Approximation implicite des Equations d'Euler 2D par une méthode de relaxation sur maillages non-structurés

ANGO Laurent

22 septembre 2010

Rapport de stage

Sup Galilée, Villetaneuse

CEA, Saclay

Ecole d'ingénieur, Troisième année,
Mathématiques Appliquées et Calcul Scientifique

Encadrants : CHALONS Christophe, KOKH Samuel

Table des matières

Remerciements

Je voudrais remercier Christophe Chalons et Samuel Kokh qui ont assuré mon encadrement durant ce stage. Le contact avec eux a indéniablement enrichi ma culture mathématique numérique.

Je remercie Christophe pour m'avoir guidé dans l'aspect mathématique du stage, pour m'avoir facilité l'appropriation de la méthode de relaxation pour l'approximation des équations d'Euler, d'abord en 1D avant le passage en 2D grâce à des explications claires et concises.

Je remercie Samuel pour m'avoir donné, d'abord, cette opportunité de faire ce stage au CEA et, puis bien sûr, pour tous les nombreux conseils et explications donnés au cours des différentes étapes de l'élaboration du code.

1 Introduction

1.1 Contexte du stage

Le CEA est un centre de recherche scientifique français dont les activités recouvrent de nombreux domaines tels que la défense, les technologies de l'information, la santé et bien sûr l'énergie. Le CEA est fondé en octobre 1945 et devient officiellement en mars 2010 le « Commissariat à l'énergie atomique et aux énergies alternatives ». Cet organisme est implanté sur plusieurs centres dans toute la France. Si les équipes travaillant au CEA sont regroupées autour d'une thématique technologique ou scientifique, il est important de noter qu'elles sont pluridisciplinaires et regroupent des compétences qui sont la modélisation des problèmes (biologie, mécanique, astrophysique), l'analyse des modèles (mathématiques appliquées), la simulation, la réalisation d'expériences, etc....

Parmi les activités du CEA, les technologies liées à la production d'énergie nucléaire sont un élément essentiel qui fait partie des missions originelles du CEA. Notre stage qui s'intéresse à la simulation numérique d'écoulements de fluides s'inscrit dans cette thématique.

Les écoulements de fluides sont nombreux dans les centrales nucléaires. Rappelons brièvement le principe de fonctionnement d'une centrale nucléaire. Une centrale a pour but de produire de l'électricité. Pour cela, elle utilise la chaleur issue de la fission de noyaux au sein de combustibles comme l'uranium ou le plutonium pour faire tourner un alternateur qui produit de l'électricité. Cette transformation thermomécanique de l'énergie n'est rendue possible que par la circulation du fluide caloporteur qui fait le lien entre le réacteur et l'alternateur en circuit fermé. Ainsi, il est important pour l'étude du comportement d'un réacteur en régime nominal (amélioration des performances) ou en régime accidentel (étude de sûreté) de pouvoir simuler les écoulements de fluides au sein du réacteur.

1.2 Objectifs du stage

Les objectifs relèvent de deux catégories. Il y a tout d'abord des objectifs de R&D qui consistent à étudier l'application de certaines techniques d'approximation d'EDP. Le second volet des objectifs concerne les techniques de génie logiciel, via l'apprentissage et l'utilisation de méthodes de programmation rigoureuses et des outils développés.

1.2.1 Techniques d'approximation d'EDP par relaxation

Les techniques d'approximation d'EDP par des méthodes de relaxation ont été l'objet de nombreux travaux ces dernières années. Néanmoins, ces techniques ne sont pas encore très utilisées pour les simulations de thermohydrauliques comme celles qui concernent les applications du nucléaire civil.
L'objectif de ce stage consiste à implémenter le schéma de relaxation décrit dans [2] dans un code 2D de calcul qui permet de se rapprocher des conditions d'utilisation d'un code d'étude. Afin de respecter cette exigence, nous avons choisi :

- de travailler avec des maillages non-structurés de triangles ;
- d'implémenter une version explicite et une version implicite en temps de l'algorithme choisi.

1.2.2 Génie logiciel

Les outils développés au cours de ce stage sont destinés à une pérennité qui dépasse le cadre de cette étude que ce soit pour leur utilisation ou leur développement.
Soulignons d'abord que le développement du code est parti d'une feuille blanche : aucune structure préexistante n'a été utilisée.
Aussi, notre travail a été encadré par des règles de programmation d'autant plus strictes que nous n'avions pas à composer avec un système logiciel préexistant. La liste des règles adoptées sont les suivantes :

- gestion de versions : tout le développement du code a été « versionné » grâce au logiciel Mercurial. Le travail sous gestion de versions est particulièrement utile lorsqu'il s'agit de structurer ou partager un travail collaboratif entre plusieurs personnes ;
- gestion de sytème de compilation : l'outil CMake a été choisi pour gérer toute la compilation du code. Ceci permet de bien découper le projet en modules de code munis de leurs propres instructions de compilation sans avoir à gérer les problèmes complexes de dépendances croisées ;
- tests unitaires : utilisation quasi-systématique de tests unitaires au cours du développement. A cette fin, nous avons utilisé la librairie CUnit qui permet d'automatiser de manière simple la gestion de ces tests ;
- librairie PETSc : la gestion de la résolution des systèmes linéaires est laissée à la librairie PETSc. Ceci permet de bénéficier d'un très large éventail d'algorithmes de résolution et de préconditionnement. Notons que le code lui-même ne repose pas sur PETSc et que les appels à PETSc sont très « localisés ». Ainsi, il est facile de compiler le code ne faisant pas appel aux inversions de systèmes sans PETSc (solver explicite par exemple) tout comme il est facile de remplacer les appels à PETSc par ses propres routines ;
- structure de données : attention particulière à donner aux noms de fonctions et de variables, découpage des fonctions principales en sous-routines sans pour autant dénaturer l'esprit de l'algorithme, respect d'une arborescence clairement définie des dossiers, réduire le plus possible le champ d'action des fonctions pour des soucis de flexibilité et de cohérence ;
- modularité : volonté de rendre le code le plus modulaire possible. Le code est divisé en plusieurs librairies statiques qui produisent le programme final. Il est prévu de mettre au point le chargement des solveurs et des conditions aux limites via des librairies dynamiques. Ce souci de modularité exige qu'un soin particulier soit pris pour la définition des structures de données du code.

2 Modélisation de l'écoulement

On modélise les écoulements d'un fluide par les équations d'Euler.

2.1 Equations d'Euler en deux dimensions

Les équations d'Euler en deux dimensions s'écrivent de la manière suivante :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial(\rho u)}{\partial x} + \dfrac{\partial(\rho v)}{\partial y} = 0 \\ \dfrac{\partial(\rho u)}{\partial t} + \dfrac{\partial(\rho u^2 + p)}{\partial x} + \dfrac{\partial(\rho uv)}{\partial y} = 0 \\ \dfrac{\partial(\rho v)}{\partial t} + \dfrac{\partial(\rho uv)}{\partial x} + \dfrac{\partial(\rho v^2 + p)}{\partial y} = 0 \\ \dfrac{\partial(\rho E)}{\partial t} + \dfrac{\partial(\rho Eu + pu)}{\partial x} + \dfrac{\partial(\rho Ev + pv)}{\partial y} = 0 \end{cases} \tag{1}$$

avec $p = p(\rho, \rho e)$. On pose $\mathbf{x} = (x, y)$. On a donc pour $\mathbb{U} = (\rho, \rho u, \rho v, \rho E)^t$, $F(\mathbb{U}) = (\rho u, \rho u^2 + p, \rho uv, \rho Eu + pu)^t$ et $G(\mathbb{U}) = (\rho v, \rho uv, \rho v^2 + p, \rho Ev + pv)^t$:

$$\partial_t \mathbb{U} + \partial_x F(\mathbb{U}) + \partial_y G(\mathbb{U}) = 0$$

Les inconnues sont ρ (masse volumique), $\vec{\mathbf{v}} = (u, v)$ (vitesse), qui possède une composante u selon x et une composante v selon y, et E (énergie totale). On notera aussi le volume spécifique τ comme étant l'inverse de la densité ρ. La première équation découle de la conservation de la masse, la deuxième et la troisième de la conservation de la quantité de mouvement (la deuxième loi de Newton), et la dernière de la conservation de l'énergie. Toutefois, ce système d'équations ne suffit pas pour caractériser le comportement du fluide car il faut aussi connaître son équation d'état qui fait intervenir la pression p. D'autres grandeurs peuvent être utilisées dans le cadre des écoulements de fluides comme la température T ou la vitesse du son c.

Exemple Dans le cas particulier du gaz parfait diatomique la pression p et la vitesse du son c s'expriment de la manière suivante :

$$\begin{aligned} p(\rho, e) &= (\gamma - 1) e \rho \\ c(p, \rho) &= \sqrt{\frac{\gamma p}{\rho}} \qquad \text{avec} \quad \gamma = 1.4 \end{aligned}$$

L'ensemble de ces grandeurs sont récapitulées dans la figure 1 avec leurs unités respectives.

2.2 Propriétés mathématiques des équations d'Euler 1D

Avant d'aborder directement la question de l'approximation des équations d'Euler en 2D, rappelons certains résultats théoriques dans le cas 1D. Pour plus de détails, les propriétés mathématiques des équations d'Euler 1D sont largement étudiées dans [4]. Le système d'équations d'Euler en 1D se présente de la manière suivante :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial(\rho u)}{\partial x} = 0 \\ \dfrac{\partial(\rho u)}{\partial t} + \dfrac{\partial(\rho u^2 + p)}{\partial x} = 0 \\ \dfrac{\partial(\rho E)}{\partial t} + \dfrac{\partial(\rho Eu + pu)}{\partial x} = 0 \end{cases} \tag{2}$$

notation	grandeur physique	unité	dimension
x	longueur horizontale	m	L
y	longueur verticale	m	L
t	temps	s	T
ρ	masse volumique	$\mathrm{kg.m^{-3}}$	$\mathrm{M.L^{-3}}$
u	composante horizontale de la vitesse	$\mathrm{m.s^{-1}}$	$\mathrm{L.T^{-1}}$
v	composante verticale de la vitesse	$\mathrm{m.s^{-1}}$	$\mathrm{L.T^{-1}}$
$\tau = \frac{1}{\rho}$	volume spécifique	$\mathrm{m^3.kg^{-1}}$	$\mathrm{L^3.M^{-1}}$
E	énergie totale spécifique	$\mathrm{J.kg^{-1}}$	$\mathrm{L^2.T^{-2}}$
$e = E - \frac{u^2+v^2}{2}$	énergie interne spécifique	$\mathrm{J.kg^{-1}}$	$\mathrm{L^2.T^{-2}}$
p	pression	Pa	$\mathrm{M.T^{-2}.L^{-1}}$
c	vitesse du son	$\mathrm{m.s^{-1}}$	$\mathrm{L.T^{-1}}$
T	température	K	θ

FIG. 1 – Grandeurs physiques rencontrées pour les écoulements

avec $p = p(\rho, \rho e)$. Le système (2) correspond à l'écriture sous forme conservative des équations d'Euler 1D et peut être ramené à cette écriture vectorielle :

$$\partial_t \mathbb{U} + \partial_x F(\mathbb{U}) = 0$$

avec $\mathbb{U} = (\rho, \rho u, \rho E)^t$ et $F(\mathbb{U}) = (\rho u, \rho u^2 + p, \rho E u + p u)^t$

2.2.1 Cas général : les gaz réels

Matrice jacobienne.

Soit $\mathbb{U} \in \mathbb{R}^3$.

$$\mathbb{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix}$$

Soit $F : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$,

$$F(\mathbb{U}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho E u + p u \end{pmatrix}$$

De plus,

$$\begin{aligned} p &= p(\rho, e) \\ c^2 &= \frac{\partial p}{\partial \rho}_{|e} + \frac{p}{\rho^2} \frac{\partial p}{\partial e}_{|\rho} \end{aligned}$$

Par conséquent, $\nabla F : \mathbb{R}^3 \longrightarrow \mathcal{M}_{3,3}(\mathbb{R})$

$$A(\mathbb{U}) = \nabla F(\mathbb{U}) = \begin{pmatrix} 0 & 1 & 0 \\ -u^2 + K & u(2-k) & k \\ u(K-H) & H - u^2 k & u(1+k) \end{pmatrix}$$

avec

$$\begin{aligned} H &= E + \frac{p}{\rho} \\ k &= \frac{1}{\rho}\frac{\partial p}{\partial e}_{|\rho} \\ K &= c^2 + k(u^2 - H) \end{aligned}$$

On peut alors montrer que $A(\mathbb{U})$ a pour valeurs propres $\lambda_1 = u - c$, $\lambda_2 = u$ et $\lambda_3 = u + c$ associées respectivement aux vecteurs propres suivants :

$$K_1 = \begin{pmatrix} 1 \\ u-c \\ H - uc \end{pmatrix} \quad K_2 = \begin{pmatrix} 1 \\ u \\ H - \frac{c^2}{k} \end{pmatrix} \quad K_3 = \begin{pmatrix} 1 \\ u+c \\ H + uc \end{pmatrix}$$

Ondes du problème de Riemann associé.

Il existe 3 familles d'ondes possibles dans le système d'équations d'Euler (voir figure 2).

