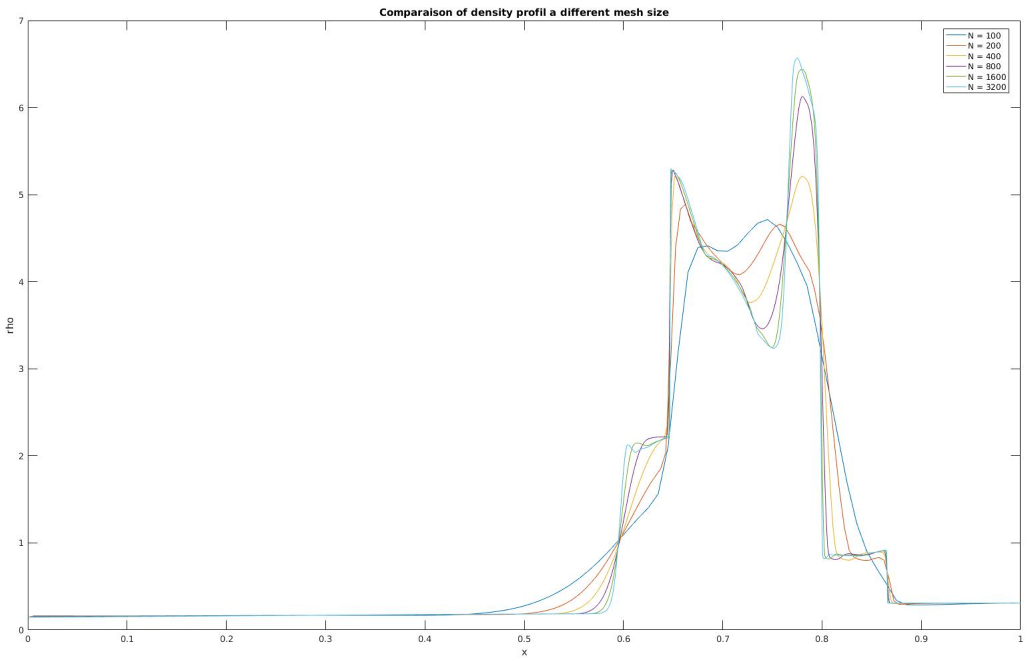
Internship Repport

Non regular Nodes method for Euler Gas Equation

Universität zu Köln
INSA de Rouen Normandie - 2017



Author :
Timothée SCHMODERER

Referent :
Gregor GASSNER
XXXX XXXX

Contents

1 Introduction

2 Euler's Gas Equation

Let's begin with a brief introduction about the euler gas equations (see [4]).
These equations were first described by Euler in 1757. They describe fluid behaviour when the flow is adiabatic, without momentum exchange by viscosity or energy exchange by thermal conduction. That is to say, it especially work well with gas.
For a one dimensional system, a flow in a tube, the governing equations are the following :

$$\boxed{\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ P+\rho u^2\\ u(E+P)\end{pmatrix}=0 \qquad \forall x\in\Omega,\ \forall t\geq 0} \tag{1}$$

Actually, these equations are just a formulation for the conservation of mass, momentum and energy across time and space.

Where :

1. ρ stands for the fluid density is strictly positive.
2. P stands for the fluid pressure and is also strictly positive.
3. E is the total energy of the system
4. u is the fluid velocity which could be either positive (if gas is moving toward increasing value in space) or negative.

All this variable depends of time and space. We omit this dependence for lighter notation.

The system (1) is close by a fourth equation, a thermodynamics equation of state which bound the four variable together :

$$\boxed{E=\frac{P}{\gamma-1}-\frac{\rho u^2}{2}} \tag{2}$$

Where γ is the heat capacity ratio (or adiabatic index) of the fluid. It will be taken equal to : 1.4.
We will also need to compute the speed of sound in the gas, which is given by :

$$c=\sqrt{\frac{\gamma P}{\rho}} \tag{3}$$

In general there is no analytic solution of the system (1 - 2). That's why we need numerical scheme that can good approximate the solution. Moreover usually we can evaluate a numerical scheme by comparing the result of a computation to the analytic solution but as we don't know it, we will have to consider different kind of convergence and error analysis of our numerical scheme.
The system (1) describe non linear hyperbolic partial differential equations. A new difficulty is that, even if the initial conditions are smooth, the solution could present discontinuities,

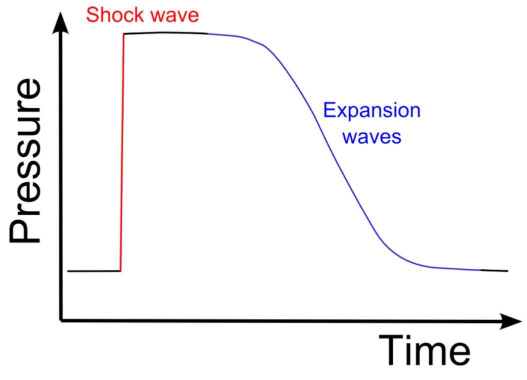


FIGURE 1 – Instance of shock waves

or in a more physical vocabulary, shock waves. This is a major problem to construct numerical method to capture them.

