_{ij}} [\boldsymbol{r} - \frac{\boldsymbol{r}_i + \boldsymbol{r}_j}{2}] dA \tag{13}$$

It appears that $\frac{\boldsymbol{r}_{ij}}{r_{ij}} = \boldsymbol{n}$, $-\int_{\partial V} \boldsymbol{r} \times d\boldsymbol{n} = \int_V \nabla \times \boldsymbol{r} dV$ thus $\int_{\partial V} \boldsymbol{r}_i \times d\boldsymbol{n} = \overrightarrow{0}$ because $\boldsymbol{r}_i$ is constant here and V_i is a closed volume, also $\boldsymbol{r}_{ij} \times \frac{\boldsymbol{r}_i + \boldsymbol{r}_j}{2} = \boldsymbol{r}_{ij} \times$

$\boldsymbol{r}_i = \boldsymbol{r}_i \times \boldsymbol{r}_j$. Moreover the term $\sum_{i \neq j} \int_{A_{ij}} d\boldsymbol{n} \times \boldsymbol{r}$ vanishes because ∂V is a closed surface. Finally we have the equality (13) that is equal to $\overrightarrow{0}$.
And then :

$$\langle \nabla \phi \rangle_i = \frac{1}{V_i} \sum_{i \neq j} Aij[(\phi_j - \phi_i) \frac{\boldsymbol{c}_{ij}}{r_{ij}} - \frac{\phi_i + \phi_j}{2} \cdot \frac{\boldsymbol{r}_{ij}}{r_{ij}}] \tag{14}$$

Now for all $\boldsymbol{r} \in V_i$ we set $\rho(\boldsymbol{r}) = \rho_i + \langle \nabla \rho \rangle_i \cdot (\boldsymbol{r} - \boldsymbol{s}_i)$, the same goes for the speed and the pressure.
N.B. : I could have used $\boldsymbol{r}_i$ and not $\boldsymbol{s}_i$, I chose $\boldsymbol{s}_i$ because we are more interested into the mass movement than the mesh movement. This shows the importance of keeping $\boldsymbol{r}_i \simeq \boldsymbol{s}_i$.

B - 4 Slope-limiting the gradient

> **(iii)** Evaluate the gradient of the density, of the speed components and of the pressure in each cell and **apply a slope-limiter to avoid new extrema and overshoots.**

To avoid apparition of new extrema I set a slope-limiter that keeps the values of the intensive variables on the interface $\boldsymbol{f}_{ij}$ between those of the centres of mass $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$.
I will replace $\langle \nabla \phi \rangle_i$ by $\langle \nabla \phi \rangle'_i = \alpha_i \langle \nabla \phi \rangle_i$. With

$$\alpha_i = \min_{j \text{ neighbour of } i} (1, \psi_{ij})$$

$$\psi_{ij} = \begin{cases} \frac{\phi_i^{\max} - \phi_i}{\Delta \phi_{ij}} & \text{if } \Delta \phi_{ij} > 0 \\ \frac{\phi_i - \phi_i^{\min}}{\Delta \phi_{ij}} & \text{if } \Delta \phi_{ij} < 0 \\ 1 & \text{else} \end{cases}$$

with $\Delta \phi_{ij} = \langle \nabla \phi \rangle_i \cdot (\boldsymbol{f}_{ij} - \boldsymbol{s}_i)$ estimation of the variation between the centre of the face and the centre of mass of the cell.
we also have $\phi_i^{\max} = \max_{j \text{ neighbour of } i \text{ including } i} (\phi_j)$ and $\phi_i^{\min} = \min_{j \text{ neighbour of } i \text{ including } i} (\phi_j)$.
Oscillations are not technically avoided because it is not TVD.

B - 5 Set the $\boldsymbol{w}_i$

> **(iv)** Compute $\boldsymbol{w}_i$ the speed of the mesh-generating point for each $\boldsymbol{r}_i$.

For a close to purely Lagrangian scheme we set $\boldsymbol{w}_i = \boldsymbol{v}_i$ (this is going to be the mainly used) which is the most convenient for an advection equation but I will also explore two variations in the value of $\boldsymbol{w}_i$ to keep mesh regularity.

B - 5.1 Keep cells round

To keep $\boldsymbol{r}_i \simeq \boldsymbol{s}_i$ one possibility is to make a round cell (a cell is round iif $2R_i = \text{diam}(V_i)$ with $R_i = \sqrt{\frac{V_i}{\pi}}$ in 2D and $R_i = \sqrt[3]{\frac{V_i}{4\pi}}$ in 3D).
If the value $d_i = |\boldsymbol{r}_i - \boldsymbol{s}_i| > \eta R_i$ we will add a component proportional to the local soundspeed ($c_i = \sqrt{\frac{\gamma P_i}{\rho_i}}$). To smoothen this factor there will be a transition :

$$\boldsymbol{w}_i' = \boldsymbol{w}_i + \chi \begin{cases} 0 & \text{for } d_i < 0.9\eta R_i \\ c_i \frac{\boldsymbol{s}_i - \boldsymbol{r}_i}{d_i} \cdot \frac{d_i - 0.9\eta R_i}{0.2\eta R_i} & \text{for } 0.9\eta R_i \leq d_i < 1.1\eta R_i \\ c_i \frac{\boldsymbol{s}_i - \boldsymbol{r}_i}{d_i} & \text{for } d_i \geq 1.1\eta R_i \end{cases}$$

To do so I will add two new factors : η and χ. Generally I will use $\eta = 0.25$ and $\chi = 1$.