1. Discontinuité de contact.
   Elle est associée à la valeur propre $\lambda_2 = u$. Dans ce cas, on arrive à montrer avec les invariants de Riemann (3) que la vitesse et la pression restent constantes tandis que seule la densité peut subir une discontinuité :

$$\frac{d\rho}{1} = \frac{d(\rho u)}{u} = \frac{d(\rho E)}{H - \frac{c^2}{k}} \tag{3}$$

   - $\dfrac{d(\rho u)}{u} = \dfrac{u d\rho + \rho du}{u} = d\rho + \dfrac{\rho}{u} du$ et $\dfrac{d(\rho u)}{u} = d\rho$

   donc $du = 0$ soit $u =$ constante

   - $dp = \dfrac{\partial p}{\partial e}_{|\rho} de + \dfrac{\partial p}{\partial \rho}_{|e} d\rho$ et $e = E - \frac{u^2}{2}$

   $dp = \dfrac{\partial p}{\partial e}_{|\rho} (dE - udu) + \dfrac{\partial p}{\partial \rho}_{|e} d\rho$ or $du = 0$

   $dp = \dfrac{\partial p}{\partial e}_{|\rho} dE + \dfrac{\partial p}{\partial \rho}_{|e} d\rho$

   de plus, $d(\rho E) = d\rho\Big(H - \dfrac{c^2}{k}\Big)$ soit $\rho dE = \dfrac{p}{\rho} d\rho - \dfrac{c^2 d\rho}{\frac{1}{\rho}\frac{\partial p}{\partial e}_{|\rho}}$

   donc $dE = \Big(\dfrac{p}{\rho^2} - \dfrac{c^2 d\rho}{\frac{\partial p}{\partial e}_{|\rho}}\Big) d\rho$

   donc $dp = \Big(\dfrac{p}{\rho^2}\dfrac{\partial p}{\partial e}_{|\rho} - c^2 + \dfrac{\partial p}{\partial \rho}_{|e}\Big) d\rho$ or $c^2 = \dfrac{\partial p}{\partial \rho}_{|e} + \dfrac{p}{\rho^2}\dfrac{\partial p}{\partial e}_{|\rho}$

   donc $dp = 0$ soit $p =$ constante

2. Onde de choc
   Elle est associée aux valeurs propres $\lambda_1 = u - c$ et $\lambda_3 = u + c$. Dans ce cas, la densité, la vitesse et la pression peuvent subir une discontinuité.

3. Onde de raréfaction
   Elle est associée aux valeurs propres $\lambda_1 = u - c$ et $\lambda_3 = u + c$. Dans ce cas, la densité, la vitesse et la pression peuvent subir une transition continue entre l'état gauche et l'état droit.

Equation de la pression.

Nous pouvons l'établir en manipulant les équations d'Euler :

$$\begin{aligned}
\frac{\partial p}{\partial t} &= \frac{\partial e}{\partial t}\frac{\partial p}{\partial e}_{|\rho} + \frac{\partial \rho}{\partial t}\frac{\partial p}{\partial \rho}_{|e} \\
&= \frac{\partial (E - \frac{u^2}{2})}{\partial t}\frac{\partial p}{\partial e}_{|\rho} - \frac{\partial (\rho u)}{\partial x}\frac{\partial p}{\partial \rho}_{|e} \\
&= \left(\frac{\partial E}{\partial t} - u\frac{\partial u}{\partial t}\right)\frac{\partial p}{\partial e}_{|\rho} - \frac{\partial (\rho u)}{\partial x}\frac{\partial p}{\partial \rho}_{|e}
\end{aligned}$$

Or la troisième équation de (2) permet d'écrire :

$$\partial_t E = \frac{1}{\rho}\left[-\partial_x(\rho E u + p u) - E\partial_t \rho\right]$$

La première équation de (2) permet la simplification suivante :

$$\partial_t E = -u\partial_x E - \frac{1}{\rho}\partial_x(pu) \tag{4}$$

Par ailleurs, la deuxième équation de (2) nous donne :

$$u\partial_t \rho + \rho\partial_t u + 2\rho u\partial_x u + u^2\partial_x \rho + \partial_x p = 0$$

En factorisant par u et ρ, on obtient :

$$u(\partial_t \rho + \rho\partial_x u + u\partial_x \rho) + \rho(\partial_t u + u\partial_x u + \frac{1}{\rho}\partial_x p) = 0$$

On reconnaît alors la première équation de (2), ce qui nous permet de déduire que :

$$\partial_t u + u\partial_x u + \frac{1}{\rho}\partial_x p = 0 \tag{5}$$

Ainsi, les équations (4) et (5) donnent :

$$\begin{aligned}
\frac{\partial p}{\partial t} &= \left[u\Big(-\frac{\partial E}{\partial x} + u\frac{\partial u}{\partial x}\Big) - \frac{p}{\rho}\frac{\partial u}{\partial x}\right]\frac{\partial p}{\partial e}_{|\rho} - \frac{\partial (\rho u)}{\partial x}\frac{\partial p}{\partial \rho}_{|e} \\
&= \left(-u\frac{\partial e}{\partial x} - \frac{p}{\rho}\frac{\partial u}{\partial x}\right)\frac{\partial p}{\partial e}_{|\rho} - \rho\frac{\partial u}{\partial x}\frac{\partial p}{\partial \rho}_{|e} - u\frac{\partial \rho}{\partial x}\frac{\partial p}{\partial \rho}_{|e} \\
&= -u\frac{\partial p}{\partial x} - \rho\frac{\partial u}{\partial x}\frac{\partial p}{\partial \rho}_{|e} - \frac{p}{\rho}\frac{\partial u}{\partial x}\frac{\partial p}{\partial e}_{|\rho} \\
&= -u\frac{\partial p}{\partial x} - \rho\frac{\partial u}{\partial x}\left(\frac{\partial p}{\partial \rho}_{|e} + \frac{p}{\rho^2}\frac{\partial p}{\partial e}_{|\rho}\right) \\
&= -u\frac{\partial p}{\partial x} - \rho c^2\frac{\partial u}{\partial x}
\end{aligned}$$

La pression p obéit donc à l'équation suivante :

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \rho c^2\frac{\partial u}{\partial x} = 0$$

Connaissant la première équation du système (2), on arrive facilement à établir que :

$$\frac{\partial (\rho p)}{\partial t} + \frac{\partial (\rho p u + \rho^2 c^2 u)}{\partial x} = 0$$

2.2.2 Cas particulier : les gaz parfaits.

Matrice jacobienne.

Soit $\mathbb{U} \in \mathbb{R}^3$.

$$\mathbb{U} = \begin{pmatrix} \rho \\ q \\ r \end{pmatrix} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix}$$

Soit $F : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$,

$$F(\mathbb{U}) = \begin{pmatrix} q \\ \frac{q^2}{\rho}\frac{3-\gamma}{2} + (\gamma-1)r \\ \gamma\frac{rq}{\rho} - (\gamma-1)\frac{q^3}{2\rho^2} \end{pmatrix} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho E u + p u \end{pmatrix}$$

Par conséquent, $\nabla F : \mathbb{R}^3 \longrightarrow \mathcal{M}_{3,3}(\mathbb{R})$

$$A(\mathbb{U}) = \nabla F(\mathbb{U}) = \begin{pmatrix} 0 & 1 & 0 \\ \frac{-q^2(3-\gamma)}{2\rho^2} & \frac{q(3-\gamma)}{\rho} & \gamma-1 \\ \frac{-\gamma r q}{\rho^2} + \frac{(\gamma-1)q^3}{\rho^3} & \frac{\gamma r}{\rho} - \frac{3(\gamma-1)q^2}{2\rho^2} & \frac{\gamma q}{\rho} \end{pmatrix}$$

La matrice jacobienne $A(\mathbb{U})$ peut se réécrire en fonction de c et de u :

$$A(\mathbb{U}) = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)u^2 & (3-\gamma)u & \gamma-1 \\ \frac{1}{2}(\gamma-2)u^3 - \frac{c^2 u}{\gamma-1} & \frac{3-2\gamma}{2}u^2 + \frac{c^2}{\gamma-1} & \gamma u \end{pmatrix}$$

On peut alors montrer que $A(\mathbb{U})$ a pour valeurs propres $\lambda_1 = u - c$, $\lambda_2 = u$ et $\lambda_3 = u + c$ associées respectivement aux vecteurs propres suivants :

$$K_1 = \begin{pmatrix} 1 \\ u - c \\ \frac{\rho E + p}{\rho} - uc \end{pmatrix} \quad K_2 = \begin{pmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{pmatrix} \quad K_3 = \begin{pmatrix} 1 \\ u + c \\ \frac{\rho E + p}{\rho} + uc \end{pmatrix}$$

Ondes du problème de Riemann associé.

Il existe 3 familles d'ondes possibles dans le système d'équations d'Euler (voir figure 2).

1. Discontinuité de contact.
   Elle est associée à la valeur propre $\lambda_2 = u$. Dans ce cas, on arrive à montrer avec les invariants de Riemann (6) que la vitesse et la pression restent constantes tandis que seule la densité peut subir une discontinuité :

$$\frac{d\rho}{1} = \frac{d(\rho u)}{u} = \frac{d(\rho E)}{\frac{1}{2}u^2} \tag{6}$$

   - $\dfrac{d(\rho u)}{u} = \dfrac{u d\rho + \rho du}{u} = d\rho + \dfrac{\rho}{u}du$ et $\dfrac{d(\rho u)}{u} = d\rho$

   donc $du = 0$ soit $u =$ constante

   - $d(\rho E) = \rho u du + \dfrac{u^2}{2}d\rho + \rho de + e d\rho$ et $dp = (\gamma-1)(\rho de + e d\rho)$

   donc $d(\rho E) = \rho u du + \dfrac{u^2}{2}d\rho + \dfrac{dp}{\gamma-1}$ or $d(\rho E) = \dfrac{u^2}{2}d\rho$

   donc $\rho u du + \dfrac{dp}{\gamma-1} = 0$ or $du = 0$

   donc $dp = 0$ soit $p =$ constante

2. Onde de choc
   Elle est associée aux valeurs propres $\lambda_1 = u - c$ et $\lambda_3 = u + c$. Dans ce cas, la densité, la vitesse et la pression peuvent subir une discontinuité.
3. Onde de raréfaction
   Elle est associée aux valeurs propres $\lambda_1 = u - c$ et $\lambda_3 = u + c$. Dans ce cas, la densité, la vitesse et la pression peuvent subir une transition continue entre l'état gauche et l'état droit.

Equation de la pression.
L'équation de la pression dans le cas du gaz parfait donne :

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + \rho\gamma(\gamma-1)\Big(E - \frac{u^2}{2}\Big)\frac{\partial u}{\partial x} = 0$$

ou encore

$$\frac{\partial(\rho p)}{\partial t} + \frac{\partial}{\partial x}\left(\rho p u + \rho^2\gamma(\gamma-1)\Big(E - \frac{u^2}{2}\Big)u\right) = 0$$

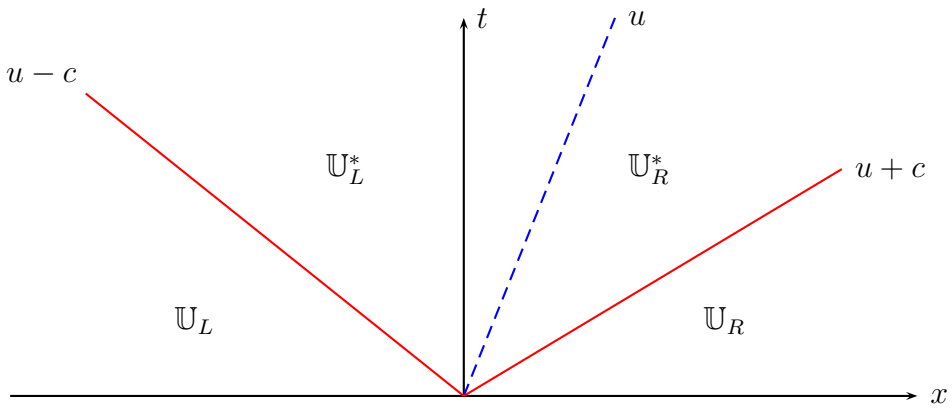


FIG. 2 – Structure de la solution à un problème de Riemann constituée de trois ondes pour le système d'équations d'Euler

2.3 Approximation des équations d'Euler 1D par relaxation

La mise en œuvre de la méthode de relaxation présentée ici reprend les idées et les résultats contenus dans [2]. Elle consiste à travailler non pas directement sur la solution du système (2) mais sur celle d'un système, qu'on dit relaxé et qui contient une inconnue et une équation supplémentaires par rapport à (2), que l'on fait tendre vers la solution de (2) en appliquant une relaxation raide sur la nouvelle inconnue.

2.3.1 Le système d'équations d'Euler relaxé

La relaxation consiste à remplacer p par une nouvelle inconnue π que l'on fait évoluer selon sa propre EDP. Celle-ci est choisie de telle sorte que π soit proche de p. Idéalement,

on utiliserait l'EDP propre à la pression pour faire évoluer π mais le facteur $\rho^2 c^2$ n'étant pas linéaire, on préfèrera le remplacer par a^2 avec a une constante positive dont le choix est abordé un peu plus loin. Ainsi, le système d'équations d'Euler relaxé s'écrit de la manière suivante :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial (\rho u)}{\partial x} = 0 \\ \dfrac{\partial (\rho u)}{\partial t} + \dfrac{\partial (\rho u^2 + \pi)}{\partial x} = 0 \\ \dfrac{\partial (\rho E)}{\partial t} + \dfrac{\partial (\rho E u + \pi u)}{\partial x} = 0 \\ \dfrac{\partial (\rho \pi)}{\partial t} + \dfrac{\partial (\rho \pi u + a^2 u)}{\partial x} = \dfrac{\rho}{\delta}(p - \pi) \end{cases} \tag{7}$$

avec $\frac{1}{\delta} > 0$ le coefficient de relaxation. Le système (7) peut être ramené à cette écriture vectorielle :

$$\partial_t \mathbf{u} + \partial_x f(\mathbf{u}) = \frac{1}{\delta} \mathcal{Z}(\mathbf{u})$$

avec $\mathbf{u} = (\mathbb{U}, \rho\pi)^t = (\rho, \rho u, \rho E, \rho\pi)^t$, $f(\mathbf{u}) = (\rho u, \rho u^2 + \pi, \rho E u + \pi u, \rho \pi u + a^2 u)^t$ et $\mathcal{Z}(\mathbf{u}) = (0, 0, 0, \rho(p - \pi))^t$. L'idée est maintenant de chercher la solution de ce nouveau système en faisant tendre δ vers zéro. Pour y arriver, nous adoptons une stratégie de splitting qui se decompose en deux étapes.