To put the system 1 is a more compact way, let $U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}$ be the state vector of the system and $f(U) = \begin{pmatrix} \rho u \\ P + \rho u^2 \\ u(E+P) \end{pmatrix}$ be the flux vector such as (1) rewrite as :

$$\frac{\partial U}{\partial t} + \frac{\partial f(U)}{\partial x} = 0$$

This a general notation for hyperbolic systems. For instance, in one dimension if we choose f to be the identity then we get the advection equation :

$$u_t + u_x = 0$$

The problem is complete with appropriate initial conditions and boundary conditions. I will describe the different kind of boundary conditions we will meet in this work :

1. **Periodic :** the easiest conditions. It is simply that what append at one side of the domain is reported at the other side :

$$\Phi(a,t) = \Phi(b,t) \qquad \Phi = u,\ \rho \text{ and } \ P$$

2. **Solid wall :** the most natural conditions. The fluid will behave like there is two solid wall at the two sides of the tubes. In term of equation, it is (assume $\Omega = [a,b]$) :

$$u(a,t) = u(b,t) = 0$$
$$\frac{\partial \Phi}{\partial x}(a,t) = \frac{\partial \Phi}{\partial x}(b,t) = 0 \qquad \Phi = \rho \text{ and } P$$

3. **Inlet :** this condition prescribe the value of the flow at the boundaries. It is like there is some source of fluids :

$$\Phi(x_B,t) = \Phi_{in}(t) \qquad \Phi = u,\ \rho \text{ and } \ P$$

4. **Outflow :** All the fluids that reaches the boundary x_B is evacuate. In terms of equations :

$$\frac{\partial \Phi}{\partial x}(x_B,t) = 0 \qquad \Phi = u,\ \rho \text{ and } P$$

3 Numerical scheme

The goal is to implement the scheme describe in [2] and to test it on different cases. Notice that the method describe in [2] concentrate on moving boundary in two dimensions. But we forgot that and focus on the case the boundaries are fixed and the dimension is one.

3.1 General method for hyperbolic system

Many methods have been developed to solve hyberbolics equations. We can think about finite volume method which are very appropriate for these kind of problem, but also finite differences, finite element, discontinuous Galerkin ..
The method we are interested in use finite differences. This kind of method are in general easy to implement, very efficient on Cartesian grid and very easy to make high order.

3.1.1 Cells and Nodes

Choose a integer N (obviously not zero, we won't go far), and divide Ω into N equals grid "cells" (Figure 2), let x_i be the cell-centers and Δx be the cell's width.

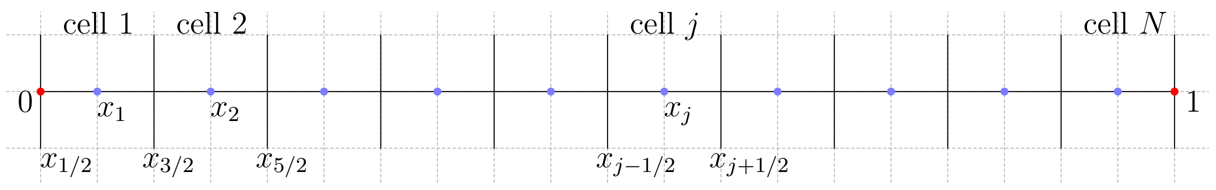


FIGURE 2 – Cells construction on Ω

We denote by $x_{j\pm1/2}$ the cells interfaces : $x_j \pm \dfrac{\Delta x}{2}$.

I will now focus on the scheme propose in the paper I am interested in [2].

The goal is to get the following equations :

$$\boxed{\frac{dU_j}{dt} = -\frac{\hat{f}_{j+1/2} - \hat{f}_{j-1/2}}{\Delta x} \qquad j = 1, \ldots, N \qquad (4)}$$

Where, of course, $f_{j\pm1/2} = f(U(x_{j\pm1/2}, t))$. And, by analogy with finite volume method, the hat ˆ denote that this is our numerical approximation of the flux.
Remark : This form seems very natural, the time variation of the state in a cell is equal to the difference between the in-flux and the out-flux at the cell interfaces.
Reader familiar with finite volume method will notice the strong parallel with this equations.
We compute the $\{\hat{f}_{j\pm1/2}\}$ as follows :

1. Compute the spectral radius of the Jacobian at $x = x_j$. Analytic computation lead us to know that for compressible Euler equation it is :

$$a_j = |u_j| + c_j$$

2. We split the flux function f as :

$$f(U_j) = f_j^+ + f_j^- \qquad f_j^\pm = \frac{1}{2}\left(f(U_j) \pm a_j U_j\right)$$

