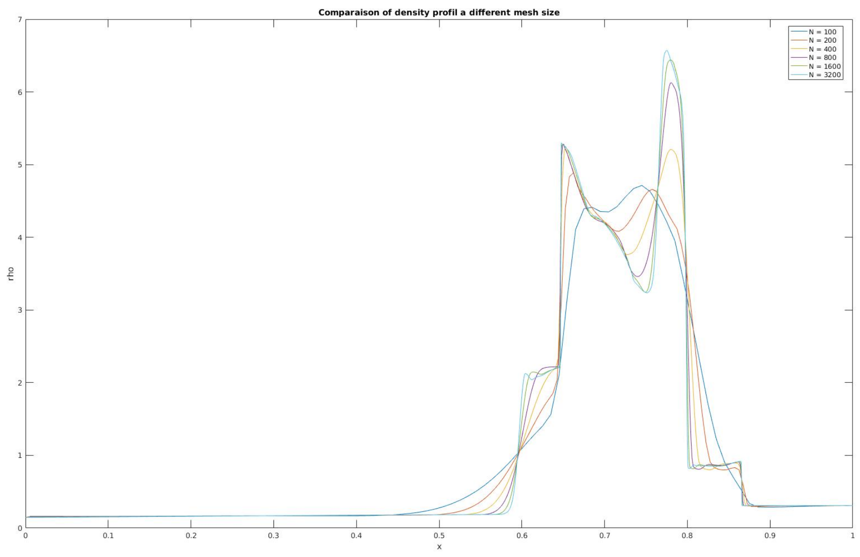
Internship Repport

Non regular Nodes method for Euler Gas Equation

Universität zu Köln
INSA de Rouen Normandie - 2017



Author :
Timothée Schmoderer

Referent :
Gregor Gassner
XXXX XXXX

Table des matières

1 Introduction

2 Euler Gas Equation

Let's begin with a brief introduction about the euler gas equations ([2]). These equations descibe fluid behaviour when we could consider that there is no viscosity. That is to say especially for gas. For a one dimensionnal system, the equations are written as follow :

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

Convention : This system illustrates a convention we will systematically make. The partial derivative are intends to act on each components of the vectors.

Where :

1. ρ stands for the density is strictly positive.
2. P stands for the presure and is also strictly positive.
3. E is the internal energy
4. u is the gas 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. The system (1) is close by a fourth equation, an equation of state which bound pressure, density and energy :

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

Where γ is some gas constant coming from adiabatic. It will be taken equal to : 1.4.
We will also need to compute the speed of sound in the gas, this is givern by :

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

In general there is no analytic solution of this system, which will lead us to knew technique of convergence rate analysis.
The system 1 describe non linear hyberpolics partial differential equations. The difficulties is that even is the initial conditiosn are smooth, the solution could present discontinuities, or in a more physical vocabulary, chock 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 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 :

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

2.1 The problem

In this etud, the domain is taken to be : $\Omega = [0, 1]$ and the time in $[0, 0.1]$ The problem is closed with proper boundary and initial conditions :

$$\boxed{\begin{aligned} u(0) = u(1) &= 0 \\ \left.\frac{\partial \rho}{\partial x}\right|_{x=0,1} &= 0 \\ \left.\frac{\partial P}{\partial x}\right|_{x=0,1} &= 0 \\ u(x,0) &= 0 \quad \forall x \in \Omega \\ \rho(x,0) &= 1 \quad \forall x \in \Omega \\ P(x,0) &= \begin{cases} 1000 & \text{if } 0 < x < 0.1 \\ 0.01 & \text{if } 0.1 < x < 0.9 \\ 100 & \text{if } 0.9 < x < 1 \end{cases} \end{aligned} \qquad (3)}$$

Which we can interpret as solid wall conditions on both extremities of the tubes.

2.2 Expecting results

The initial condition suggests that at $t = 0$ there is some kind of valves at $x = 0.1$ and $x = 0.9$ that are oppened. After we can easy imagine that due to pressure the gas "want" to uniform his energy, so the left side will be moving to the right, the right side to the left. But what happend when this two wave colliddes ?
In fact due to the hyperbolic characteristics of the system 1 we can expect chock waves. The numerical difficulties is to capture them.