2.3.2 Les deux étapes du splitting

Au premier pas, on résout le problème de Riemann associé au système relaxé (7) en omettant le second membre, ce qui revient à résoudre le système suivant :

$$\partial_t \mathbf{u} + \partial_x f(\mathbf{u}) = 0$$

Au second pas, on résout le système suivant en tenant compte du second membre avec δ qui tend vers zéro :

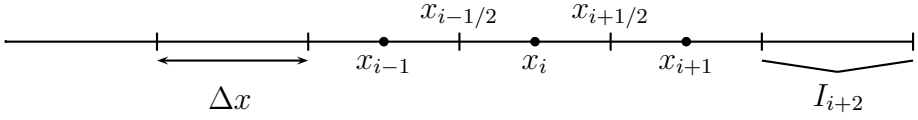
$$\partial_t \mathbf{u} = \frac{1}{\delta} \mathcal{Z}(\mathbf{u})$$

3 Discrétisation par une méthode volumes finis dans le cas 1D

3.1 Convention

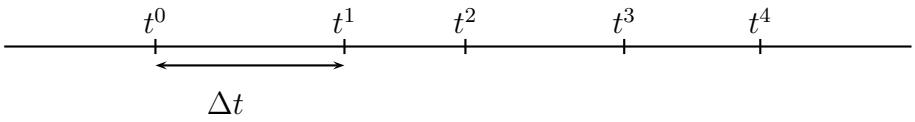
On se donne un maillage 1D spatial qui suit cette convention pour le segment $[a, b]$ divisé en N cellules I_i :

$$\begin{cases} I_i = [x_{i-1/2}, x_{i+1/2}] & \text{cellule} \\ \Delta x = x_{i+1/2} - x_{i-1/2} = \frac{b-a}{N} & \text{pas d'espace} \\ x_{i+1/2} = a + i\Delta x, \ \ i \in \{0, ..., N\} & \text{interface} \\ x_i = a + (i - 1/2)\Delta x, \ \ i \in \{1, ..., N\} & \text{centre des cellules} \end{cases}$$



On a besoin aussi d'une discrétisation temporelle : $t^0 = t_0$, $t^1 = t^0 + \Delta t$, $t^2 = t^1 + \Delta t$

jusqu'à atteindre le temps final t_f. On peut remarquer ici que l'on prend en général un pas d'espace fixe mais il ne peut pas être de même pour le pas de temps qui doit respecter un critère qu'on appelle CFL. Cette question est abordée un peu plus tard dans le rapport.



Le but de notre schéma numérique sera de calculer une approximation $\mathbb{U}_j^n$ de la moyenne de la solution exacte au temps $t = t^n$ sur chacune des cellules I_j du domaine discrétisé et jusqu'au temps final $t = t_f$:

$$\mathbb{U}_j^n \approx \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} \mathbb{U}(x, t^n) dx$$

3.2 Donnée initiale

On part d'une donnée initiale connue $\mathbb{U}(x, 0)$ et on considère à la première itération :

$$\mathbb{U}_j^0 = \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} \mathbb{U}(x, 0) dx$$

3.3 Traitement spécifique de l'inconnue π

La méthode numérique utilisée ici se décompose en deux étapes dans son passage de t^n à t^{n+1}.

$t^n \rightarrow t^{n+1-}$

La première étape consiste à appliquer une relaxation raide à l'inconnue π. Pour cela, on résout le système suivant :

$$\begin{cases} \partial_t \rho = 0 \\ \partial_t (\rho u) = 0 \\ \partial_t (\rho E) = 0 \\ \delta \partial_t (\rho \pi) = p - \pi \text{ avec } \delta \rightarrow 0 \end{cases}$$

Connaissant la donnée initiale $\mathbb{U}_j^n$ sur chaque cellule I_j, on peut donc établir que :

$$\begin{cases} \rho^{n+1-} = \rho^n \\ (\rho u)^{n+1-} = (\rho u)^n \\ (\rho E)^{n+1-} = (\rho E)^n \\ \pi^{n+1-} = p(\rho^n, E^n - \dfrac{u^{n2}}{2}) \end{cases}$$

Cette étape nous permet de définir uniquement π en utilisant la loi de pression p. Les autres inconnues ne bougent pas.

$t^{n+1-} \longrightarrow t^{n+1}$

Pour ce second pas, on résout l'opérateur convectif avec comme donnée initiale la solution calculée dans le premier pas c'est-à-dire $\mathbb{U}_j^{n+1-}$ sur chaque cellule I_j :

$$\begin{cases} \partial_t \rho + \partial_x(\rho u) = 0 \\ \partial_t(\rho u) + \partial_x(\rho u^2 + \pi) = 0 \\ \partial_t(\rho E) + \partial_x(\rho E u + \pi u) = 0 \\ \partial_t(\rho \pi) + \partial_x(\rho \pi u + a^2 u) = 0 \end{cases}$$

Cette étape correspond au paragraphe suivant : on applique la formule de mise à jour.

3.4 Formule de mise à jour

Considérant $\mathbb{U}_j^n$ comme étant connu, le but est de calculer $\mathbb{U}_j$ au temps t^{n+1}. Pour une cellule donnée I_j, on intègre de la manière suivante le système (2) :

$$\int_{t^n}^{t^{n+1}} \int_{x_{j-1/2}}^{x_{j+1/2}} \partial_t \mathbb{U} + \partial_x F(\mathbb{U}) dx dt = 0$$

$$\int_{x_{j-1/2}}^{x_{j+1/2}} \mathbb{U}(x, t^{n+1}) - \mathbb{U}(x, t^n) dx + \int_{t^n}^{t^{n+1}} F(\mathbb{U}(x_{j+1/2}, t)) - F(\mathbb{U}(x_{j-1/2}, t)) dt = 0$$

Le schéma que nous employons ici repose sur la résolution de problèmes de Riemann. Dans cette optique, on donne à la formule de mise à jour cette écriture en utilisant l'approximation propre aux volumes finis :

$$\Delta x(\mathbb{U}_j^{n+1} - \mathbb{U}_j^n) + \Delta t(\mathcal{G}_{j+1/2} - \mathcal{G}_{j-1/2}) = 0$$

avec $\mathcal{G}_{j+1/2}$ une approximation de la moyenne en temps du flux à l'interface $x_{j+1/2}$ calculée à l'aide d'une solution à un problème de Riemann. Au final, la formule de mise à jour devient :

$$\mathbb{U}_j^{n+1} = \mathbb{U}_j^n - \frac{\Delta t}{\Delta x}(\mathcal{G}_{j+1/2}^n - \mathcal{G}_{j-1/2}^n)$$

3.5 Calcul du flux à chaque interface

Avec la relaxation, les familles d'ondes associées au système relaxé ne sont que des discontinuités de contact. Il s'agit là du principal avantage de cette méthode : elle évite l'intervention des ondes de choc et de raréfaction dans la résolution du problème de Riemann. Bien sûr, on paie cette simplification en y ajoutant la relaxation raide de la pression décrite dans le paragraphe 3.3.

$$\mathbb{U}(x/t; \mathbb{U}_j, \mathbb{U}_{j+1}) = \begin{cases} \mathbb{U}_j & \text{si } \frac{x}{t} < \sigma_1 \\ \mathbb{U}_j^* & \text{si } \sigma_1 < \frac{x}{t} < \sigma_2 \\ \mathbb{U}_{j+1}^* & \text{si } \sigma_2 < \frac{x}{t} < \sigma_3 \\ \mathbb{U}_{j+1} & \text{si } \sigma_3 < \frac{x}{t} \end{cases}$$

En observant la configuration du problème du Riemann qui nous concerne ici, on remarque quatre états distincts où le flux prend des valeurs différentes selon les trois discontinuités.

En superposant les contributions de chacune des trois ondes, on peut donc reconstruire le flux numérique.

$$\mathcal{G}_{j+1/2} = \frac{1}{2}[F(\mathbb{U}_j) + F(\mathbb{U}_{j+1}) - |\sigma_1|(\mathbb{U}_j^* - \mathbb{U}_j) - |\sigma_2|(\mathbb{U}_{j+1}^* - \mathbb{U}_j^*) - |\sigma_3|(\mathbb{U}_{j+1} - \mathbb{U}_{j+1}^*)]$$

En plus, du résultat issu des invariants de Riemann, on peut établir la définition des états étoiles grâce aux relations de Rankine-Hugoniot :

$$\left\{ \begin{array}{lcll} f(\mathbf{u}_{j+1}) - f(\mathbf{u}_{j+1}^*) & = & \sigma_3(\mathbf{u}_{j+1} - \mathbf{u}_{j+1}^*) & \text{avec} \quad \sigma_3 = u_{j+1} + a_{j+1/2}\tau_{j+1} \\ f(\mathbf{u}_j^*) - f(\mathbf{u}_j) & = & \sigma_1(\mathbf{u}_j^* - \mathbf{u}_j) & \text{avec} \quad \sigma_1 = u_j - a_{j+1/2}\tau_j \\ f(\mathbf{u}_{j+1}^*) - f(\mathbf{u}_j^*) & = & \sigma_2(\mathbf{u}_{j+1}^* - \mathbf{u}_j^*) & \text{avec} \quad \sigma_2 = u^* \end{array} \right.$$

Les états étoiles sont donc définis de la manière suivante :

$$\begin{array}{ll} \pi_j = p(\rho_j, E_j - \frac{u_j^2}{2}) & \\ \sigma_1 = u_j - a_{j+1/2}\tau_j & \\ \sigma_2 = u^* = \frac{1}{2}(u_j + u_{j+1}) + \frac{1}{2a}(\pi_j - \pi_{j+1}) & \\ \sigma_3 = u_{j+1} + a_{j+1/2}\tau_{j+1} & \\ \pi^* = \frac{1}{2}(\pi_j + \pi_{j+1}) - \frac{a_{j+1/2}}{2}(u_{j+1} - u_j) & \\ \tau_j^* = \tau_j + \frac{u^* - u_j}{a_{j+1/2}} & \tau_{j+1}^* = \tau_{j+1} - \frac{u^* - u_{j+1}}{a_{j+1/2}} \\ \rho_j^* = \frac{1}{\tau_j^*} & \rho_{j+1}^* = \frac{1}{\tau_{j+1}^*} \\ u_j^* = u^* & u_{j+1}^* = u^* \\ E_j^* = E_j + \frac{\pi_j u_j - \pi^* u^*}{a_{j+1/2}} & E_{j+1}^* = E_{j+1} - \frac{\pi_{j+1} u_{j+1} - \pi^* u^*}{a_{j+1/2}} \end{array}$$

Afin d'assurer théoriquement la convergence des solutions du système (7) vers les solutions du système (2), il est montré dans la littérature que la constante a, qui a pour vocation de remplacer ρc, doit être choisie strictement plus grande que ρc (condition de Whitham). Numériquement, nous avons fait le choix d'un a localisé à chaque interface et valant :

$$a_{j+1/2} = 1.01 \times \max(\rho_j c_j, \rho_{j+1} c_{j+1})$$

3.6 Calcul du pas de temps à chaque itération

Le choix du pas de temps Δt doit respecter une condition CFL nécessaire pour assurer la stabilité de l'algorithme. L'idée est de recoller les solutions aux problèmes de Riemann posés aux interfaces. Toutefois, la solution locale du problème de Riemann à $x_{j+1/2}$, $\mathbb{U}(\frac{x - x_{j+1/2}}{t - t^n}; \mathbb{U}_j^n, \mathbb{U}_{j+1}^n)$, peut être recollée à la solution du problème de Riemann voisin à condition d'éviter toute interaction des ondes. Posons σ^n la vitesse maximale de propagation des ondes issues de l'ensemble des interfaces du domaine au temps t^n :

$$\sigma^n = \max_{j=\{0,\ldots,N\}} (|u_j^n - a_{j+1/2}^n \tau_j^n|, |u_{j+1}^n + a_{j+1/2}^n \tau_{j+1}^n|)$$

En se basant sur la figure 4, dans le cas extrême où l'on prend la vitesse de l'onde allant de gauche à droite issue de $x_{j-1/2}$ égale à σ^n et la vitesse de l'onde allant de droite à gauche issue de $x_{j+1/2}$ égale à $-\sigma^n$, on obtient :