3. We compute the slopes in each cells using the minmod function :

$$(f_x)_j^\pm = minmod\left(\theta\frac{f_j^\pm - f_{j-1}^\pm}{\Delta x}, \frac{f_{j+1}^\pm - f_{j-1}^\pm}{2\Delta x}, \theta\frac{f_{j+1}^\pm - f_j^\pm}{\Delta x}\right)$$

   Where $\theta \in [1,2]$ is a parameter that control the amount of numerical dissipation, larger values of θ typically lead to sharper resolution of discontinuities, but may cause some oscillations, it will be taken at 1.5.
   The *minmod* function is defined by :

$$minmod(a,b,c) = \begin{cases} \min(a,b,c) & \text{if } a > 0,\ b > 0 \text{ and } c > 0 \\ \max(a,b,c) & \text{if } a < 0,\ b < 0 \text{ and } c < 0 \\ 0 & \text{else} \end{cases}$$

4. Construct f^E and f^W as :

$$f_j^E = f_j^+ + \frac{\Delta x}{2}(f_x)_j^+ \qquad f_j^W = f_j^- - \frac{\Delta x}{2}(f_x)_j^-$$

5. Finally :

$$\hat{f}_{j+1/2} = f_j^E + f_{j+1}^W \quad \hat{f}_{j-1/2} = f_{j-1}^E + f_j^W$$

3.2 Ghost Nodes

While presenting the scheme I haven't discuss on which nodes we have to apply the method. Obviously there is no probleme when we are in the middle of the domain problems arrises when we come close to the boundaries.
In order to achieve the method two ghosts point are needed on each side, with value dicted by the boundary conditions.
Why two ? Remember that we want values at each cell's interfaces. Then for instance to get $\hat{f}_{1-1/2}$ we will need f_0^E so at this point we need one ghost cell left. Moreover to get f_0^E we will need to get the slopes for $j = 0$ then we might need $f_{-1}^{\pm}$ in the *minmod* function. So one more ghost cell.
I describe the comportement on the left side, the same can be applied to the right side.

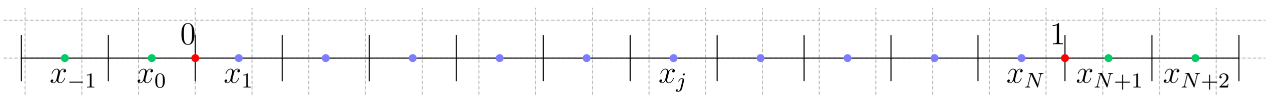


FIGURE 3 – Ω with ghost points

The whole method can be summarize in this array (red are ghost values, and dot show for what j we can compute the value) :

cell	-1	0	1	2	$\dots$	j	$\dots$	$N-1$	N	$N+1$	$N+2$
a_j	·	·	·	·	·	·	·	·	·	·	·
$f^{\pm}$	·	·	·	·	·	·	·	·	·	·	·
$f_j^{\pm} - f_{j-1}^{\pm}$		·	·	·	·	·	·	·	·	·	·
$f_{j+1}^{\pm} - f_{j-1}^{\pm}$		·	·	·	·	·	·	·	·	·	
$f_{j+1}^{\pm} - f_j^{\pm}$	·	·	·	·	·	·	·	·	·	·	
$(f_x)^{\pm}$		·	·	·	·	·	·	·	·	·	
f^E		·	·	·	·	·	·	·	·	·	
f^W		·	·	·	·	·	·	·	·	·	
$\hat{f}_{j+1/2}$		·	·	·	·	·	·	·	·		
$\hat{f}_{j-1/2}$			·	·	·	·	·	·	·	·	
$\dfrac{dU_j}{dt}$			·	·	·	·	·	·	·		

3.3 Boundary condition

By the boundary condition we get the value of ρ, u and P at the two ghost cells centers :

$$\begin{aligned} \rho_0 &= \rho_1 & u_0 &= -u_1 & P_0 &= P_1 \\ \rho_{-1} &= \rho_2 & u_{-1} &= -u_2 & P_{-1} &= P_2 \end{aligned}$$

Which in term of U is the following :

$$U_0 = \begin{pmatrix} U_1^1 \\ -U_1^2 \\ U_1^3 \end{pmatrix} \qquad U_{-1} = \begin{pmatrix} U_2^1 \\ -U_2^2 \\ U_2^3 \end{pmatrix}$$

where the upperscript is the denotes the i-th component.