B - 5.2 Keep mass-volume constant

For computational reason, it can be interesting to keep a factor depending on the mass and volume of each cell constant. Indeed, generally mass is a good indicator of which zone is the most active, thus decreasing the volume when the mass increases can be a low-cost, efficient way to have mesh moving adaptation. In the following I will present a way to make the value $K_i := \frac{m_i}{\tilde{m}} + \frac{V_i}{\tilde{V}}$ constant in space. With $\tilde{m}$ and $\tilde{V}$ constants defined apriori to be maximal possible values of m_i and V_i.
Let $n(\boldsymbol{x})$ be the density of mesh generating point in $\boldsymbol{x}$ and $n_0(\boldsymbol{x})$ the ideal density (in the sense that : $\forall i \; K_i = \tilde{K}$). To n we associate positions $\boldsymbol{x}_i$ and to n_0 positions $\boldsymbol{q}_i$ and we note : $\boldsymbol{x}_i = \boldsymbol{q}_i + \epsilon \boldsymbol{d}_i$ with $\epsilon = 1$ for this situation. We will suppose that $\boldsymbol{d}$ can be expressed as the gradient of a scalar field : $\boldsymbol{d} = -\nabla\Psi$.
We will now make ϵ vary with time and study the resulting motion of the points and density n.
With n_ϵ the density with generating points $\boldsymbol{x}_i^\epsilon = \boldsymbol{q}_i + \epsilon \boldsymbol{d}_i$ there is in fist order :

$$n_\epsilon(\boldsymbol{x} + \epsilon \boldsymbol{d}) = \epsilon n(\boldsymbol{x} + \epsilon \boldsymbol{d}) + (1 - \epsilon) n_0(\boldsymbol{x})$$

We set the velocity of a point : $\boldsymbol{v} = \frac{d\boldsymbol{x}^\epsilon}{d\epsilon} = \boldsymbol{d}$.
Then the Lagrange continuity equation gives : $\frac{dn}{d\epsilon} + n\nabla \cdot \boldsymbol{v} = 0$. Which gives for $\epsilon = 0$ the following :

$$\nabla^2\Psi = \frac{n(\boldsymbol{x}+\boldsymbol{d})}{n_0(\boldsymbol{x})} - 1 \simeq \frac{n(\boldsymbol{x})}{n_0(\boldsymbol{x})} - 1.$$

Now let's compute n_0, for the ideal mesh we have $m_i = \frac{\rho(\boldsymbol{q}_i)}{n_0(\boldsymbol{q}_i)}$ and $V_i = \frac{1}{n_0(\boldsymbol{q}_i)}$, then :

$$n_0(\boldsymbol{x}) = \frac{1}{\tilde{K}}\left(\frac{\rho(\boldsymbol{x})}{m_i} + \frac{1}{V_i}\right)$$

So in the end, we find the displacement $\boldsymbol{d} = -\nabla\Psi$ with $\nabla^2\Psi = \frac{\tilde{K}n(\boldsymbol{x})}{\frac{\rho(\boldsymbol{x})}{\tilde{m}} + \frac{1}{\tilde{V}}} - 1$.
For periodic boundary conditions, the existence of a solution to the Poisson equation on an infinite periodic space enforces : $\tilde{K} = \frac{V_{\text{tot}}}{\sum_i (\frac{\rho_i}{\tilde{m}} + \frac{1}{\tilde{V}})^{-1}}$.
Indeed to have the existence of a solution on an infinite periodic space, then it enforces $\int_{V_{\text{tot}}} \nabla^2\Psi = 0$ thus :

$$\int_{V_{\text{tot}}} \frac{\tilde{K}n(\boldsymbol{x})}{\frac{\rho(\boldsymbol{x})}{\tilde{m}} + \frac{1}{\tilde{V}}} = V_{\text{tot}}$$

Then :

$$\tilde{K}\sum_i \left(\frac{\rho_i}{\tilde{m}} + \frac{1}{\tilde{V}}\right)^{-1} \underbrace{\int_{V_i} n(\boldsymbol{x})}_{=1} = V_{\text{tot}}$$

with V_i the cell determined by $\boldsymbol{x}_i$.
Finally :

$$\tilde{K} = \frac{V_{\text{tot}}}{\sum_i (\frac{\rho_i}{\tilde{m}} + \frac{1}{\tilde{V}})^{-1}}$$

And we correct the mesh generating points' with the calculated displacements $\boldsymbol{w}_i' = \boldsymbol{w}_i + \kappa\frac{\boldsymbol{d}_i}{\Delta t_i}$ generally with $\kappa = 0.5$.

B - 6 Compute the timestep

(v) Use the Courant criterion to determine a timestep Δt_i.

We can use either a global timestep (the same timestep for every cell) or an individual timestep (a tailored timestep for each cell).

B - 6.1 Global timestep

We will there use a global Courant criterion modified because of the motion of the cell. In that case the global timestep is :

$$\Delta t_i = C_{\mathrm{CFL}} \frac{R_i}{c_i + |\boldsymbol{v}_i \text{-} \boldsymbol{w}_i|} \tag{15}$$

$$\delta t = \min(\Delta t_i) \tag{16}$$

B - 6.2 Individual timestep

It can be more interesting to use individual timestepping : we can compute the state more often for more involved cells than for cells far from any movement. To do so we will build a hierarchy of timesteps : suppose that a timestep Δt_i is computed and that we have determined a given constant Δt which will be used as a basis (it is a fixed constant taken to bradly shape our problem), then we can associate $k_i = \lfloor \log_2(\frac{\Delta t}{\Delta t_i}) \rfloor + 1$ to each cell. This hierarchy will be done in decreasing k_i. The effective timestep for each cell will be $\frac{\Delta t}{2^{k_i}}$.