- $x - x_{j-1/2} = \sigma^n (t - t^n)$
- $x - x_{j+1/2} = -\sigma^n (t - t^n)$

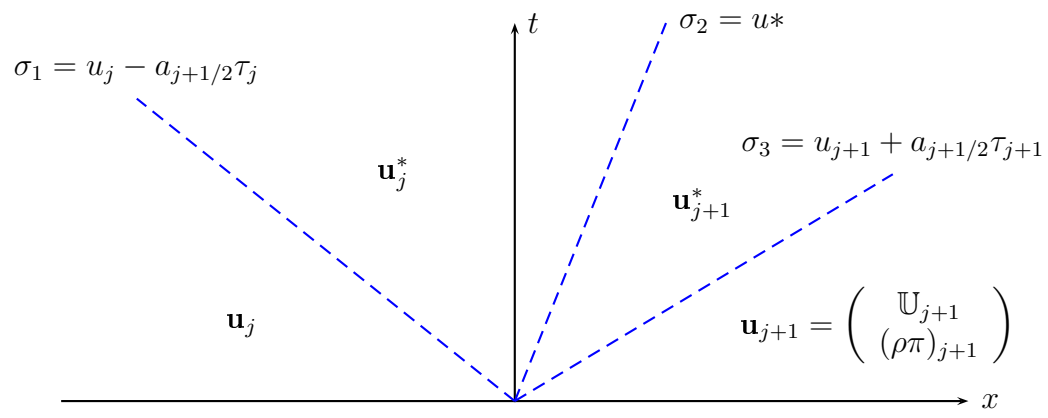


FIG. 3 – Structure de la solution à un problème de Riemann pour le système relaxé 1D

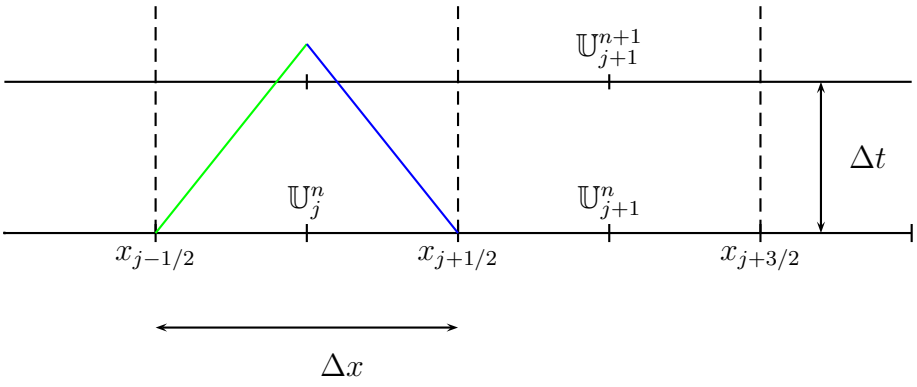


FIG. 4 – Choix du pas de temps en 1D

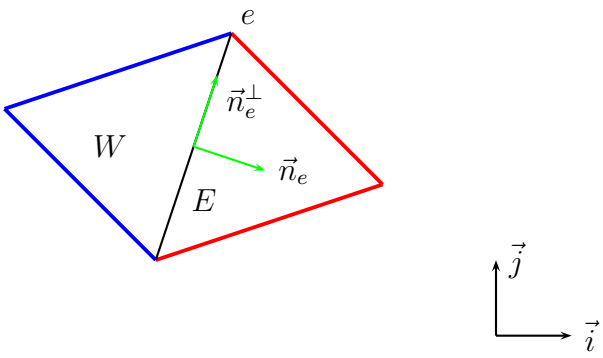


FIG. 5 – Configuration d'une arête

En ôtant à la première ligne la seconde, on constate l'égalité suivante :

$$\Delta x = 2\sigma^n(t - t^n)$$

On en déduit que le pas de temps doit être pris strictement positif tout en respectant l'inégalité suivante :

$$\Delta t \leq \frac{1}{2}\frac{\Delta x}{\sigma^n}$$

4 Discrétisation par une méthode de volumes finis dans le cas 2D

4.1 Présentation du problème

4.1.1 Convention

Soit Ω un ouvert borné connexe de $\mathbb{R}^2$. Considérons $\mathcal{T}_h = \bigcup_{k=1}^{\texttt{NbCells}} T_k$ un maillage triangulaire de Ω c'est-à-dire une famille de triangles T_k qui respectent les propriétés suivantes :

- l'intersection de deux triangles distincts est vide ou réduite à une arête ou à un sommet ;
- les triangles ne sont pas dégénérés (*ie* l'aire de chaque triangle est non nulle).

Soient e une arête et W, E les cellules situées de part et d'autre de e et appartenant à $\mathcal{T}_h$. Par convention, le sens de la normale $\vec{n}_e$ à l'arête e est toujours de W vers E (voir figure 5). Les vecteurs $\vec{n}_e = (n_x, n_y)$ et $\vec{n}_e^{\perp} = (n_x^{\perp}, n_y^{\perp})$ sont exprimés dans la base $\{\vec{i}, \vec{j}\}$. $\vec{n}_{e,W}$ est la normale à e sortante de W. Ainsi, le produit scalaire $\langle \vec{n}_{e,W}, \vec{n}_e \rangle$ vaut 1 alors que $\langle \vec{n}_{e,E}, \vec{n}_e \rangle$ vaut -1 car $\vec{n}_{e,E} = -\vec{n}_{e,W}$. $|e|$ est la longueur de e et $|W|$ l'aire de W.

4.1.2 Inconnues du problème

Pour mettre en œuvre la méthode des volumes finis, en plus de discrétiser le domaine spatial, nous discrétisons le temps t en n instants espacés de Δt tel que $t^n = n \times \Delta t$.

Notre vecteur d'inconnues $\mathbb{U}_K^n$ pour la cellule K au temps n est défini pour que $\mathbb{U}_K^n \approx \frac{1}{|K|}\int_K \mathbb{U}(\mathbf{x}, t^n)d\mathbf{x}$. Autrement dit $\mathbb{U}_K^n$ est une approximation de la valeur moyenne de la solution exacte sur le volume de contrôle K à l'instant t^n.

4.1.3 Condition initiale

On suppose connue la donnée initiale $\mathbb{U}(\mathbf{x}, 0)$ du problème. On part donc de la condition initiale suivante :

$$\forall K, \quad \mathbb{U}_K^0 = \frac{1}{|K|}\int_K \mathbb{U}(\mathbf{x}, 0)d\mathbf{x}$$

L'enjeu de la méthode réside maintenant dans le fait d'établir une relation de récurrence ($\mathbb{U}_K^{n+1} = \phi(\mathbb{U}_K^n)$) pour connaître l'évolution de $\mathbb{U}$ dans le temps.

4.2 Schéma explicite

4.2.1 Formule de mise à jour

On réalise une double intégration du système (1) en temps entre les instants t^n et t^{n+1} et en espace sur une cellule K :

$$\int_{t^n}^{t^{n+1}} \int_K \partial_t \mathbb{U}(\mathbf{x}, t) + \text{div} \begin{pmatrix} F(\mathbb{U}(\mathbf{x}, t)) \\ G(\mathbb{U}(\mathbf{x}, t)) \end{pmatrix} d\mathbf{x}dt = 0$$

En utilisant le théorème de Green-Ostrogradski, on obtient :

$$\int_K \mathbb{U}(\mathbf{x}, t^{n+1})d\mathbf{x} - \int_K \mathbb{U}(\mathbf{x}, t^n)d\mathbf{x} + \int_{t^n}^{t^{n+1}} \int_{\partial K} \begin{pmatrix} F(\mathbb{U}(\mathbf{x}, t)) \\ G(\mathbb{U}(\mathbf{x}, t)) \end{pmatrix} .\vec{n}_{\partial K} d\Gamma dt = 0$$

avec $\vec{n}_{\partial K}$, vecteur normé et normal à la frontière de K, dirigé vers l'extérieur. Notre cellule K étant ici un triangle, on peut réécrire cette égalité de la manière suivante :

$$\int_K \mathbb{U}(\mathbf{x}, t^{n+1})d\mathbf{x} - \int_K \mathbb{U}(\mathbf{x}, t^n)d\mathbf{x} + \sum_{e \subset \partial K} \int_{t^n}^{t^{n+1}} \int_e \begin{pmatrix} F(\mathbb{U}(\mathbf{x}, t)) \\ G(\mathbb{U}(\mathbf{x}, t)) \end{pmatrix} .\vec{n}_{e,K} d\Gamma dt = 0$$

avec $\vec{n}_{e,K}$, vecteur normé et normal à l'arête de K, dirigé vers l'extérieur. Considérons les approximations suivantes :

$$\begin{cases} \displaystyle\int_K \mathbb{U}(\mathbf{x}, t^n)d\mathbf{x} & \approx & |K|\mathbb{U}_K^n \\ \displaystyle\int_K \mathbb{U}(\mathbf{x}, t^{n+1})d\mathbf{x} & \approx & |K|\mathbb{U}_K^{n+1} \\ \displaystyle\int_{t^n}^{t^{n+1}} \int_e \begin{pmatrix} F(\mathbb{U}(\mathbf{x}, t)) \\ G(\mathbb{U}(\mathbf{x}, t)) \end{pmatrix} .\vec{n}_{e,K} d\Gamma dt & \approx & \Delta t |e| \mathcal{G}_e^n \langle \vec{n}_e, \vec{n}_{e,K} \rangle \end{cases}$$

$\mathcal{G}_e$ désigne ici le flux numérique calculé selon la normale $\vec{n}_e$ de sorte que $\langle \vec{n}_e, \vec{n}_{e,K} \rangle$ assure une orientation correcte du flux.
On en déduit la formule générale de mise à jour de $\mathbb{U}_K^n$ pour chaque cellule K en fonction des flux numériques $\mathcal{G}_e^n$ calculés à chaque arête e de K :

$$\mathbb{U}_K^{n+1} = \mathbb{U}_K^n - \frac{\Delta t}{|K|} \sum_{e \subset \partial K} |e| \mathcal{G}_e^n \langle \vec{n}_e, \vec{n}_{e,K} \rangle$$

Il n'y a que le flux numérique $\mathcal{G}_e$ qui change d'une méthode à l'autre (Godunov exacte, Roe, relaxation parmi d'autres).

4.2.2 Changement de base

Considérons les vecteurs $\vec{n}_e$ et $\vec{n}_e^{\perp}$:

$$\vec{n}_e = \begin{pmatrix} n_x \\ n_y \end{pmatrix} \quad \text{et} \quad \vec{n}_e^{\perp} = \begin{pmatrix} n_x^{\perp} \\ n_y^{\perp} \end{pmatrix} = \begin{pmatrix} -n_y \\ n_x \end{pmatrix}$$

En exprimant $\vec{\mathbf{v}} = \bar{u}\,\vec{n}_e + \bar{v}\,\vec{n}_e^{\perp}$ dans la base locale $\{\vec{n}_e, \vec{n}_e^{\perp}\}$, on obtient :

$$\begin{cases} u = \bar{u}n_x + \bar{v}n_x^{\perp} \\ v = \bar{u}n_y + \bar{v}n_y^{\perp} \end{cases}$$

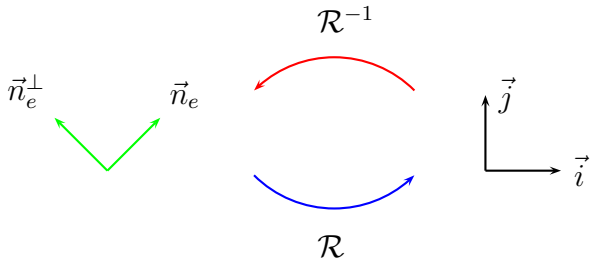
Ce qui donne sous forme matricielle, l'égalité suivante :

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} n_x & n_x^{\perp} \\ n_y & n_y^{\perp} \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix}$$

Notons $\mathcal{R}(\vec{n}_e)$ la rotation qui passe de la base $\{\vec{n}_e, \vec{n}_e^{\perp}\}$ à la base $\{\vec{i}, \vec{j}\}$. Soit $\bar{\mathbb{U}}_K^n = (\rho_K^n, (\rho\bar{u})_K^n, (\rho\bar{v})_K^n, (\rho E)_K^n)$. L'image de $\bar{\mathbb{U}}_K^n$ par la rotation $\mathcal{R}(\vec{n}_e)$ donne $\mathbb{U}_K^n$. On a $\mathbb{U}_K^n = \mathcal{R}(\vec{n}_e)\bar{\mathbb{U}}_K^n$ ou $\bar{\mathbb{U}}_K^n = \mathcal{R}^{-1}(\vec{n}_e)\mathbb{U}_K^n$ avec

$$\mathcal{R}(\vec{n}_e) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & n_x & n_x^{\perp} & 0 \\ 0 & n_y & n_y^{\perp} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

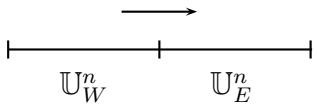
$$\mathcal{R}^{-1}(\vec{n}_e) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & n_x & n_y & 0 \\ 0 & n_x^{\perp} & n_y^{\perp} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



4.2.3 Passage à un problème quasi-1D

On peut donc définir $\mathcal{G}_e^n$ en considérant que $\mathcal{G}_e^n = \mathcal{R}(\vec{n}_e)\bar{\mathcal{G}}_e^n$ avec $\bar{\mathcal{G}}_e^n$ le flux numérique correspondant au modèle quasi-1D suivant :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial(\rho\bar{u})}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho\bar{u})}{\partial t} + \dfrac{\partial(\rho\bar{u}^2 + p)}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho\bar{v})}{\partial t} + \dfrac{\partial(\rho\bar{u}\bar{v})}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho E)}{\partial t} + \dfrac{\partial(\rho E\bar{u} + p\bar{u})}{\partial \bar{x}} = 0 \end{cases} \tag{8}$$