3 Numerical scheme

3.1 General method for hyberbolic system

3.1.1 Cells and Nodes

Choose a integer N (obviously not null, we won't go far), and let divide Ω into N grid "cells" (Figure 1), and let x_i be the cell-centers. A simple reasonnement will lead us to the main scheme. Let us integrate over one cell the partial derivative equation :

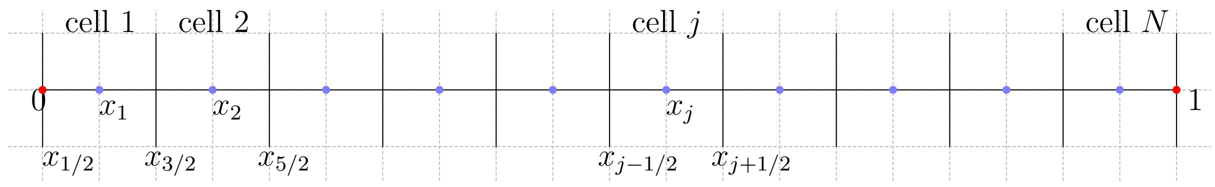


FIGURE 1 – Cells construction on Ω

$$\int_{x_{j-1/2}}^{x_{j+1/2}} \frac{\partial U}{\partial t} dx = -\int_{x_{j-1/2}}^{x_{j+1/2}} \frac{\partial f(U)}{\partial x} dx$$
$$\frac{d}{dt}\int_{x_{j-1/2}}^{x_{j+1/2}} U dx = -\int_{x_{j-1/2}}^{x_{j+1/2}} \frac{\partial f(U)}{\partial x} dx$$

Here the magic appears, on left side we almost reconize the mean value of U in the cell, and on the right side the computation is obvious since we integrate a space derivative with respect to space. Hence :

$$\Delta x \frac{d\bar{U}}{dt} = -(f(U(x_{j+1/2}, t)) - f(U(x_{j-1/2}, t)))$$

Where $\bar{U}$ is the mean value of U in the j cell, and $\Delta x = \frac{1}{N}$ is the cell length.

The difficulty that arrises is that we know the values of U in the cell center by setting it to be the mean value of U in the cell. However we need to have the values of U at the cells interfaces in order to compute the right member.

3.1.2 Reconstruction

In order to solve the precedent problem we have to "guess" the value of U at the cell interfaces withe the knowledge of value in the center. The first possibility is to considet U to be constant over each cell : Thus another problem arise, we construct two value for U

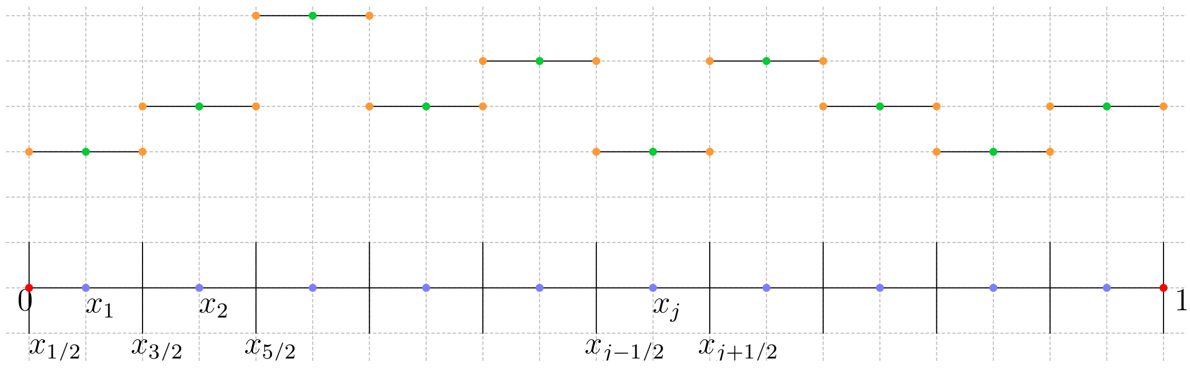


FIGURE 2 –

at the cells interfaces, which pne we should choose ?
Second possibility is to consider piecewise linear function between centers : We solve the problem of multiple value at interfaces but we see that we would need a special treatement for the two extrems nodes.

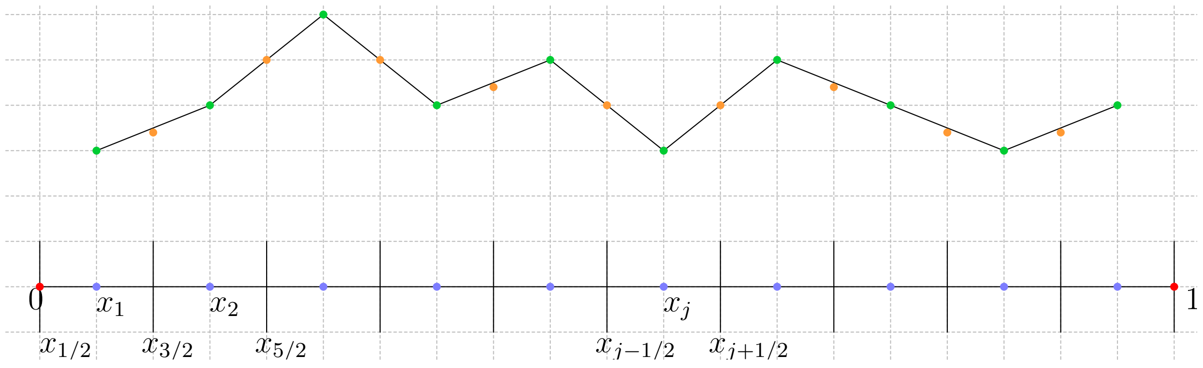


Figure 3 –

In fact, there infinetly many ways to do this reconstruction procedure (with paraboloids, cubics, splines ...) so I will now focus on the scheme propose in the paper.
The Goal is to get the following equations :

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

We compute the $\{\hat{f}_{j\pm 1/2}\}$ as follows :

1. Compute the spectral radius of the jacobian at $x = x_j$. Analytics 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 aparameter that control te amount of numerical dissipation, larger values of θ typically lead to sharper resolution of discontinuities, but may cause some oscillations.
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 Boundary condition and Ghost Nodes

While presenting the scheme I haven't discuss on which nodes we have to applys the method. Obvioustly 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.