3.4 Integration of the semi-discrete system

Once we get the scheme 4 we have to numerically integrate it. Here I present two method. The easiest way to do it is with the Euler forward scheme. We took a time discretization $\{t_n = n\Delta t\}$ and we approximate the time derivative by :

$$\frac{dU_j}{dt} \approx \frac{U_j^{n+1} - U_j^n}{\Delta t}$$

Thus the final discrete equation we get is :

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{\Delta x}\left(\hat{f}_{j+1/2} - \hat{f}_{j-1/2}\right)$$

However this method present one major inconvenient, depending on the choice of Δt the numerical method could be unstable (that is to say, unbounded oscilations appears in the numerical solution). Thus we will have to determine the so called Courant's number which bound the time step to the space step.
Moreover, even with a "good" choose of Δt some oscillation could occurs which are only numerical. this oscillations are due to the choice of the integration scheme. To handle this problem I used a Strong Stability Preserving Runge-Kutta method of order 3 (SSP - RK). It presents two main advantages. First the courant number is the same as the one with the Euler scheme, and second, this method is TVD (total variation diminution) :

$$TV(U^{n+1}) \leq TV(U^n) \qquad TV(U^n) = \sum_j |U_{j+1}^n - U_j^n|$$

I implemented the following scheme from ([3]) :

$$\begin{aligned} u^{(1)} &= u^n + \Delta t L(u^n) \\ u^{(2)} &= \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)}) \\ u^{n+1} &= \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)}) \end{aligned} \tag{5}$$

Where the L operator is the one that give us the second member. This scheme is SSP with the same Courant's number as the Euler Forward scheme and third order in time.

Courant's number : This number is a constant C for which the method is stable and such as :

$$\frac{|u|\Delta t}{\Delta x} \leq C$$

Numerical experiment suggests that $C = 0.437$ was good. Then as we know that approximately the velocity is bound by 23 we can choose the time step to be :

$$\Delta t = \frac{0.437}{23}\Delta x$$

4 Implementation

4.1 Data Representation

I hope no one here is a purist mathematician beacause I will abuse with the "dot" notation of Matlab meaning component by component operation. Choosing Matlab for programming

language was a logical first choice to do, because many operation are transparent in Matlab. However we should keep in mind that greater efficiency could be achieve with mor basic language.
With a very natural way, the partial differential equation is describeb by a vector U whose size is : 3 row times (N+4) colums. Why N+4 ? Because there is N internal nodes and 2 ghost points on each sides.
Thus in only on structure I could represent my system at time t. For instance for $N = 10$ cells and $t = 0$ I get the structure :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.025 & 2500 & 2500 & 0.025 & 0.025 & 0.025 & 0.025 & 0.025 & 0.025 & 0.025 & 0.025 & 250 & 250 & 0.025 \end{pmatrix}$$

Let recall that with that structure we have information in the center of the cell and we want achieve information at cells interfaces.

4.2 Code

I will describe the code implementing the method in Matlab language [1]. I recall that all the operation should be thinking made in one cell. The choice of the matlab language offers us the "element by element" operations which I allow us to compute all the cell in the same operation.

First is teh function implementing the flux : $\begin{pmatrix} \rho u \\ \rho u^2 + P \\ u(E+P) \end{pmatrix}$. Given the vector of state U we can get easily u, ρ and E but we will need the adiabatic constant γ to compute the pressure with the equation of state (2). Thus the flux is given by :

$$f(U) = U.*u + \begin{bmatrix} 0_{\mathbb{R}^{N+4}} \\ P \\ P.*u \end{bmatrix}$$

```
function y = f(U,gamma)
  % Velocity
  v = U(2,:)./U(1,:);
  % Pressure
  P = (gamma-1)*(U(3,:)-0.5*U(1,:).*v.*v);
  % Flux
  y = U.*v+[zeros(size(P));P;P.*v];
end
```

Code 1 – Implementation of the flux function - f.m

Second easy step is to compute the speed of the sound in the domain. Same argumentation leads us to compute density and pressure from the vector of state and the adiabatic constant :

$$c = \sqrt{\gamma P./\rho}$$

1. Actuaally, it will not works with matlab beacause I write it in Octave and this software is more tolerant. But minor changes will make it work with Matlab.

```
function c = speedofsound(U,gamma)
  % Density
  rho = U(1,:);
  % Pressure
  P = (gamma-1)*(U(3,:)-0.5*(U(2,:).^2)./rho);
  % Speed of sound
  c = sqrt(gamma*P./rho);
end
```

Code 2 – Implementation of the computation of the speed of sound - speedofsound.m

Another minor function is the one that construct the initial condition of pressure :

$$P_0(x) = \begin{cases} 1000 & \text{if } x \in]0, 0.1[\\ 0.01 & \text{if } x \in]0.1, 0.9[\\ 100 & \text{if } x \in]0.9, 1[\end{cases}$$

However the function here is not well implemented because it will search thrre times in the x array where are the value. This would be avoid by making only one loop over x. But, I let the function in this form for lighter code.
One important function in the reconstruction process in the *minmod* function. This is maybe the function I am the less proud about because it is certainly using the full power offers by the data structure, but it is not the speedest way to do it and it is strongly dependant of this particular case.