Sous forme vectorielle, le système (8) s'écrit alors :

$$\partial_t \bar{\mathbb{U}} + \partial_{\bar{x}} F(\bar{\mathbb{U}}) = 0$$

On réalise alors la relaxation sur ce cas 1D pour déterminer $\bar{\mathcal{G}}_e^n$. Ensuite, on applique la rotation $\mathcal{R}(\vec{n}_e)$ pour retomber sur $\mathcal{G}_e^n$. Sans surprise, le système quasi-1D relaxé s'écrit de la manière suivante :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial(\rho\bar{u})}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho\bar{u})}{\partial t} + \dfrac{\partial(\rho\bar{u}^2 + \pi)}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho\bar{v})}{\partial t} + \dfrac{\partial(\rho\bar{u}\bar{v})}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho E)}{\partial t} + \dfrac{\partial(\rho E\bar{u} + \pi\bar{u})}{\partial \bar{x}} = 0 \\ \dfrac{\partial(\rho\pi)}{\partial t} + \dfrac{\partial(\rho\pi\bar{u} + a^2\bar{u})}{\partial \bar{x}} = \dfrac{\rho}{\delta}(p - \pi) \end{cases} \tag{9}$$

Le système (9) écrit sous forme vectorielle devient :

$$\partial_t \mathbf{u} + \partial_x f(\mathbf{u}) = \frac{1}{\delta}\mathcal{Z}(\mathbf{u})$$

avec $\mathbf{u} = (\bar{\mathbb{U}}, \rho\pi)^t = (\rho, \rho\bar{u}, \rho\bar{v}, \rho E, \rho\pi)^t$, $f(\mathbf{u}) = (\rho\bar{u}, \rho\bar{u}^2 + \pi, \rho\bar{u}\bar{v}, \rho E\bar{u} + \pi\bar{u}, \rho\pi\bar{u} + a^2\bar{u})^t$ et $\mathcal{Z}(\mathbf{u}) = (0, 0, 0, \rho(p - \pi))^t$. Comme précédemment l'inconnue π est traitée séparément des autres inconnues contenues dans $\bar{\mathbb{U}}$.

4.2.4 Constante de relaxation associée à chaque arête

Comme pour le cas 1D, la constante de relaxation qui a vocation à remplacer ρc est choisie localement à chaque arête e de sorte que :

$$a_e = 1.001 \times \max(\rho_W c_W, \rho_E c_E)$$

4.2.5 Choix du pas de temps

$$\Delta t \leq \frac{1}{4} \max_{K \in \mathcal{T}_h} \theta_K$$

avec

$$\begin{cases} \theta_K = \dfrac{P_K}{|K|} \max\limits_{e \subset \partial K} \mathcal{A}^e \\ \mathcal{A}^e = \max(|u_W - a_e \tau_W|, |u_E + a_e \tau_E|) \\ P_K = \displaystyle\sum_{e \subset \partial K} |e| \end{cases}$$

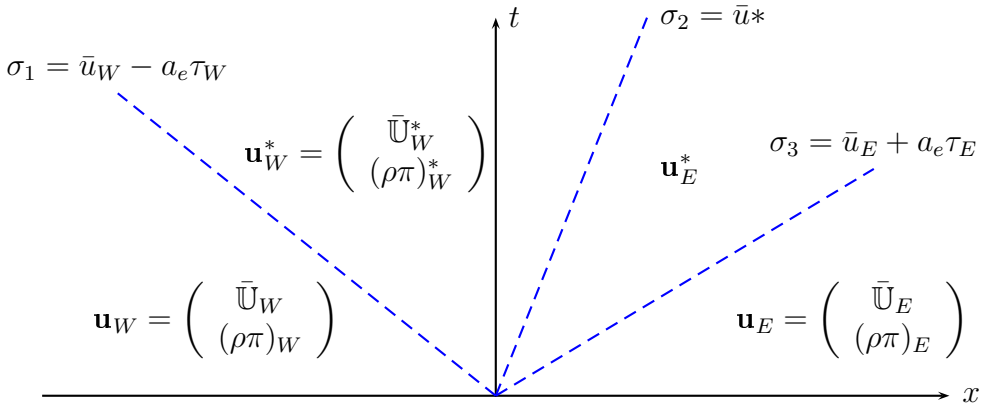


FIG. 6 – Structure de la solution à un problème de Riemann pour le système relaxé quasi-1D

4.2.6 Calcul du flux

Comme en 1D, on retrouve le flux en superposant les contributions des trois ondes :

$$\begin{aligned} \bar{\mathcal{G}}_e(\bar{\mathbb{U}}_W, \bar{\mathbb{U}}_E) &= \frac{1}{2}\Big(F(\bar{\mathbb{U}}_W) + F(\bar{\mathbb{U}}_E)\Big) - \frac{|\sigma_1|}{2}(\bar{\mathbb{U}}_W^* - \bar{\mathbb{U}}_W) \\ &\quad - \frac{|\sigma_2|}{2}(\bar{\mathbb{U}}_E^* - \bar{\mathbb{U}}_W^*) - \frac{|\sigma_3|}{2}(\bar{\mathbb{U}}_E - \bar{\mathbb{U}}_E^*) \end{aligned}$$

4.2.7 Définition des états étoilés

Le problème de Riemann associé au sytème d'équations d'Euler relaxé nous amène à la configuration suivante :
Les trois discontinuités de contact nous permettent d'utiliser les relations de Rankine-Hugoniot :

$$\left\{ \begin{array}{lcl} f(\mathbf{u}_E) - f(\mathbf{u}_E^*) &=& \sigma_3(\mathbf{u}_E - \mathbf{u}_E^*) \\ f(\mathbf{u}_W^*) - f(\mathbf{u}_W) &=& \sigma_1(\mathbf{u}_W^* - \mathbf{u}_W) \\ f(\mathbf{u}_E^*) - f(\mathbf{u}_W^*) &=& \sigma_2(\mathbf{u}_E^* - \mathbf{u}_W^*) \end{array} \right.$$

De plus, comme pour le système d'équations d'Euler non relaxé, on sait montrer en manipulant les invariants de Riemann que :

$$\left\{ \begin{array}{l} \bar{u}_W^* = \bar{u}_E^* = \bar{u}^* \\ \pi_W^* = \pi_E^* = \pi^* \end{array} \right.$$

Tout est donc réuni pour déterminer facilement les états étoilés :

$$
\begin{array}{ll}
\bar{u}^* = \frac{1}{2}(\bar{u}_W + \bar{u}_E) + \frac{1}{2a_e}(\pi_W - \pi_E) & \\
\pi^* = \frac{1}{2}(\pi_W + \pi_E) - \frac{a_e}{2}(\bar{u}_E - \bar{u}_W) & \\
\sigma_1 = \bar{u}_W - a_e\tau_W & \\
\sigma_2 = \bar{u}^* = \frac{1}{2}(\bar{u}_W + \bar{u}_E) + \frac{1}{2a}(\pi_W - \pi_E) & \\
\sigma_3 = \bar{u}_E + a_e\tau_E & \\
\tau_W^* = \tau_W + \frac{\bar{u}^* - \bar{u}_W}{a_e} & \tau_E^* = \tau_E - \frac{\bar{u}^* - \bar{u}_E}{a_e} \\
\rho_W^* = \frac{1}{\tau_W^*} & \rho_E^* = \frac{1}{\tau_E^*} \\
\bar{v}_W^* = \bar{v}_W & \bar{v}_E^* = \bar{v}_E \\
E_W^* = E_W + \frac{\pi_W \bar{u}_W - \pi^* \bar{u}^*}{a_e} & E_E^* = E_E - \frac{\pi_E \bar{u}_E - \pi^* \bar{u}^*}{a_e}
\end{array}
$$

4.3 Schéma implicite

4.3.1 Implicitation du flux par linéarisation

L'idée de départ de la méthode implicite est de calculer les $\mathbb{U}_K^{n+1}$ en utilisant des termes de flux qui sont eux-mêmes au temps t^{n+1}, ce qui donne :

$$\mathbb{U}_K^{n+1} = \mathbb{U}_K^n - \frac{\Delta t}{|K|} \sum_{e \subset \partial K} |e| \mathcal{G}_e^{n+1} \langle \vec{n}_e, \vec{n}_{e,K} \rangle$$

Pour y arriver, on aimerait utiliser la formule de Taylor pour linéariser le flux implicite $\mathcal{G}_e^{n+1}$ ou plus exactement $\bar{\mathcal{G}}_e^{n+1}$. Toutefois, ce flux, tel qu'il est défini, n'est pas différentiable notamment à cause des valeurs absolues. On introduit donc un flux numérique qu'on appelle $\bar{\mathcal{G}}_{e,d}^{n+1}$ différentiable proche de $\bar{\mathcal{G}}_e^{n+1}$. Soient $b = \max_{e \in \mathcal{T}_h}(|\sigma_1|, |\sigma_2|, |\sigma_3|)$ et $B = bI_4$. Posons $\Delta\bar{\mathbb{U}}_K^{n+1} = \bar{\mathbb{U}}_K^{n+1} - \bar{\mathbb{U}}_K^n$ et $\mathcal{R} = \mathcal{R}(\vec{n}_e)$. Replaçons nous au niveau d'une arête (voir figure 5). On a établi précédemment que :

$$\mathcal{G}_e^{n+1} = \mathcal{R}\bar{\mathcal{G}}_e^{n+1}$$

De la même manière, notre flux de substitution donne :

$$\mathcal{G}_{e,d}^{n+1} = \mathcal{R}\bar{\mathcal{G}}_{e,d}^{n+1}$$

avec

$$\bar{\mathcal{G}}_{e,d}^{n+1} = \frac{1}{2}\Big(F(\bar{\mathbb{U}}_W^{n+1}) + F(\bar{\mathbb{U}}_E^{n+1})\Big) - \frac{1}{2}B(\bar{\mathbb{U}}_E^{n+1} - \bar{\mathbb{U}}_W^{n+1})$$

En utilisant la formule de Taylor, on peut considérer que :

$$
\begin{array}{rcl}
\bar{\mathcal{G}}_{e,d}^{n+1} & \approx & \frac{1}{2}\left(F(\bar{\mathbb{U}}_W^n) + F(\bar{\mathbb{U}}_E^n)\right) + \frac{1}{2}\nabla F(\bar{\mathbb{U}}_W^n)\Delta\bar{\mathbb{U}}_W^{n+1} \\
 & & +\frac{1}{2}\nabla F(\bar{\mathbb{U}}_E^n)\Delta\bar{\mathbb{U}}_E^{n+1} - \frac{1}{2}B(\bar{\mathbb{U}}_E^{n+1} - \bar{\mathbb{U}}_W^{n+1})
\end{array}
$$

On ajoute et on retranche de manière artificielle le terme $\frac{1}{2}B(\bar{\mathbb{U}}_E^n - \bar{\mathbb{U}}_W^n)$ au membre de gauche ce qui nous permet d'écrire :

$$
\begin{array}{rcl}
\bar{\mathcal{G}}_{e,d}^{n+1} & \approx & \bar{\mathcal{G}}_{e,d}^n + \frac{1}{2}\left(\nabla F(\bar{\mathbb{U}}_W^n)\Delta\bar{\mathbb{U}}_W^{n+1} + \nabla F(\bar{\mathbb{U}}_E^n)\Delta\bar{\mathbb{U}}_E^{n+1}\right) \\
 & & -\frac{1}{2}B(\bar{\mathbb{U}}_E^{n+1} - \bar{\mathbb{U}}_W^{n+1}) + \frac{1}{2}B(\bar{\mathbb{U}}_E^n - \bar{\mathbb{U}}_W^n) \\
\bar{\mathcal{G}}_{e,d}^{n+1} & \approx & \bar{\mathcal{G}}_{e,d}^n + \frac{1}{2}\left[\nabla F(\bar{\mathbb{U}}_W^n) + B\right]\Delta\bar{\mathbb{U}}_W^{n+1} \\
 & & +\frac{1}{2}\left[\nabla F(\bar{\mathbb{U}}_E^n) - B\right]\Delta\bar{\mathbb{U}}_E^{n+1}
\end{array}
$$

En multipliant par $\mathcal{R}$ l'expression précédente on obtient :

$$\begin{array}{rcl} \mathcal{G}_{e,d}^{n+1} & \approx & \mathcal{G}_{e,d}^{n} + \frac{1}{2}\mathcal{R}\left[\nabla F(\bar{\mathbb{U}}_W^n) + B\right]\Delta\bar{\mathbb{U}}_W^{n+1} \\ & & +\frac{1}{2}\mathcal{R}\left[\nabla F(\bar{\mathbb{U}}_E^n) - B\right]\Delta\bar{\mathbb{U}}_E^{n+1} \end{array}$$

En utilisant le fait que $\Delta\bar{\mathbb{U}}_W^{n+1} = \mathcal{R}^{-1}\Delta\mathbb{U}_W^{n+1}$, on peut établir que :

$$\begin{array}{rcl} \mathcal{G}_{e,d}^{n+1} & \approx & \mathcal{G}_{e,d}^{n} + \frac{1}{2}\mathcal{R}\left[\nabla F(\bar{\mathbb{U}}_W^n) + B\right]\mathcal{R}^{-1}\Delta\mathbb{U}_W^{n+1} \\ & & +\frac{1}{2}\mathcal{R}\left[\nabla F(\bar{\mathbb{U}}_E^n) - B\right]\mathcal{R}^{-1}\Delta\mathbb{U}_E^{n+1} \end{array}$$

En posant :

$$\left\{ \begin{array}{rcl} M_W^e & = & \frac{1}{2}\mathcal{R}\left[\nabla F(\bar{\mathbb{U}}_W^n) + B\right]\mathcal{R}^{-1} \\ M_E^e & = & \frac{1}{2}\mathcal{R}\left[\nabla F(\bar{\mathbb{U}}_E^n) - B\right]\mathcal{R}^{-1} \end{array} \right.$$

on aboutit donc à :

$$\mathbb{U}_K^{n+1} = \mathbb{U}_K^n - \frac{\Delta t}{|K|}\sum_{e\subset\partial K}|e|(\mathcal{G}_{e,d}^n + M_W^e\Delta\mathbb{U}_W^{n+1} + M_E^e\Delta\mathbb{U}_E^{n+1})\langle\vec{n}_e, \vec{n}_{e,K}\rangle$$

Pour gagner en précision, nous préférons remplacer le flux de substitution $\mathcal{G}_{e,d}^n$ par le vrai flux $\mathcal{G}_e^n$. La formule de mise à jour devient alors :

$$\mathbb{U}_K^{n+1} = \mathbb{U}_K^n - \frac{\Delta t}{|K|}\sum_{e\subset\partial K}|e|(\mathcal{G}_{e}^n + M_W^e\Delta\mathbb{U}_W^{n+1} + M_E^e\Delta\mathbb{U}_E^{n+1})\langle\vec{n}_e, \vec{n}_{e,K}\rangle$$

Dès lors, nous pouvons deviner la structure de la matrice et les blocs qui la composent.