4 Numerical Experiment for regular nodes distribution

4.1 Data Representation

I hope no one is a purist mathematician beacause I will abuse with the "dot" notation of Matlab meaning point by point 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.

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 Implementation

I will describe the code implementing the method in Matlab language [1].

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 :

$$f(U) = U. * u + \begin{bmatrix} 0 \\ P \\ P. * u \end{bmatrix}$$

```
function y = f(U,gamma)
v = U(2,:)./U(1,:);
P = (gamma-1)*(U(3,:)-0.5*U(1,:).*v.*v);
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.

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,:);
v2 = (U(2,:)./rho).^2;
% 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.

```
function y = P0(x)
l = length(x);
for i = 1:l
z = x(i);
if 0 < z && z < 0.1
y(i) = 10^3;
elseif 0.1 < z && z < 0.9
y(i) = 10^-2;
elseif 0.9 < z && z < 1
y(i) = 10^2;
else
y(i) = 0;
end
end
end
```

Code 3 – Implementation of the initial condition of pressure - P0.m

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 not using the full power offers by the data structure and is strongly dependant of this particular case.

```
function y = minmod(a,b,c)
s = size(a);
y = zeros(s);
for i=1:s(1)
for j = 1:s(2)
% limit data access
tmpa = a(i,j);
tmpb = b(i,j);
tmpc = c(i,j);
if tmpa > 0 && tmpb > 0 && tmpc > 0
y(i,j) = min([tmpa,tmpb,tmpc]);
elseif tmpa < 0 && tmpb < 0 && tmpc < 0
y(i,j) = max([tmpa,tmpb,tmpc]);
else
y(i,j) = 0;
end
end
end
```

```
end
```

Code 4 – 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

```
function q = qf_uniform(U,gamma,theta,dx)
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))/dx; % Option 1
dfp1 = (fp(:,3:end) - fp(:,1:end-2))/(2*dx); % Option 2
dfp2 = theta*(fp(:,3:end) - fp(:,2:end-1))/dx; % Option 3

dfp = minmod(dfp0,dfp1,dfp2);

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

dfm = minmod(dfm0,dfm1,dfm2);

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

Code 5 – 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 and boundary conditions
tmprho = ones(1,s(2)-4);
rho = [tmprho(2),tmprho(1),tmprho,tmprho(end), tmprho(end-1)];
tmpv = zeros(1,s(2)-4);
%tmpv = 0.5*exp(-200*(x(3:end-2)-0.5).^2);
v = [-tmpv(2),-tmpv(1),tmpv,-tmpv(end),-tmpv(end-1)];
tmpP = P0(x(3:end-2));
%tmpP = ones(1,s(2)-4);
P = [tmpP(2),tmpP(1),tmpP,tmpP(end),tmpP(end-1)];

clear tmpP tmprho tmpv

E = P/(gamma-1) + 0.5*rho.*v.*v;

U = [rho;rho.*v;E];
%%% Let's Go %%%
U1 = zeros(size(U)); % Reserve mem space
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]);

rho = U(1,3:end-2);
v = U(2,3:end-2)./rho;
P = (gamma-1)*(U(3,3:end-2) - 0.5*rho.*v.*v);
c = speedofsound(U,gamma);

% Loop
```

```
U = U1;
if sum(imag(c) > 0) >0 % in case of instability of the method, this
    criterion will stopes the loop
break;
end

% Save chock case
if (t*dt <= 0.038 && dt*(t+1) > 0.038)
plot(x(3:end-2),rho,'r');
title(['rho,␣␣t␣=␣',num2str(t*dt)]);
print(['../img/chock_',num2str(N),'_Nodes.png'],"-dpng");
dlmwrite(['../Results/Uniform/',num2str(N),'␣Nodes/chock_rho.dat'
   ],[rho'],'␣')
end
if (t*dt > 0.045)
close all;
end
end
end % end function
```

Code 6 – Implementation of the main function - main_ euler_ uniform.m

4.3 Results

4.3.1 Euler vs Runge Kutta

4.3.2 Rate of convergence

4.3.3 Chock capture

4.3.4 Error and convergence

5 Adaptation on non regular nodes distribution

6 Conclusion

Références

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

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

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

Table des figures

List of Codes