$$minmod(a,b,c) = \begin{cases} min(a,b,c) & \text{if } a > 0,\ b > 0 \text{ and } c > 0 \\ max(a,b,c) & \text{if } a < 0,\ b < 0 \text{ and } c < 0 \\ 0 & \text{else} \end{cases}$$

```
function y = minmod(a,b,c)
  y = zeros(size(a));
  % Index where the three numbers are positive
  iM = find(a > 0 & b > 0 & c > 0);
  y(iM) = min(min(a(iM),b(iM)),c(iM));
  % Index where the three numbers are negative
  im = find(a < 0 & b < 0 & c < 0);
  y(im) = max(max(a(im),b(im)),c(im));
end
```

Code 3 – Implementation of the minmod function - minmod.m

Finally the hearth of the method is contain in this last function file. Given U, the adiabatic constant, the minmod reconstruction constant and the mesh size, it will produce the second member in 4. I just change the input values of the *minmod* function. In deed, in the paper they divided this inputs by Δx but then the results is multiply by Δx to get f^E and f^W. So to gain six division and two multiplication, I directly cancel them.

```
function [q dt] = qf_uniform(U,gamma,theta,dx,cfl)
  fU = f(U,gamma);
  c = speedofsound(U,gamma);
  a = abs(U(2,:)./U(1,:)) + c;

  % Compute f+ & f-
  fp = 0.5 * (fU + a.*U);
  fm = 0.5 * (fU - a.*U);

  % Compute df+
  dfp0 = theta*(fp(:,2:end-1) - fp(:,1:end-2)); % Option 1
  dfp1 = (fp(:,3:end) - fp(:,1:end-2))/2; % Option 2
  dfp2 = theta*(fp(:,3:end) - fp(:,2:end-1)); % Option 3

  dfp = minmod(dfp0,dfp1,dfp2);

  % Compute df-
  dfm0 = theta*(fm(:,2:end-1) - fm(:,1:end-2)); % Option 1
  dfm1 = (fm(:,3:end) - fm(:,1:end-2))/2; % Option 2
  dfm2 = theta*(fm(:,3:end) - fm(:,2:end-1)); % Option 3

  dfm = minmod(dfm0,dfm1,dfm2);

  fE = fp(:,2:end-1) + 0.5*dfp;
  fW = fm(:,2:end-1) - 0.5*dfm;

  % Compute f+0.5 and f-0.5
  fphalf = fE(:,2:end-1) + fW(:,3:end);
  fmhalf = fE(:,1:end-2) + fW(:,2:end-1);

  % Compute second member
  q = (fphalf - fmhalf)/dx;
  dt = cfl*dx/max(a);
end
```

Code 4 – Implementation of the computation of the second member - qf_uniform.m

And least the main function is setting all the parameters for the study of this case and process the SSP Runge Kutta method.

```
function main_euler_uniform(N)
  % Constantes
  a = 0;
  b = 1;
  theta = 1.5;
  gamma = 1.4;
  T = 0.1;
  dx = (b-a)/N;
  x = [a-dx/2:dx:b+dx/2];
  x = [x(1)-dx,x,x(end)+dx];
  cfl = 0.437/23;
  dt = cfl*dx;
  s = size(x);
  niter = ceil(T/dt);

  % Initialisation in Omega at t=0
  tmprho = ones(1,s(2)-4);
  tmpv = zeros(1,s(2)-4);
  tmpP = P0(x(3:end-2));
  % Apply boundary condition to fill the ghost cells
  rho = [tmprho(2),tmprho(1),tmprho,tmprho(end), tmprho(end-1)];
  v = [-tmpv(2),-tmpv(1),tmpv,-tmpv(end),-tmpv(end-1)];
  P = [tmpP(2),tmpP(1),tmpP,tmpP(end),tmpP(end-1)];
  E = P/(gamma-1) + 0.5*rho.*v.*v;
  % Construction of the vector of state
  U = [rho;rho.*v;E];

  % U at the next time step
  U1 = zeros(size(U));
  % Reserve memory space for the RK method
  U1_1 = zeros(size(U));
  U1_2 = zeros(size(U));

  for t = 1:niter

    % SSP RK order 3
    q = qf_uniform(U,gamma,theta,dx);
    U1_1(:,3:end-2) = U(:,3:end-2) - dt*q;
    U1_1(:,[1 2 end-1 end]) = [1;-1;1].*U1_1(:,[4 3 end-2 end-3]);

    q1 = qf_uniform(U1_1,gamma,theta,dx);
    U1_2(:,3:end-2) = 0.75*U(:,3:end-2) + 0.25*U1_1(:,3:end-2) -
       0.25*dt*q1;
    U1_2(:,[1 2 end-1 end]) = [1;-1;1].*U1_2(:,[4 3 end-2 end-3]);

    q2 = qf_uniform(U1_2,gamma,theta,dx);
    U1(:,3:end-2) = (1/3)*U(:,3:end-2) + (2/3)*U1_2(:,3:end-2) -
       (2/3)*dt*q2;
    U1(:,[1 2 end-1 end]) = [1;-1;1].*U1(:,[4 3 end-2 end-3]);
    c = speedofsound(U,gamma);
    % Loop
    U = U1;

    if sum(imag(c) > 0) >0 % in case of instability of the method,
       this criterion will stop the loop
      break;
    end
  end
end % end function
```

Code 5 – Implementation of the main function - main_euler_uniform.m

Note that each at each Runge-Kutta time step I apply the boundary condition to U.