4.3.2 Gradient de F

cas du gaz réel

Nous ne connaissons pas la loi d'état régissant la pression p mais nous savons qu'elle dépend de ρ et de e :

$$p = p(\rho, e) \text{ avec } e = E - \frac{u^2+v^2}{2}$$

Ainsi,

$$\begin{array}{rcl} \frac{\partial p}{\partial \rho} & = & \frac{\partial p}{\partial \rho}_{|e} + \frac{\partial e}{\partial \rho}\frac{\partial p}{\partial e}_{|\rho} \\ \frac{\partial p}{\partial \rho} & = & \frac{\partial p}{\partial \rho}_{|e} + \left(\frac{-E}{\rho} + \frac{u^2}{\rho} + \frac{v^2}{\rho}\right)\frac{\partial p}{\partial e}_{|\rho} \end{array}$$

Or

$$c^2 = \frac{\partial p}{\partial \rho}_{|e} + \frac{p}{\rho^2}\frac{\partial p}{\partial e}_{|\rho}$$

Donc

$$\frac{\partial p}{\partial \rho} = c^2 + \frac{1}{\rho}\frac{\partial p}{\partial e}_{|\rho}\left(u^2 + v^2 - E - \frac{p}{\rho}\right)$$

Par ailleurs,

$$\begin{array}{rcl} \frac{\partial p}{\partial (\rho u)} & = & \frac{\partial e}{\partial (\rho u)}\frac{\partial p}{\partial e}_{|\rho} \\ \frac{\partial p}{\partial (\rho u)} & = & -\frac{u}{\rho}\frac{\partial p}{\partial e}_{|\rho} \end{array}$$

Et de même,

$$\begin{aligned}\frac{\partial p}{\partial(\rho v)} &= \frac{\partial e}{\partial(\rho v)}\frac{\partial p}{\partial e}_{|\rho}\\ \frac{\partial p}{\partial(\rho v)} &= -\frac{v}{\rho}\frac{\partial p}{\partial e}_{|\rho}\end{aligned}$$

Enfin,

$$\begin{aligned}\frac{\partial p}{\partial(\rho E)} &= \frac{\partial e}{\partial(\rho E)}\frac{\partial p}{\partial e}_{|\rho}\\ \frac{\partial p}{\partial(\rho E)} &= \frac{1}{\rho}\frac{\partial p}{\partial e}_{|\rho}\end{aligned}$$

Une fois les dérivées partielles de p établies, nous pouvons définir ∇F.
$\nabla F : \mathbb{R}^4 \longrightarrow \mathcal{M}_{4,4}(\mathbb{R})$

$$\nabla F(\mathbb{U}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -u^2+K & u(2-k) & -vk & k \\ -uv & v & u & 0 \\ u(K-H) & H-u^2k & -uvk & u(1+k) \end{pmatrix}$$

avec

$$\begin{aligned}H &= E + \frac{p}{\rho}\\ k &= \frac{1}{\rho}\frac{\partial p}{\partial e}_{|\rho}\\ K &= c^2 + k(u^2+v^2-H)\end{aligned}$$

cas du gaz parfait

Soit $\mathbb{U} \in \mathbb{R}^4$.

$$\mathbb{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}$$

Soit $F : \mathbb{R}^4 \longrightarrow \mathbb{R}^4$,

$$F(\mathbb{U}) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho Eu + pu \end{pmatrix} = \begin{pmatrix} \rho u \\ \frac{(\rho u)^2}{\rho} + (\gamma-1)(\rho E - \frac{(\rho u)^2}{2\rho} - \frac{(\rho v)^2}{2\rho}) \\ \frac{(\rho u)(\rho v)}{\rho} \\ \frac{(\rho E)(\rho u)}{\rho} + \frac{\rho u}{\rho}(\gamma-1)(\rho E - \frac{(\rho u)^2}{2\rho} - \frac{(\rho v)^2}{2\rho}) \end{pmatrix}$$

Par conséquent, $\nabla F : \mathbb{R}^4 \longrightarrow \mathcal{M}_{4,4}(\mathbb{R})$

$$\nabla F(\mathbb{U}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{(\gamma-3)u^2+(\gamma-1)v^2}{2} & (3-\gamma)u & (1-\gamma)v & \gamma-1 \\ -uv & v & u & 0 \\ -\gamma Eu + (\gamma-1)u(u^2+v^2) & \gamma E - \frac{(\gamma-1)(3u^2-v^2)}{2} & (1-\gamma)uv & \gamma u \end{pmatrix}$$

4.3.3 Assemblage de la matrice

Soient $\mathtt{i_W}$ et $\mathtt{i_E}$ les indices des cellules W et E. Nous savons que la matrice est de taille $(4 \times \mathtt{NbCells}) \times (4 \times \mathtt{NbCells})$. M_E^e et M_W^e sont des matrices de taille 4×4 tandis que I_4 désigne la matrice identité de taille 4×4. $\mathtt{A[i,j]}$ désigne l'emplacement du bloc matriciel de taille 4×4 tel que

$$\mathtt{A[i,j]} = \begin{pmatrix} \mathtt{A[4i,4j]} & \mathtt{A[4i,4j+1]} & \mathtt{A[4i,4j+2]} & \mathtt{A[4i,4j+3]} \\ \mathtt{A[4i+1,4j]} & \mathtt{A[4i+1,4j+1]} & \mathtt{A[4i+1,4j+2]} & \mathtt{A[4i+1,4j+3]} \\ \mathtt{A[4i+2,4j]} & \mathtt{A[4i+2,4j+1]} & \mathtt{A[4i+2,4j+2]} & \mathtt{A[4i+2,4j+3]} \\ \mathtt{A[4i+3,4j]} & \mathtt{A[4i+3,4j+1]} & \mathtt{A[4i+3,4j+2]} & \mathtt{A[4i+3,4j+3]} \end{pmatrix}$$

L'algorithme d'assemblage de la matrice se présente alors de la manière suivante :

```
Function A←AssembleMatrix()
        Loop for each edge e
                A[iW,iW] ← Δt/|W| |e| M_W^e + A[iW,iW]
                A[iW,iE] ← Δt/|W| |e| M_E^e + A[iW,iE]
                A[iE,iE] ← −Δt/|E| |e| M_E^e + A[iE,iE]
                A[iE,iW] ← −Δt/|E| |e| M_W^e + A[iE,iW]
        End loop for each edge e
        Loop for each cell K
                A[iK,iK]← I_4 + A[iK,iK]
        End loop for each cell K
End function
```

4.3.4 Construction du second membre

Soient $\mathtt{i_W}$ et $\mathtt{i_E}$ les indices des cellules W et E. Nous savons que le second membre est un vecteur de taille $(4 \times \mathtt{NbCells})$. $\mathcal{G}_e^n$ est un vecteur de taille 4. $\mathtt{b[i]}$ désigne l'emplacement du bloc vectoriel de taille 4 tel que

$$\mathtt{b[i]} = \begin{pmatrix} \mathtt{b[4i]} \\ \mathtt{b[4i+1]} \\ \mathtt{b[4i+2]} \\ \mathtt{b[4i+3]} \end{pmatrix}$$

L'algorithme d'assemblage du second membre du système linéaire se présente alors de la manière suivante :

```
Function b←BuildRightHandVector()
        Loop for each edge e
                b[iW] ← −Δt/|W| |e| G_e^n + b[iW]
                b[iE] ← Δt/|E| |e| G_e^n + b[iE]
        End loop for each edge e
End function
```

5 Initialisation

5.1 Maillage 2D

5.1.1 Création du maillage

Avant même de penser à lancer les calculs propres à l'algorithme, il est indispensable de générer un maillage et d'en récupérer les données. Pour cela, nous avons décidé d'utiliser le programme Triangle (voir [3] pour plus de renseignements). Nous allons montrer sur un exemple simple comment on peut générer un maillage d'un domaine de dimension 2 avec cet outil. Prenons le cas d'un carré dont les sommets sont indicés de 0 à 3 et ayant pour coordonnées respectivement (0.0,0.0) (0.0,1.0) (1.0,0.0) (1.0,1.0). Ces derniers délimitent la frontière du maillage. On peut décider d'imposer un sommet à l'intérieur de celui-ci comme le sommet 4 de coordonnées (0.5,0.5). Nous avons donc eu besoin de 5 sommets pour définir notre maillage. A chaque sommet il est possible d'attribuer (en plus de son propre indice de 0 à 3) un indice de région (on a utilisé 5 dans notre exemple) mais cela reste facultatif. Si on veut établir plusieurs marqueurs de région, le nombre de marqueurs est fixé à 1, 0 si on se contente d'un marqueur unique. Pour que Triangle puisse générer le maillage il faut aussi lui préciser la manière dont sont reliés les sommets. Ici, nous construisons 4 bords en reliant les quatres sommets du carré. A chaque bord il est demandé d'attribuer (en plus de son propre indice de 0 à 3) un marqueur de frontière. Si on veut établir plusieurs marqueurs de frontières, le nombre de marqueurs est fixé à 1, 0 si on se contente d'un marqueur unique. Par exemple, on peut prendre 1 pour le bord 0 reliant les sommets 0 et 1, 1 pour celui reliant les sommets 3 et 2, 2 pour celui reliant les sommets 1 et 3 et enfin 2 pour celui reliant les sommets 2 et 0. Enfin, il faut préciser à Triangle le nombre de cavités contenues dans le domaine que l'on veut discrétiser. Ce nombre vaut ici 0. Il est possible d'associer des attributs aux sommets mais nous n'utilisons pas cette option ici et fixons le nombre d'attribut à 0. Toutes ces informations sont alors utilisées dans le fichier que l'on nommera `square.poly` qui se présentera de la manière suivante :

```
5 2 0 1
0 0.0 0.0 5
1 0.0 1.0 5
2 1.0 0.0 5
3 1.0 1.0 5
4 0.5 0.5
4 1
0 0 1 1
1 1 3 2
2 3 2 1
3 2 0 2
0
```

Une fois le fichier créé, Triangle génère un maillage en tapant la ligne de commande suivante :

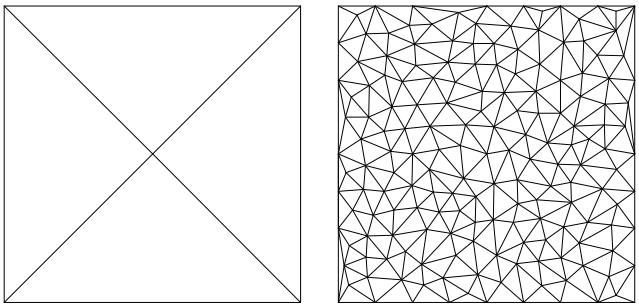
$./triangle -nez square.poly

Les commandes `-n` et `-e` obligent Triangle à générer les fichiers `.neigh` et `.edge` tandis que `-z` impose une numérotation des éléments en partant de 0 au lieu de 1 ce qui est préférable ici car nous avons fait le choix d'utiliser le langage C. Quatres fichiers (`.node`, `.ele`, `.edge`, `.neigh`) sont alors créés contenant les informations propres au maillage qui vient d'être constitué. Dans ce cas là, Triangle se contente de travailler avec les sommets

définis dans le fichier `.poly` et n'en génère pas de lui-même. Pour raffiner le maillage, on peut utiliser la commande `-a` suivie de la valeur de l'aire maximale désirée pour les mailles considérées comme par exemple :

```
$./triangle -neza0.005 square.poly
```

On obtient alors les maillages suivants avec les deux commandes données précédemment :



5.1.2 Présentation des fichiers générés par le mailleur

Nous avons vu que Triangle génère quatre fichiers pour la création d'un maillage 2D. Nous allons présenter le contenu de ces fichiers dans le cas particulier du maillage créé précédemment sans raffinement.

Fichier `square.node`

```
  5 2 0 1
0 0.0 0.0 5
1 0.0 1.0 5
2 1.0 0.0 5
3 1.0 1.0 5
4 0.5 0.5 0
```

Les informations contenues dans ce fichier sont récapitulées sur la figure 7.