Evaluating Δt_i

The evaluation of Δt_i has to take into account two things : the Courant criterion for the cell i and the minimum time before a wave coming from another cell arrives in the cell i even if this wave is faster than the local soundspeed c_i.

For this second condition we will introduce a "signal" speed between two cells i and j :

$$v_{ij}^{\text{sig}} = c_i + c_j - \boldsymbol{v}_{ij} \cdot \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{17}$$

$\boldsymbol{v}_{ij} = \boldsymbol{v}_i - \boldsymbol{v}_j$ and $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$.

This speed is always bigger than the speed of a wave that would propagate from i to j (or j to i) and ensures that the cell i (or j) would be active again before the wave arrives on it.

Now to compute Δt_i we do :

$$\Delta t_i = C_{\text{CFL}} \min(\tau_i, \frac{R_i}{c_i + |\boldsymbol{w}_i - \boldsymbol{v}_i|}) \tag{18}$$

$$\tau_i = \min_{j \neq i}(\frac{r_{ij}}{v_{ij}^{\text{sig}}}) \tag{19}$$

I will discuss in the computational aspects a tree method to efficiently compute every timestep.

On the other hand, with individual timestepping, the algorithm from Springer[1] is going to be changed in its last part.

Changing the algorithm

Suppose that we computed all the k_i. Suppose also that we are in the state transition between t and $t + \frac{\Delta t}{2^{k_{\max}}}$ with :
$k_{\max} = \max(k_i),\ \exists! k \in \mathbb{N},\ t + \frac{\Delta t}{2^{k_{\max}}} = m\frac{\Delta t}{2^k},\ m \in \mathbb{N},\ 2 \wedge m = 1.$
N.B. : - the transition will always be done with the timestep $\frac{\Delta t}{2^{k_{\max}}}$
- t is always a multiple of $\frac{\Delta t}{2^k}$ for some k because all the transitions between states are done with timesteps of the form $\frac{\Delta t}{2^{k_{\max}}}$
- we will see later in **C - 4** why there always is $k \leq k_{\max}$.

We will call "active" every cell i such that $k_i \geq k$.
For each interface $i \leftrightarrow j$, we call this interface "active" if $\max(k_i, k_j) \geq k$.

Than the algorithm will be as follows (the changes will be written in red) :

(i) Create the Voronoi mesh and set a tree organisation of your data
(ii) Compute $\boldsymbol{W}_i$ **of each active cell**
(iii) Evaluate the gradient **of each active cell**
(iv) Compute $\boldsymbol{w}_i$
(v) Thanks to the tree organisation of your data, **compute every individual timestep for active cells, this will be achieved such as** $k_i^{\text{new}} \geq k$ **an than compute** $k_{\max}$ **to find the next timestep where cells are active, update** t **by** $t + \frac{\Delta t}{2^{k_{\max}}}$ **and compute** k **to determine all active cell**
(vi) Compute flux **across each active interface, for an interface** $i \leftrightarrow j$ **the flux will be computed for a time :** $\frac{\Delta t}{2^{\max(k_i, k_j)}}$
(vii) Compute $Q_i^{(n+1)}$ **for each cell including an active interface**
(viii) Move the $\boldsymbol{r}_i$ with the $\boldsymbol{w}_i$.

B - 7 Flux computation

(vi) For each face of the mesh, by linear extrapolation in space and prediction in time determine the state of the fluid on both sides of each face (the state of the face will be approximated by the state on the center of the face) and then compute $\hat{F}_{ij}^{(n+\frac{1}{2})}$ by solving a 1D Riemann problem.

B - 7.1 Speed of the interface

The interface between two cells with speed $\overrightarrow{w_i}$ and $\overrightarrow{w_j}$ of the generating points r_i and r_j has a speed that can be caracterised by the speed of the centroïd : let $\overrightarrow{OM}(t) = \frac{\overrightarrow{Or_i}+\overrightarrow{Or_j}}{2}$, $\overrightarrow{x} = \frac{\overrightarrow{Mr_i}}{|Mr_i|}(t=0)$ and $\overrightarrow{y} = \frac{\overrightarrow{Mf_{ij}}}{|Mf_{ij}|}(t=0)$ and let's work on the orthonormal $(M, \overrightarrow{x}, \overrightarrow{y})$ (if it is not in direct orientation you do $i \leftrightarrow j$). Let's suppose that the point M does not move (to achieve that, one just has to shift all speeds by $-\frac{\overrightarrow{w}_i+\overrightarrow{w}_j}{2}$), hence $\overrightarrow{w}_i = -\overrightarrow{w}_j = v\overrightarrow{x} + \omega|Mr_i|\overrightarrow{y}$.

In time, the angle between the mediator and $\overrightarrow{x}$ will be equal to : $\alpha(\delta t) = \omega\delta t$ and thus the displacement of f_{ij} will be : $\overrightarrow{d} = \alpha(\delta t)|Mf_{ij}|(t=0)\overrightarrow{y}$ (if we do not take into account the $\overrightarrow{x}$ component of $\overrightarrow{w}_i$, we make this approximation because the divergence between r_i and r_j changes the shape of the face, but this change depends on the other neighbours, thus at first order, in the sense that we only take into account the two interfacing cells we can neglect v).
And thus the speed is

$$\overrightarrow{w} = \frac{\overrightarrow{w_i}+\overrightarrow{w_j}}{2} + (\overrightarrow{w_i}-\overrightarrow{w_j})\cdot(\overrightarrow{Of_{ij}} - \frac{\overrightarrow{Or_i}+\overrightarrow{Or_j}}{2})\cdot\frac{\overrightarrow{r_jr_i}}{|\overrightarrow{r_jr_i}|^2} \tag{20}$$

B - 7.2 Flux computation

Now we will be able to compute the flux. To do so one has to compute the state $(\rho, \boldsymbol{v}, P)^T$ on the centre of the interface as the solution to a 1D Riemann problem. Thus first, I shift all the speeds by $\boldsymbol{w}$, then I predict the state further in space and further by $\frac{\delta t}{2}$ in time and rotate the referential so $(r_i; f_{ij})$ colinear with the x-axis.