5 Numerical Experiment for regular nodes distribution

I recall that more results and some animations are available at https://github.com/tschmoderer/euler-prj.

5.1 shock capture

I ran many cases with different mesh size. The bigger the mesh is, the more accurate is the solution computed.

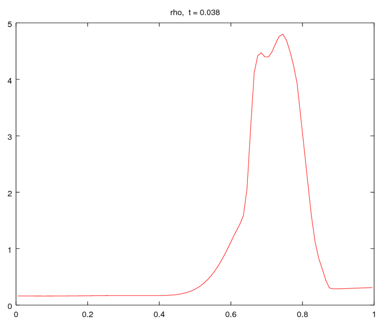


FIGURE 4 – shock at $t = 0.038$ for $N = 100$ nodes

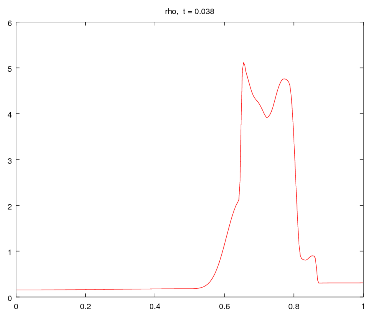


FIGURE 5 – shock at $t = 0.038$ for $N = 300$ nodes

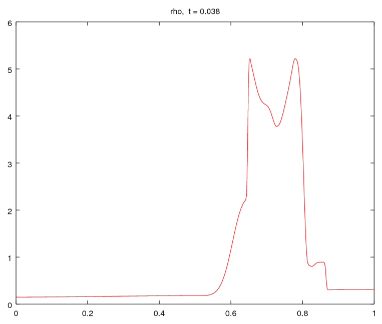


FIGURE 6 – shock at $t = 0.038$ for $N = 400$ nodes

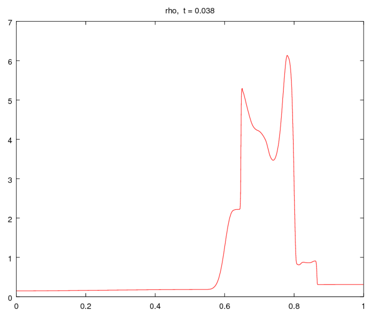


FIGURE 7 – shock at $t = 0.038$ for $N = 800$ nodes

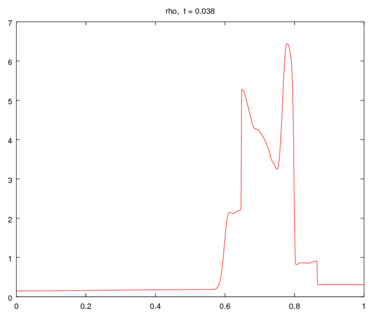


FIGURE 8 – shock at $t = 0.038$ for $N = 1600$ nodes

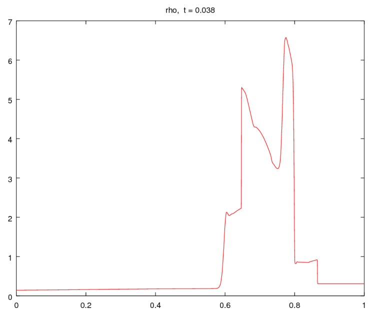


FIGURE 9 – shock at $t = 0.038$ for $N = 3200$ nodes

5.2 Euler vs Runge Kutta

Let's illustrate the influence of the choice of the integration method. We can see on figure 10 that some oscillations occurs in the computed solution with Euler's forward scheme, but that there is no on figure 11 where the solution was computed with a third order SSP Runge-Kutta method.