Fichier `square.edge`

```
  8 1
0 1 0 1
1 0 4 0
2 4 1 0
3 4 2 0
4 2 3 1
5 3 4 0
6 0 2 2
7 3 1 2
```

Les informations contenues dans ce fichier sont récapitulées sur la figure 8.

Fichier `square.ele`

```
  4 3 0
0 1 0 4
1 4 2 3
2 2 4 0
```

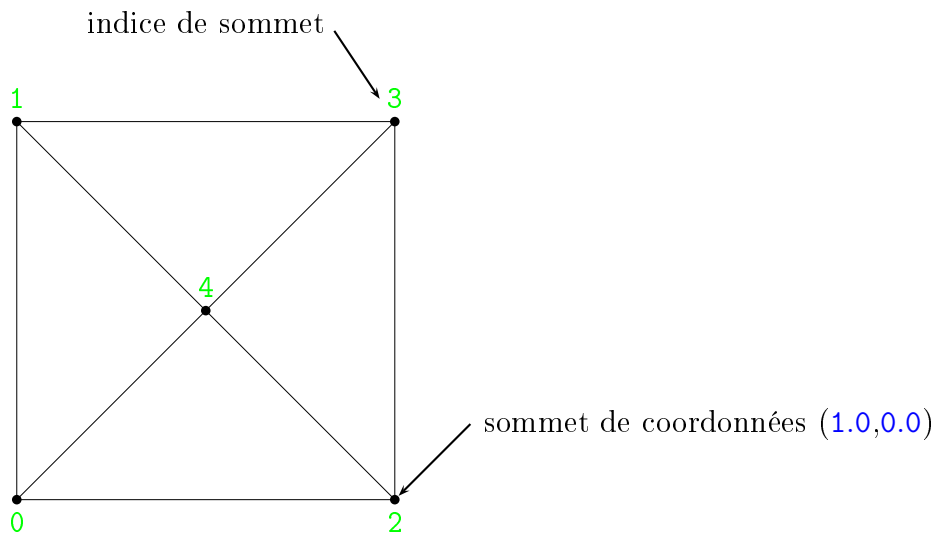


FIG. 7 – Informations contenues dans le fichier `square.node`

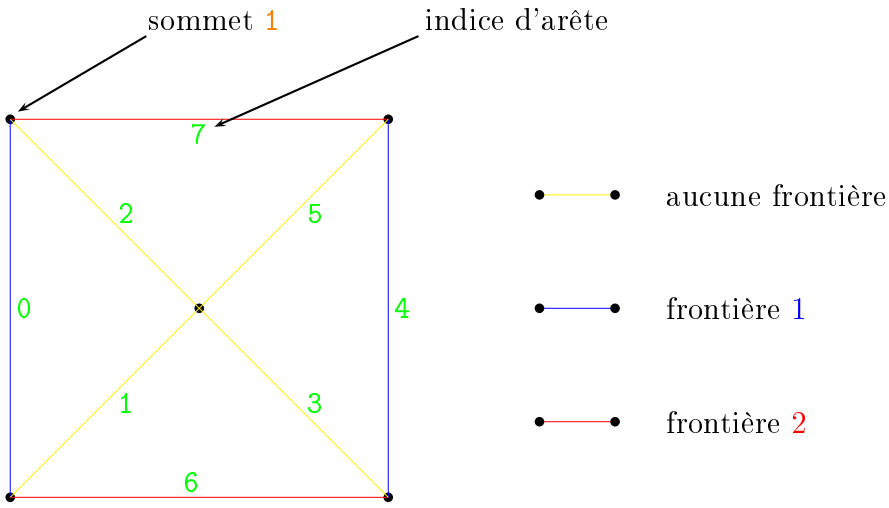


FIG. 8 – Informations contenues dans le fichier `square.edge`

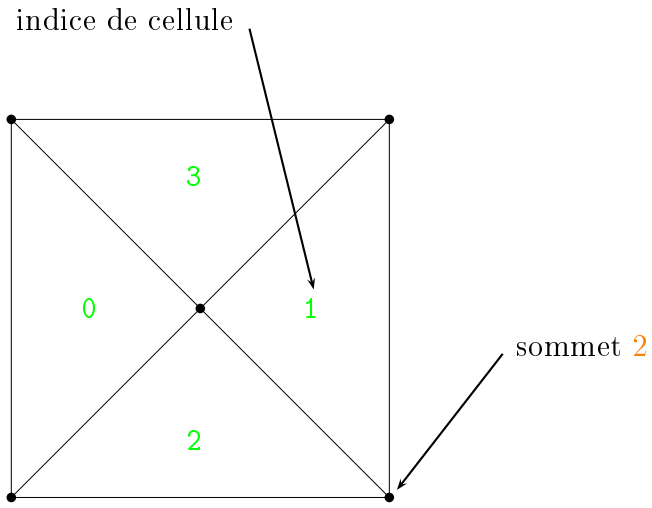


FIG. 9 – Informations contenues dans le fichier `square.ele`

```
3 4 3 1
```

Les informations contenues dans ce fichier sont récapitulées sur la figure 9.

Fichier `square.neigh`

```
  4 3
0 2 3 -1
1 -1 3 2
2 0 -1 1
3 -1 0 1
```

Les informations données ici sont redondantes. Elles permettent toutefois de connaître directement les indices des cellules voisines de chaque cellule numérotée de 0 à 3. Le chiffre -1 signifie que la cellule possède une arête appartenant à la frontière du domaine.

5.2 Initialisation des données

L'initialisation des données c'est-à-dire le calcul des valeurs du vecteur $\mathbb{U}_K^0$ pour chaque cellule K du maillage considéré au temps initial n'est pas assurée par le code mais par un script écrit en Python qui écrit ensuite ces valeurs dans un fichier de type `vtk` avec pour nom de fichier `data000.vtk`. Le code se contente alors de lire ce fichier `vtk` pour connaître les valeurs associées à chaque cellule au temps $t = 0$.

5.3 Fichier de configuration

Ce fichier appelé `config.tms` est lu par le code pour déterminer ce qu'il doit accomplir lors de son exécution. Parmi les informations présentes, on y trouve :

- la valeur du temps final ;
- le nombre de sauvegardes à effectuer avant d'arriver au temps final ;
- le nom des quatres fichiers générés par le mailleur Triangle ;
- le nom du solveur que l'on souhaite utiliser (relaxation implicite ou relaxation explicite) ;
- la valeur de la cfl ;

- préciser la nature du gaz (gaz parfait par exemple) et la valeur des constantes liées (γ, c_v pour l'exemple du gaz parfait) ;
- nombre de frontières maximales.

6 Programmation en C

6.1 Compilation

Le code étant relativement long (notamment à cause des tests unitaires) et amené à subir des modifications/ajouts, la gestion manuelle des makefiles peut très vite devenir difficile vu le nombre de fichiers à manipuler. Un moyen qui permet de contourner ce problème est d'utiliser un outil qui traite lui-même la compilation, l'utilisateur se contentant de lui donner uniquement le chemin des fichiers sources. Le générateur automatique de fichiers makefiles qu'on a utilisé ici s'appelle CMake. L'avantage est un gain de temps indéniable. Notons que cette solution exige, bien entendu, que CMake soit installé sur la machine où se fait la compilation.

6.1.1 Compilation du code

Supposons que le dossier contenant le projet TriMeshSolver a pour chemin :

$$/\text{home/user/project} - \text{TriMeshSolver}$$

Pour générer l'executable, il faut créer un dossier qui le contiendra :

$$\$ \ \text{mkdir} \ \ /\text{home/user/exe/build}$$

Ensuite on se met dans le dossier créé :

$$\$ \ \text{cd} \ \ /\text{home/user/exe/build}$$

A ce niveau, l'utilisateur peut utiliser deux modes de compilation :
- compiler l'executable principal et l'ensemble des tests unitaires présents dans le projet :
  $\$ \ \text{cmake} \ -\text{DCOMPIL\_TEST\_EXEC} = 1 \ /\text{home/user/project} - \text{TriMeshSolver/src}$
- compiler uniquement l'executable principal :
  $\$ \ \text{cmake} \ /\text{home/user/project} - \text{TriMeshSolver/src}$

Une fois le choix défini, l'executable est généré avec la commande suivante :

$$\$ \ \text{make}$$

Maintenant, on peut lancer le programme en précisant le chemin où se trouvent certaines données comme les fichiers propres au maillage. Une fois présent dans le dossier contenant l'executable, on tape donc par exemple :

$$\$ \ ./\text{TMS.exe} \ /\text{path\_data/DataSodShock}$$

6.1.2 Insertion d'un nouveau code dans le projet

Imaginons qu'on ait codé en plus des solveurs de relaxation explicite et implicite un autre solveur. Comment peut-on l'inclure dans le projet TriMeshSolver ? A l'intérieur du répertoire contenant le projet, se trouve le répertoire src :

/home/user/project – TriMeshSolver/src

Ce dernier contient plusieurs éléments notamment :

```
src /
    CMakeLists.txt
    common
    ...
    ...
    mesh
    solverRelax
    solverRelaxImp
    template
```

Introduire le nouveau solveur consiste à copier le dossier template toujours dans le même dossier et à lui donner le nom du solveur qui sera ici solverNew. Dans ce nouveau dossier, il suffit d'insérer les fichiers sources `.c` et `.h` qui contiennent les routines propres au nouveau solveur. Le contenu du dossier pourrait donner par exemple :

```
solverNew /
          CMakeLists.txt
          solverNew.c
          solverNew.h
          solverNew_types.h
```

Les fichiers sources étant en place, il faut modifier les paramètres de compilation du projet pour qu'ils prennent en compte le nouveau solveur. Pour cela, il faut modifier le fichier `CMakeLists.txt` contenu dans le dossier solverNew. Les modifications à apporter sont présentées ci-dessous en rouge :

```
PROJECT( solverNew )
SET( lib_NAME solverNew )
SET( lib_SRC_FILES solverNew.c )
ADD_LIBRARY( ${lib_NAME} STATIC ${lib_SRC_FILES} )
TARGET_LINK_LIBRARIES(${lib_NAME} ${TriMeshSolver_LIBS})
IF(COMPIL_TEST_EXEC)
          ADD_SUBDIRECTORY( test )
ENDIF(COMPIL_TEST_EXEC)
```

Il faut ensuite remonter d'un cran pour se replacer dans le dossier parent src pour ajouter une ligne à deux endroits dans le fichier `CMakeLists.txt` qui s'y trouve :

- premier ajout

```
SET( TriMeshSolver_LIBS $PETSC_LIBRARIES
m
cunit
globals
...
...
...
solverRelax
solverRelaxImp
solverNew
main
)
```

– second ajout

```
ADD_SUBDIRECTORY( common )
ADD_SUBDIRECTORY( mesh )
ADD_SUBDIRECTORY( debug )
...
...
...
ADD_SUBDIRECTORY( solverRelax )
ADD_SUBDIRECTORY( solverRelaxImp )
ADD_SUBDIRECTORY( solverNew )
ADD_SUBDIRECTORY( main )
```

Tout est maintenant réuni pour réaliser la compilation (voir paragraphe précédent) en tenant compte du nouveau solveur.

6.2 Connectivité

La gestion de la connectivité du maillage a été décomposée en deux étapes. La première phase a consisté à recueillir les informations propres au maillage directement disponibles dans les fichiers `.neigh`, `.ele`, `.edge` et `.node` générés par Triangle. On récupère par exemple les coordonnées des sommets et leurs indices. Ensuite, une étape de post-traitement de ces données a été réalisée pour dégager des données supplémentaires nécessaires à l'algorithme. Parmi les tâches réalisées pendant cette phase, on est amené à calculer la normale à chaque arête ou encore à évaluer la longueur de ces dernières. Une fois ces deux étapes correctement réalisées, on peut considérer que la structure `mesh_t` a été complètement remplie.

```
struct mesh_t {
    int nbVertices;
    int nbCells;
    int nbEdges;
    vertex_t *vertex;
    cell_t *cell;
    edge_t *edge;
    real_t minCellArea;
};
```

La structure `mesh_t` concentre toutes les informations relatives au maillage.

`nbVertices` correspond au nombre de sommets

`nbCells` correspond au nombre de cellules

`nbEdges` correspond au nombre d'arêtes

`*vertex` donne l'adresse du premier élément de type `vertex_t` (leur nombre total est égal à `nbVertices`)

`*cell` donne l'adresse du premier élément de type `cell_t` (leur nombre total est égal à `nbCells`)

`*edge` donne l'adresse du premier élément de type `edge_t` (leur nombre total est égal à `nbEdges`)

```
struct vertex_t {
    int id;
    point_t coord;
    int region;
};
```

avec

```
struct point_t {
    real_t x; /* x-coordinate */
    real_t y; /* y-coordinate */
};
```

La structure `vertex_t` contient les informations relatives à un sommet.

`id` correspond à son indice

`coord` désigne ses coordonnées

`region` établit l'appartenance ou non du sommet à une frontière (`region=0` si le sommet n'appartient à aucune frontière, `region>0` sinon)

```
struct edge_t {
    int id;
    int region;
    cell_t *pcellW;
    cell_t *pcellE;
    vertex_t *pvertN;
    vertex_t *pvertS;
    vect_t normal; /* from west to east */
    real_t length;
};
```

avec

```
typedef struct point_t vect_t;
```

La structure `edge_t` contient les informations relatives à une arête.