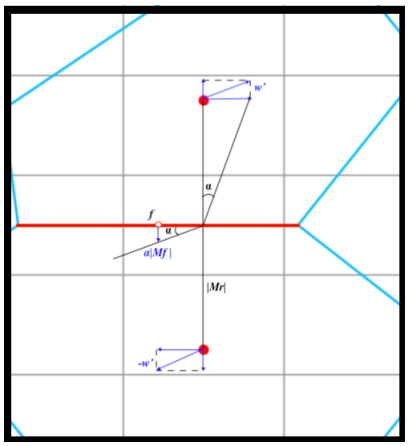


FIGURE 1 – Geometry of a face

Let $\boldsymbol{W} = (\rho, \boldsymbol{v}, P)^T = \boldsymbol{W}_i$ or $\boldsymbol{W}_j$.

$$\boldsymbol{W}' = \boldsymbol{W} + \begin{pmatrix} 0 \\ \boldsymbol{w} \\ 0 \end{pmatrix} \tag{21}$$

$$\boldsymbol{W}'' = \boldsymbol{W}' + \frac{\partial \boldsymbol{W}'}{\partial \boldsymbol{r}} \cdot (\boldsymbol{f} - \boldsymbol{s}) + \frac{\partial \boldsymbol{W}'}{\partial t} \cdot \frac{\delta t}{2} \tag{22}$$

To evaluate $\frac{\partial \boldsymbol{W}'}{\partial t}$ use $\frac{\partial \boldsymbol{W}'}{\partial t} + A(\boldsymbol{W}')\frac{\partial \boldsymbol{W}'}{\partial \boldsymbol{r}} = 0$ with $A(\boldsymbol{W}')$ being the Jacobian matrix in $\boldsymbol{W}'$:

$$A(\boldsymbol{W}) = \begin{pmatrix} \boldsymbol{v} & \rho & 0 \\ 0 & \boldsymbol{v} & \frac{1}{\rho} \\ 0 & \gamma P & \boldsymbol{v} \end{pmatrix} \tag{23}$$

Finally there remains only the rotation of the speed so $(r_i; f_{ij})$ coincides with the x-axis :

$$\boldsymbol{W}''' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \Lambda & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \boldsymbol{W}'' \tag{24}$$

And then use a Riemann solver R :

$$\boldsymbol{W}_F = R(\boldsymbol{W}_i''', \boldsymbol{W}_j''') \tag{25}$$

Now to compute the flux, come back in the lab frame and take into account the movement of the face :

$$\boldsymbol{W}_{\text{lab}} = \begin{pmatrix} \rho \\ \boldsymbol{v}_{\text{lab}} \\ P \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \Lambda^{-1} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \boldsymbol{W}_F + \begin{pmatrix} 0 \\ \boldsymbol{w} \\ 0 \end{pmatrix} \tag{26}$$

$$\hat{F} = F(U) - U\boldsymbol{w}^T = \begin{pmatrix} \rho(\boldsymbol{v}_{\text{lab}} - \boldsymbol{w}) \\ \rho \boldsymbol{v}_{\text{lab}}(\boldsymbol{v}_{\text{lab}} - \boldsymbol{w})^T + P_{\text{lab}} \\ \rho \boldsymbol{e}_{\text{lab}}(\boldsymbol{v}_{\text{lab}} - \boldsymbol{w}) + P\boldsymbol{v}_{\text{lab}} \end{pmatrix} \tag{27}$$

With $\boldsymbol{e}_{\text{lab}} = \frac{\boldsymbol{v}_{\text{lab}}^2}{2} + \frac{P}{(\gamma-1)\rho}$.

C Implementation aspects

In this section I will get interested into all the implementation aspects that I took into account. I will thus present in a first part the implementation of a quad-/octtree data structure for a Voronoi mesh construction. Then in a second part I will highlight the main elements of the parallelization of my algorithm and emphasize on the use of Julia for this work. Finally I will come back on several secondary aspects of the computation.

C - 1 The Voronoi construction

In this part we will work in the space $\Omega \subset \mathbb{R}^d$ bounded, convex with $d \in \{2, 3\}$. A Voronoi mesh is defined as follows : for a collection $(r_i)_{i\in I},\ r_i \in \Omega\ ,\ I \subset \mathbb{N}\ ,\ I$ finite, of mesh generating points, the cells will be as follows,
$\Omega_i = \{r \in \Omega \mid i \in \arg\max_{j\in I} |r - r_j|\}$.

N.B. : We have :
For all i, Ω_i is closed and convex with $\forall r \in \overset{\circ}{\Omega}_i\ |\arg\max_{j\in I} |r - r_j|| = 1$ and $\forall r \in \partial\Omega_i\ |\arg\max_{j\in I} |r - r_j|| \geq 2$.