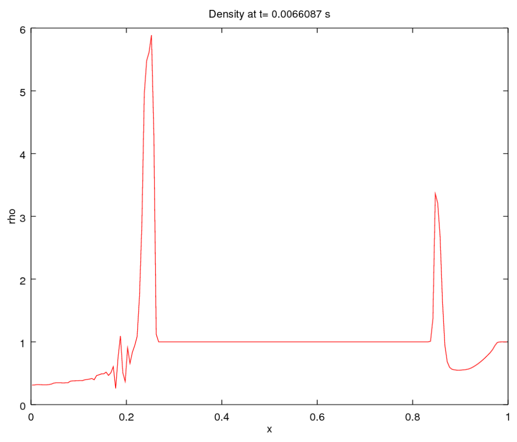


FIGURE 10 – Density with Euler forward scheme

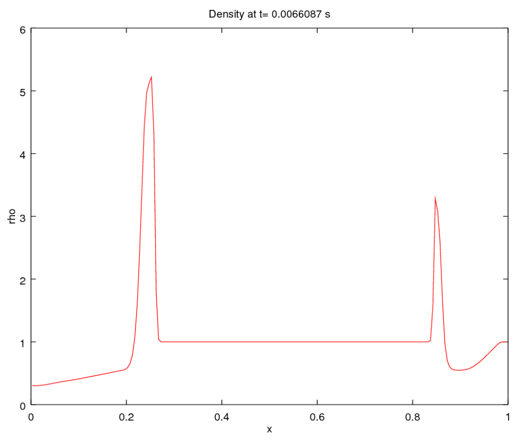


FIGURE 11 – Density with the SSP RK scheme

5.3 Influence of θ

5.4 Error and convergence

As we saw earlier, finding convergence rate is quite difficult. Because we don't know the analytics solution, we have to adopt a particular strategy. The choosen strategy is the one called "*Manufactured Solution*". The general receipe is to choose the solution of the Euler's equation and to add a corresponding source term to the equations.
Let's detail more the procedure :

1. We choose q as smooth as possible to be the solution of 1.
2. We put q in these equations and get a non zeros result (otherwise, it will mean we have an analytical solution, which is in general unknown). Let's call the rest $\mathcal{S}(x,t)$.
3. We now consider the modified problem :

$$\boxed{\begin{gathered}\frac{\partial U}{\partial t}+\frac{\partial f(U)}{\partial x}=\mathcal{S}\\ U(x,0)=q(x,0)\\ \text{Plus boundary conditions to be discusses bellow}\end{gathered}} \tag{6}$$

   For which we know the solution q.
4. We apply our method to this problem. It requires minor modification in the code to add the source term.
5. We can now compare the numerical solution to the analytical one. And compute the rate convergence.

In order to avoid problem with the boundary, we adopt periodic boundaries for these problems.

5.4.1 Case 1

In this first case we want the solution to be :

$$\begin{array}{rcl} \rho(x,t) & = & 1+0.2\sin(2\pi(x-t)) \\ u(x,t) & = & 1 \\ P(x,t) & = & 1 \end{array}$$

Then a little computation using the equation of state (2) gives the solution for the energy :

$$E(x,t) = \frac{\gamma+1}{2(\gamma-1)} + 0.1\sin(2\pi(x-t))$$

Hence the boundary and initial conditions are the following :

$$\boxed{\begin{array}{l} \rho(0,t) = \rho(1,t) \qquad \forall t \\ u(0,t) = u(1,t) \qquad \forall t \\ P(0,t) = P(1,t) \qquad \forall t \\ \rho(x,0) = 1+0.2\sin(2\pi x) \quad \forall x \in \Omega \\ u(x,0) = 1 \quad \forall x \in \Omega \\ P(x,0) = 1 \quad \forall x \in \Omega \end{array}} \tag{7}$$

In this particular case, the source term is identically zero :

$$\mathcal{S}(x,t) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

We make computation for : 100, 200, 400, 800, 1600, 3200, 6400 nodes up to 1000 iterations. . Hence we are able to compare the exact solution for the density and the numerical approximation. Figure 12 shows the log of the error against the log of the number of nodes.

5.4.2 Case 2

The second case will comforte us in the fact tat the method is of order 2. We want the solution to be :

$$\begin{array}{rcl} \rho(x,t) & = & 2+0.1\sin(2\pi(x-t)) \\ u(x,t) & = & 1 \\ E(x,t) & = & 2+0.1\cos(2\pi(x-t)) \end{array}$$

Then, the initial and boundary conditions are the following :

$$\boxed{\begin{array}{l} u(0,t) = u(1,t) \qquad \forall t \\ \rho(0,t) = \rho(1,t) \qquad \forall t \\ P(0,t) = P(1,t) \qquad \forall t \\ u(x,0) = 1 \quad \forall x \in \Omega \\ \rho(x,0) = 2+0.1\sin(2\pi x) \quad \forall x \in \Omega \\ P(x,0) = \frac{\gamma-1}{20}\left(20+2\cos(2\pi x) - sin(2\pi x)\right) \quad \forall x \in \Omega \end{array}} \tag{8}$$