We can show easily that for $d = 2$:

$\forall i, j \in I\ \Omega_i \cap \Omega_j$ is empty, a point or a segment

$\forall J \subset I,\ |J| \geq 3,\ \bigcap_{j\in J} \Omega_j$ is empty or a point

and for $d = 3$:

$\forall i, j \in I\ \Omega_i \cap \Omega_j$ is empty, a point, a segment or a bounded, convex plan

$\forall J \subset I,\ |J| = 3,\ \bigcap_{j\in J} \Omega_j$ is empty, a point or a segment

$\forall J \subset I,\ |J| \geq 4,\ \bigcap_{j\in J} \Omega_j$ is empty or a point

Therefore, $\{\overset{\circ}{\Omega}_i\}_{i\in I}$ is a triangulation of Ω.
We will call interface $i \leftrightarrow j$ the $\Omega_i \cap \Omega_j$ when it is a segment for $d = 2$ and a plan for $d = 3$ (in fact $\Omega_i \cap \Omega_j$ is in the mediator between r_i and r_j.
To compute a Voronoi tessellation, we have to know first that it is the dual of the Delaunay tessellation. A Delaunay tessellation for a collection $(r_i)_{i\in I}$ of generating points is as follows : it is a tessellation of triangles where the vertices are the generating points and a triangle $(r_i r_j r_k)$ is in the tessellation if there is no other generating point in its circumcircle.
Then the vertices of the Voronoi tessellation are all the circumcentres and the edges join every circumcentre with its three closest other vertices for $d = 2$ and

with its four closest vertices for $d = 3$.
The precise description of the algorithm to build a Voronoi tessellation based on the properties of the Delaunay tessellation is given in [1], I will not discuss it there.
To build it computationally I followed the construction in [1] in a convex closed square of size 0.95×0.95. I chose this size because the library GeometricalPredicates [10] in Julia is defined only on a 1×1 square and I needed a margin to create ghost cells representing the periodic/symmetric boundary conditions.
In the end I stored all these generating points in an oct-/quadtree [13] which gives the possibility to find a cell in $\mathcal{O} \log(n)$ which improves spectacularly the complexity of the algorithm. In these cells, I stored all the informations I will need in the course of a loop.
I built two similar trees in the middle of my code for the computation of the speed $\boldsymbol{w}_i$ (TreePM code) and of the timestep.

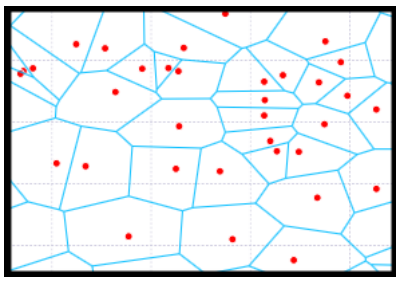


FIGURE 2 – Voronoi mesh

For the Voronoi construction I started with [12] and adapted this computation to fit with my data structure (restrict the construction to a square, adapt it for plotting with Gadfly library...).

C - 2 Parallelization

I will present the process of parallelization in three parts : first I will explain how I decomposed my domain to parallelize the computation and how it changes the algorithm, then I will introduce the classical standard for parallelization, MPI, and I will finally explain why I prefered to use the built-in parallel computation method of Julia.

C - 2.1 Domain Decomposition

As a first test I only brutally decomposed my domain into four (or nine) equal surfaces.
The question now is : which algorithm did I use to find the ghost cells ? My algorithm is based on the properties of the Delaunay tessellation. Indeed : for a given processor I want to find all the neighbours in the global Voronoi tessellation of the cells stored in this processor : these outside neighbours are called ghost cells.
The most important aspect in parallelization is having as few data communication as possible between the processors for it is costful. All outside cells considered in the computation of a given processor will be imported as ghost cells. Then at the end of each loop I export the data from each computation processor into the main processor.

C - 2.2 The MPI standard

MPI (for Message Passing Interface) is a standard designed for parallel computing. At first it was built to support parallel computing on multi-size entities (as an heterogenous cluster of computers) in C/C++ and Fortran codes, but it is also now implemented in other languages like Python or Julia. In Julia this is considered as the external standard for parallel computing for there exists an inside parallelization standard. Yet there exists a MPI library [11] which is for instance used as the standard for the initial AREPO implementation in [1].

C - 2.3 The Julia parallelization standard

Julia has its own implemented distributed computing standard. Indeed, in Julia you can parallelize the execution of channels : with this structure I was able to parallelize my code with an easy implementation. The data access was also quite convenient when I had to build processor communication.

C - 3 Other aspects

In this section I will come-back in several other aspects of the algorithm, in particular the mesh refinement strategy as well as the Riemann resolution and the ghost cells research.

C - 3.1 Adaptive mesh refinement

In the AREPO original code the mesh refinement strategy goes as follows : if if a cell's mass or volume exceeds a given upper threshold, the mesh will be refined by adding a generating point almost exactly at the same position as the previous one. If those properties go below a lower threshold it will be derefined by simply removing the generating point.
This strategy can be improved quite easily for instance by sorting the cells and refining (resp. derefining) the heavier (resp. the lighter) 5% or/and the largest (resp. the smallest) 5%, otherwise by changing the refinement criterion. One can use for instance a criterion of movement with the gradient of the values or a criterion that takes into account where the error is mostly created. For error approximation see the apriori error estimation for nonlinear conservation laws with finite volume [2] as well as a posteriori error estimation (which demands considerably more effort) [3].
Another way to improve the refinement condition is to use a shock detector ([4] Section 8) but it demands more apriori work. In this strategy, you look for the large variations of your data and refine in those regions.

C - 3.2 The Riemann solver

Another aspect of discussion is in the choice of the Riemann solver. Indeed, in this example for the Euler equation in 2D/3D, one has only to compute a 5×5 Jacobian matrix to have an exact Riemann solver. But when one has a larger system of hyperbolic conservation laws, it becomes crucial to use non-Jacobian Riemann solver. Indeed, it is possible to compute a good approximation of the solution to a Riemann problem, only with evaluations of the flux function.
As we work with hyperbolic conservation laws, all the non-exact solvers thereafter rely on an approximation of the solution for the flux. Having a non-exact, non-Jacobian solver such as the HLL (Harten-Lax-van Leer) flux helps to get speed without loosing too much precision (this is for instance what is currently implemented in the fast version of AREPO on Github [9] with HLLC for classical Euler equation and HLLD for MHD). However, those HLL solvers are always built for one specific problem (HLLC, HLLD...) because they precisely rely on computing the eigensystem. Improvement for large systems of conservation laws for HLL was explored in [6]. For a problem with more conservative variables such as in [8] the Krylov-Riemann solver [5] or the MUSTA solver [7] are interesting as they only rely on evaluation of the function and not on computing an eigen-

system (except for a global estimate of the largest propagation speed). Moreover they are both better solving discontinuities than the Lax-Wendorff flux [5].

C - 3.3 Ghost research

As for the ghost research, there are two algorithms that can be use the geometry of a Voronoi tessellation [1] :

First algorithm
Take a generating point r in a local processor and do :
- Let h be a length and s two times the size of the biggest neighbouring circumsphere.

If $h < s : h = \alpha h,\ (\alpha > 1)$ and start again.

Elseif there is no point outside the local processor and inside the sphere centered in r and $h > s$: the algorithm is done.

Else (if there is a point outside the local processor and inside the sphere centred in r and $h > s$) : add the point(s) found to the Voronoi tesselation, update s and restart the search.

Second algorithm
Suppose we begin with a Voronoi tessellation.

Take every Delaunay triangle that have at least one vertice inside the processor and find all the generating points inside the circumspheres of these Delaunay triangles.

Repeat this operation while there are still added points.

The first algorithm gives at least all the neighbours of the generating points stored inside a given processor, the second algorithm gives exactly the neighbours of the generating points inside a given processor. I chose to use the second algorithm.
In the end my algorithm is going to be as seen next page.

Initialisation : generating points, conservative and intensive variables

I/ Decompose the domain

II/ Enter the first parallel computation :

(i) Find ghost cells

(ii) Build the tree

(iii) Construct local Voronoi mesh

III/ Update all data depending on the geometry for ghost cells (volume, soundspeed, radius...)

IV/ Build the tree for timestep calculation (which concatenates the trees stored in the working processors), store it in the main processor and compute in parallel the timesteps.

V/ Find the first time δt at which there will be active cells

VI/ Enter the second parallel computation :

(i) Compute slope-limited gradients

(ii) Compute speed of the generating points

VII/ Enter the computation loop : **while** $t < t_{\mathbf{final}}$

1 - Build the tree for timestep calculation (which concatenates the trees stored in the working processors), store it in the main processor and compute in parallel the timesteps for active cells.

2 - Update the gradient and the speed of ghost cells

3 - Compute fluxes for active faces

4 - Compute the next time where particles are active and determine, $t + \frac{\Delta t}{2^k_{\max}}$, and find m s.t. $t + \frac{\Delta t}{2^{k_{\max}}} = m\frac{\Delta t}{2^k}$

5 - Shift all mesh generating points by $\delta t \boldsymbol{w}_i = \frac{\Delta t}{2^k_{\max}} \boldsymbol{w}_i$ and update conservative variables of all cells

6 - Decompose your domain

7 - Enter a parallel computation :

(i) Find ghost cells

(ii) Build the tree

(iii) Construct local Voronoi mesh

(iv) Refine and derefine the mesh

(v) Update intensive variables for active cells

8 - Update intensive variables of ghost cells as well as data depending on the geometry

9 - Enter a parallel computation :

(i) Compute slope-limited gradients

(ii) Compute speed of the generating points.

C - 4 Timestep computation

For this computation I used a tree algorithm with an oct-/quadtree : I built a tree with the same shape as the tree storing the data of my problem where on each node. If it is a leaf storing a cell, I store the soundspeed, the norm of the particle's speed and the data of the cell. If it is a leaf that stores the empty cell (the empty cell is a cell I defined) I store 0 for the soundspeed, 0 for the speed and the empty cell. As for an inside node I store the maximum value of the soundspeeds stored in its four (or eight) children, the maximum value of the particles' speed norm stored in its four (or eight) children and the empty cell. I call this tree T.

To compute the timestep of a specific cell I did as follows : I built a recursive algorithm that takes a node N, a cell C with generating point $\boldsymbol{r}_i$ and a timestep δt, I will call this algorithm $f(C, N, \delta t)$. A node in an oct-/quadtree is delimited by eight or four vertices, thanks to those I compute the distance d between the generating point of C and the node N.
If N is a cell written $C' \neq C$ with generating point $\boldsymbol{r}_j$, I compute $\boldsymbol{v}_{ij}^{\text{sig}}$, I return $\min(\frac{|\boldsymbol{r}_i - \boldsymbol{r}_j|}{\boldsymbol{v}_{ij}^{\text{sig}}}, \delta t)$.
If N is a cell storing the empty cell I return δt.
If N is the cell C I return δt.
Else, N is an inside node, I call c the soundspeed stored in the node and v the speed stored in the node
    If $\frac{d}{c+v} \geq \delta t$ I return δt
    Else, I write N_l, $l \in \{1, 2, 3, 4\}$ or $\{1, 2, 3, 4, 5, 6, 7, 8\}$, I return $\min_l(f(C, N_l, \delta t))$.