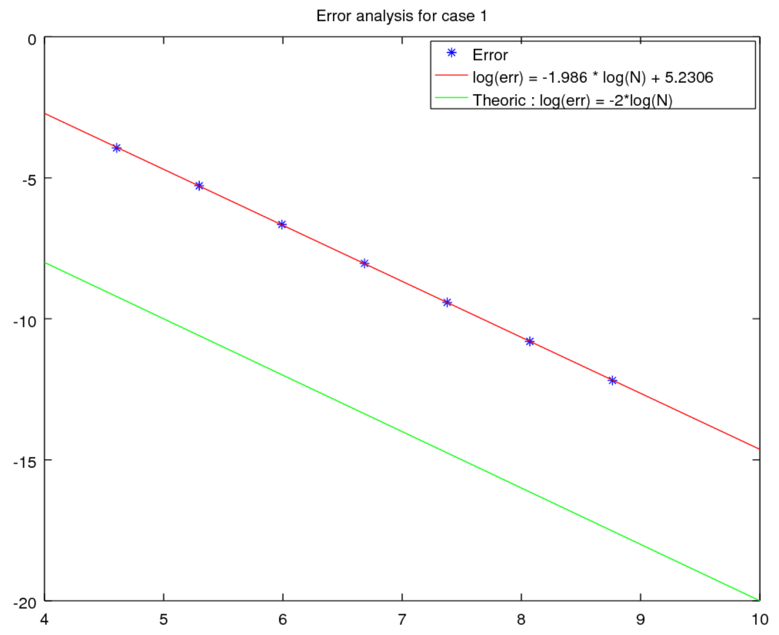


FIGURE 12

And the source term is given by :

$$\boxed{\mathcal{S}(x,t) = (1-\gamma)\pi(2\rho(x,t) + E(x,t) - 6)\begin{pmatrix}0\\1\\1\end{pmatrix} \qquad (9)}$$

We again make computation for : 100, 200, 400, 800, 1600, 3200, 6400 nodes up to 1000 iterations. . Hence we are able to compare the exact solution and the numerical approximation. Figure 13 shows the log of the error against the log of the number of nodes.

5.4.3 Rate of convergence

6 Adaptation on non regular nodes distribution

6.1 My adaptation

6.2 Legendre Gauss ...

tyjtykt

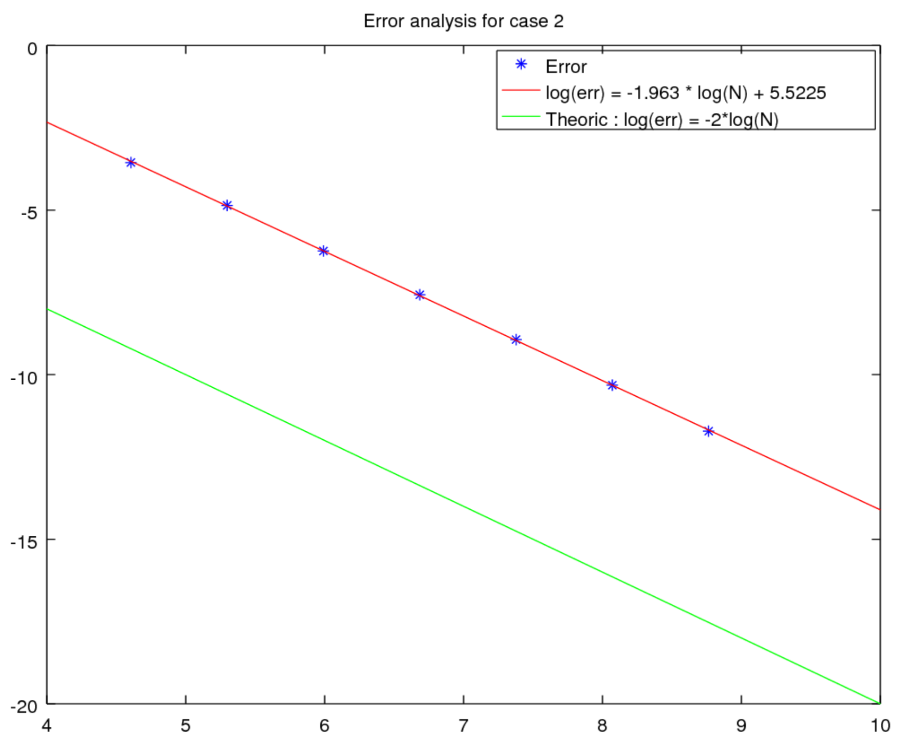


Figure 13

7 Conclusion

Références

[1] A. Baeza, P. Mulet, and D. Zorío. High order boundary extrapolation technique for finite difference methods on complex domains with cartesian meshes. *Journal of Scientific Computing*, 66(2) :761–791, Feb 2016.

[2] Kurganov Alexander Russo Giovanni Coco Armando, Chertock Alina. A second-order finite-difference method for compressible fluids in domains with moving boundaries. 2017.

[3] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1) :89–112, 2001.

[4] Wikipedia. Euler equations (fluid dynamics) — wikipedia, the free encyclopedia, 2017. [Online ; accessed 26-June-2017].

[5] Wikipedia. Shock wave — wikipedia, the free encyclopedia, 2017. [Online ; accessed 26-June-2017].

Table des figures

List of Codes