Then for each cell C I will define its timestep as $\Delta t_i = f(C, T, \frac{R_i}{c_i + |\boldsymbol{v}_i - \boldsymbol{w}_i|})$ and I will compute $k_i = \min(k, \log_2(\lfloor \frac{\Delta t}{\Delta t_i} \rfloor + 1))$ with k as in **B - 6.2** and then I finally set $\Delta t_i = \frac{\Delta t}{2^{k_i}}$.
This insures $k_{\max} \leq k$ (as long as there is no derefinement).

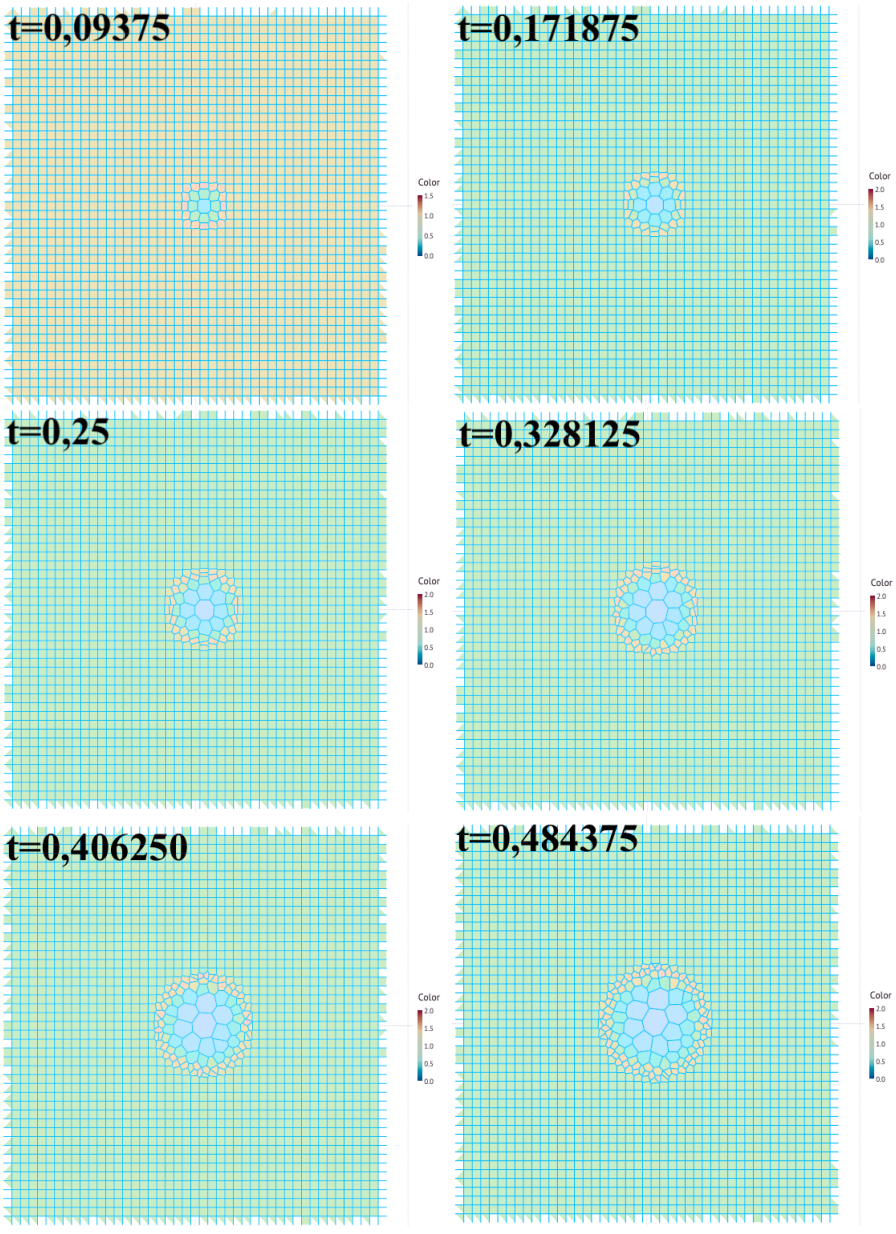
D Numerical test

The code is uploaded on [14].
I conducted one test : the point explosion. As a first experiment I need to control if my code responds the same way as in [1] for the most simple calculation.
N.B. : I could not avoid four negativity errors :

- The pressure (resp. mass) negativity of the centre of mass of the cells after a timestep when I compute the intensive variables in step **VII/ 7- (v)** (resp. when I compute the conservative variables in step **VII/ 5-**). To prevent those errors from making the algorithm crash (if they are not significant with an absolute value $|P| < 10^{-4}$). I brutally set the pressure back to zero and compute back the corresponding conservative variables. As for the mass I used a flux limiter to avoid that the mass become negative.

- The pressure and mass errors when I predict the value of the intensive variables further in time and space on the centre of the faces (step **VII/ 3-**). This is due to the fact that for some gradients, even limited, the value at the centre of the interface is zero for pressure and mass (this is due to the construction of the slope-limiter), and it can become negative with the time prediction.

We can see in Figure 3 that after a first symmetric part of the point explosion, between t=0,25 and t=0,328125 the simulation becomes non-symmetric (Figure 4). This non-symmetricity is probably due to a flaw in my code but it might also due to the capacity of the CPUs I used for the parallelization. Indeed, computation errors might occur in the Riemann solver which would cause this loss of symmetricity.


t=0,09375
t=0,171875
t=0,25
t=0,328125
t=0,406250
t=0,484375
Color
2.0
1.5
1.0
0.5
0.0

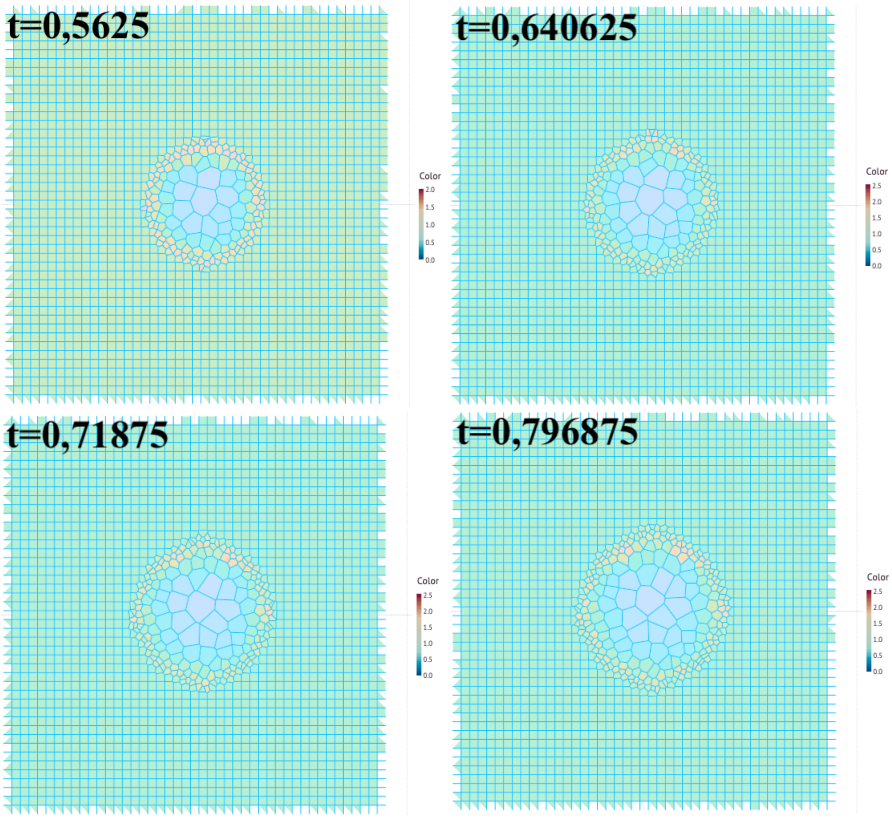


FIGURE 3 – Point explosion test, $P = 1, \rho = 1$

E Conclusion

In this thesis I have tried to clarify as much as possible the main mathematical aspects of the AREPO method for solving the Euler equation. To do so, I tried to underline the steps from a basic Lagrangian moving mesh finite volume method to something more complex in terms of mathematical theory, but with strong gains in time and precision for the computation. This complexity is particularly present in the use of the geometry of the mesh to compute both the gradient and the Riemann problem more easily as well as the introduction of individual timestepping which is an enormous gain in time. Furthermore the techniques adjusting the speeds of the mesh-generating points, in particular the Poisson equation resolution, are part of this goal to enhance stongly the precision thanks only to some apriori mathematical work.

In the end, the implementation I produced in Julia is able to give results, but I think they can be majorly improved. In particular, I had to cope with the inevitable negativity of non-negative variables occuring systematically at some point of my computations as well as the non symmetry preserving, which traduces underlying flaws in my code.

Yet I think I have achieved to give here a simple and clear implementation in Julia as well as all the potential to generalize the code to greater number of conservation laws.

F Bibliography

[1] Springel,V. : E pur si muove : Galilean-invariant cosmological hydrodynamical simulations on a moving mesh, *Monthly Notices of the Royal Astronomical Society*, Volume 401, Issue 2,Pages 791–851, January 2010

[2] Chainet-Hillairet,C. : Finite volume schemes for a nonlinear hyperbolic equation. Convergence towards the entropy solution and error estimate, *ESAIM : M2AN* **33**, Pages 129-156, 1999

[3] Kröner,D.,Ohlberger,M. : A Posteriori Error estimates for upwind finite volume schemes for nonlinear conservation laws in multi dimensions, *Mathematics of Computation* **69**, Pages 25-39, 2000

[4] Vijayan,P.,Kallinderis,Y. : A 3D finite-volume scheme for the Euler equations on adaptive tetrahedral grids, *Journal of Computational Physics* 113(2), 249-267, 1994

[5] Torrilhon,M. : Krylov : Riemann solver for large hyperbolic systems of conservation laws, *SIAM Journal on Scientific Computing* **34(4)**, Pages 2072–A2091, 2012

[6] Schmidtmann,B.,Torrilhon,M. : A Hybrid Riemann Solver for Large Hyperbolic Systems of Conservation Laws, *SIAM Journal of Scientific Computing* **39(6)**, A2911–A2934, 2017

[7] Toro,E.,F. : MUSTA : A multi-stage numerical flux, *Applied Numerical Mathematics* **56**, pp. 1464– 1479, 2006

[8] Torrilhon,M. : Two-dimensional bulk microflow simulations based on regularized Grad's 13- moment equations, *Multiscale Modelling Simulation* **5**, pp. 695–728, 2006

[9] https ://github.com/astrowq/arepo

[10] https ://github.com/JuliaGeometry/GeometricalPredicates.jl

[11] https ://github.com/JuliaParallel/MPI.jl

[12] https ://github.com/JuliaGeometry/VoronoiDelaunay.jl

[13] https ://github.com/rdeits/RegionTrees.jl

[14] https ://github.com/EzRo7511/ArepoJulia