`id` correspond à son indice

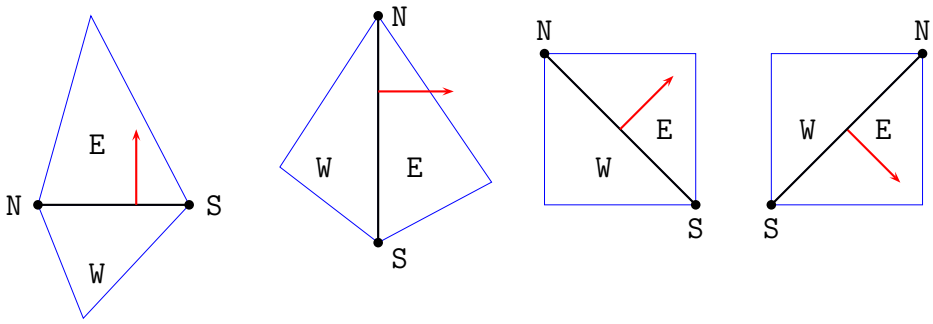
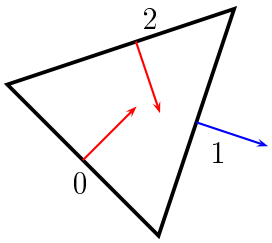


FIG. 10 – Situations possibles des normales à une arête



normalOrientation[0] = −1

normalOrientation[1] = +1

normalOrientation[2] = −1

FIG. 11 – Orientation des normales aux arêtes

region établit l'appartenance ou non de l'arête à une frontière (region=0 si l'arête n'appartient à aucune frontière, region=1 si l'arête appartient à la frontière 1 et ainsi de suite)

*pcellW désigne la cellule Ouest (W) de l'arête

*pcellE désigne la cellule Est (E) de l'arête

*pvertN désigne le sommet Nord (N) de l'arête

*pvertS désigne le sommet Sud (S) de l'arête

normal correspond à la normale à l'arête toujours orientée de l'Ouest vers l'Est, voir figure 10

length donne la longueur de l'arête.

```
struct cell_t {
    int id;
    vertex_t *pvertex[3];
    cell_t *pneighbor[3];
    edge_t *pedge[3];
    int normalOrientation[3]; /* +1 or -1 */
    point_t center;
    real_t area;
};
```

La structure `cell_t` contient les informations relatives à une cellule.

`id` correspond à son indice

`*pneighbor[3]` désigne les trois voisins de la cellule

`*pedge[3]` désigne les trois arêtes de la cellule

`normalOrientation[3]` donne l'orientation des normales à chacune des trois arêtes de la cellule (pour `i={0,1,2}`, `normalOrientation[i]=1` si la normale de l'arête `i` est sortante pour la cellule ou `normalOrientation[i]=-1` sinon, voir figure 11)

`center` désigne son centre de gravité

`area` donne l'aire de la cellule

6.3 Utilisation de macros

Pour faciliter la prise en main du code, certaines macros sont utilisées pour masquer les spécificités propres au langage C afin de se rapprocher le plus possible de la rédaction en pseudo-code. Voici quelques unes de ces macros utilisées fréquemment dans le code :

- **Accéder aux informations générales du maillage**

```
  /* en supposant ppbm correctement alloué */
int n ;
problem_t *ppbm ;
```

macro	équivalent en C
`n=GetMeshNbCells(ppbm)`	`n=ppbm->pmesh->nbCells`
`n=GetMeshNbVertices(ppbm)`	`n=ppbm->pmesh->nbVertices`
`n=GetMeshNbEdges(ppbm)`	`n=ppbm->pmesh->nbEdges`

- **Accéder à un objet du maillage**

```
  /* en supposant mesh, vertex, edge, cell correctement alloués */
mesh_t *mesh ;
int rank ;
vertex_t *vertex ;
edge_t *edge ;
cell_t *cell ;
```

macro	équivalent en C
`vertex=GetMeshVertex(mesh,rank)`	`vertex=&(mesh->vertex[rank])`
`cell=GetMeshCell(mesh,rank)`	`cell=&(mesh->cell[rank])`
`edge=GetMeshEdge(mesh,rank)`	`edge=&(mesh->edge[rank])`

- **Accéder à un objet lié à un autre**

```
  /* en supposant vertex, edge et cell correctement alloués */
int rank ;
vertex_t *vertex ;
edge_t *edge ;
cell_t *cell ;
```

macro	équivalent en C
vertex=GetEdgeVertN(edge)	vertex=edge->pvertN
vertex=GetEdgeVertS(edge)	vertex=edge->pvertS
cell=GetEdgeCellW(edge)	cell=edge->pcellW
cell=GetEdgeCellE(edge)	cell=edge->pcellE
edge=GetCellEdge(cell,rank)	edge=cell->pedge[rank]
vertex=GetCellVertex(cell,rank)	vertex = cell->pvertex[rank]

• **Automatiser les boucles générales**

En écrivant les lignes suivantes dans un fichier header utilisé par tout le code :

```
#define EachMeshCell(m,i) (i) = 0 ; (i) < (m)->nbCells ; (i)++
#define EachMeshEdge(m,e) (e) = 0 ; (e) < (m)->nbEdges ; (e)++
#define EachMeshVertex(m,v) (v) = 0 ; (v) < (m)->nbVertices ; (v)++
```

on peut obtenir les simplifications suivantes :

macro	équivalent en C

for(EachMeshCell(mesh,i)){
...
}
	for(i=0 ;i<mesh->NbCells ;i++){
	...
	}

for(EachMeshEdge(mesh,i)){
...
}
	for(i=0 ;i<mesh->NbEdges ;i++){
	...
	}

for(EachMeshVertex(mesh,i)){
...
}
	for(i=0 ;i<mesh->NbVertices ;i++){
	...
	}

6.4 Solveur de relaxation explicite

Le fonctionnement du solveur de relaxation explicite sur un pas de temps (*ie* évaluer $\mathbb{U}_K^{n+1}$ à partir de $\mathbb{U}_K^n$ $\forall K \in \mathcal{T}_h$) peut se décomposer en trois grandes étapes :

- calcul des flux associés à chaque arête du maillage sans se soucier de l'orientation des normales. L'opération est assurée par la routine `ComputeFluxForEachEdge()` qui gère la double éventualité « l'arête appartient à la frontière / l'arête n'appartient pas à la frontière » qui induit des calculs différents. Le cœur du solveur se trouve dans cette fonction car c'est ici que s'opère, entre autre, la résolution du problème de Riemann associé à chaque arête dans la fonction `ComputeFlux()`. Les rotations évoquées précédemment sont opérées par les fonctions `ComputeLocalCoord()` et `ComputeGlobalCoord()` ;
- calcul de la somme des flux associée à chaque cellule en tenant compte de l'orientation des normales. Cette étape est réalisée par la fonction `ComputeSumFluxForEachCell()`. Cette étape ne repose sur aucun résultat théorique fort comme le demandait l'étape précédente et ne fait que respecter la propriété de conservation de flux suivante :

$$\mathcal{G}_e^n \langle \vec{n}_e, \vec{n}_{e,W} \rangle = -\mathcal{G}_e^n \langle \vec{n}_e, \vec{n}_{e,E} \rangle$$

  avec E et W les cellules Est et Ouest de l'arête e ;
- mise à jour des valeurs des variables associées à chaque cellule à partir des flux calculés grâce à la fonction `ValuesUpdateForEachCell()`. Le calcul étant terminé,

cette étape ne fait que mettre à jour les variables contenues dans la structure `eulerData_t`.

6.5 Solveur de relaxation implicite

6.5.1 Fonctionnement général du solveur

Le fonctionnement du solveur de relaxation implicite sur un pas de temps peut se décomposer en quatre grandes étapes :

- calcul des flux associés à chaque arête du maillage sans soucier de l'orientation des normales. L'opération est exactement la même que pour la méthode explicite ;
- calcul de la somme des flux associée à chaque cellule en tenant compte de l'orientation des normales. Cette étape est réalisée de la même manière que pour la relaxation explicite ;
- construction et résolution du système linéaire résultant de l'implicitation du flux linéarisé. Il s'agit de la seule étape qui marque une différence avec la méthode explicite. Elle est détaillée dans le paragraphe suivant ;
- mise à jour des valeurs des variables associées à chaque cellule comme cela a été fait en explicite.

6.5.2 Systèmes linéaires avec PETSc

Contrairement à la version explicite du solveur, on a besoin ici d'utiliser des structures matricielle et vectorielle car la méthode implicite exige la résolution de systèmes linéaires. Nous avons donc eu recours à PETSc (Portable, Extensible Toolkit for Scientific Computation, voir [1]) qui permet la manipulation de matrices et la résolution de systèmes linéaires à partir d'un code en C. PETSc offre bien d'autres possibilités comme la gestion de calculs parallèles mais cela ne nous intéresse pas ici. En ce qui nous concerne, PETSc n'est véritablement utilisé qu'à 3 niveaux :

- assemblage de la matrice A assuré par la fonction `AssembleMatrixBlock()` ;
- construction du second membre b opérée par la fonction `BuildRightHandVectorBlock()` ;
- résolution du système $Ax = b$ avec la fonction `SolveLinearSystem()` qui utilise les méthodes de Krylov.

7 Visualisation des données

7.1 Fichier `vtk`

Pour la visualisation des résultats donnés par le code, on a choisi le logiciel ParaView qui impose une convention dans la lecture des données. Tout d'abord, le nom du fichier qui contient les résultats a pour extension `.vtk`. Dans notre situation, le code donne comme nom, `data00x.vtk`, à notre fichier qui correspond à la `x`-ième sauvegarde des résultats (`x=1`, `x=2`, ...). Observons dans un cas particulier le contenu d'un fichier de ce type situé dans le dossier où se trouvent les fichiers nécessaires à l'initialisation du code :

```
# vtk DataFile Version 3.1
TMS data at instant t=0.00935618
ASCII
```

```
DATASET UNSTRUCTURED_GRID
POINTS 16 FLOAT # 16 points (x, y, z) defining the grid
# remark : z is always equal to 0 as we solve 2D problems
 0 0 0          # x_0 y_0 z_0
 0.5 0 0        # x_1 y_1 z_1
 1 0 0
 1 0.1 0
 0.5 0.1 0
 0 0.1 0
 0.5 0.05 0
 0.25 0 0
 0.75 0 0
 0.375 0 0
 0.75 0.1 0
 0.625 0 0
 0.25 0.1 0
 0.625 0.1 0
 0.375 0.1 0
 0.125 0.05 0  # x_15 y_15 z_15
# we have 16 cells and we need 4 values to define each cell so there are
# 4 × 16 = 64 values needed
CELLS 16 64
# each cell is a triangle so there are 3 points needed to define it with
# their index.
3 7 15 0
3 6 1 11
3 6 9 1
3 7 12 15
3 11 13 6
3 3 8 2
3 11 8 10
3 4 6 13
3 14 6 4
3 5 0 15
3 8 3 10
3 14 12 9
3 9 12 7
3 11 10 13
3 6 14 9
3 12 5 15
CELL_TYPES 16
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  # each cell has 5 variables
CELL_DATA 16
SCALARS rho float 1 # rho is the first variable
LOOKUP_TABLE default
```

```
 1          # ρ0
 0.783807   # ρ1
 1.00362
 1
 0.75
 0.75
 0.75
 0.783807
 1.00362
 1
 0.75
 1
 1
 0.75
 1          # ρ14
 1          # ρ15
SCALARS pressure float 1 # pressure is the second variable
LOOKUP_TABLE default
 1          # p0
 0.666667   # p1
 1
 1
 0.666667
 0.666667
 0.666667
 0.666667
 1
 1
 0.666667
 1
 1
 0.666667
 1          # p14
 1          # p15
VECTORS velocity float # velocity contains the last three variables
# u,v and w
# velocity is a vector : (u,v,w) where w is always equal to 0.0
# as we solve 2D problems
```

```
1.03874e-17 0 0.0            # u_0 v_0 w_0
-0.265219 -7.96213e-14 0.0   # u_1 v_1 w_1
-0.00467767 0 0.0
-0 0 0.0
-0.333333 0 0.0
-0.333333 -4.16052e-14 0.0
-0.333333 -4.16329e-14 0.0
-0.265219 7.96213e-14 0.0
-0.00467767 0 0.0
-1.03874e-17 0 0.0
-0.333333 4.16052e-14 0.0
-0 0 0.0
-0 0 0.0
-0.333333 4.16329e-14 0.0
-0 0 0.0                      # u_14 v_14 w_14
-0 0 0.0                      # u_15 v_15 w_15
```

Les résultats présents dans le fichier ci-dessus sont visualisables sur la figure 12 à l'aide de ParaView.

7.2 Options de visualisation

Un grand avantage que propose ParaView c'est le traitement des données qui permet par exemple, de calculer les valeurs aux noeuds du maillage à partir des valeurs associées à chaque cellule avec l'option `Cell Data to Point Data` (voir figure 13). Une fois les valeurs ramenées aux noeuds, on peut observer en 3D, la répartition des grandeurs scalaires en fonction de x et y avec `Warp By Scalar` (voir figure 14). Par ailleurs, on peut aussi observer dans une coupe 1D l'évolution des grandeurs scalaires selon une direction avec l'option `Plot Over Line` (voir figure 15).

8 Résultats numériques

8.1 Tests quasi-1D

Les tests suivants sont réalisés sur des domaines quasi-1D selon x (y étant très petit devant x). Certains d'entre eux ont été employés dans [4] ou [2]. Les figures qui sont présentées dans ce paragraphe sont des courbes 1D à y fixé.

8.1.1 Test 1

Pour un domaine $\mathcal{D} = [0,1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.15$, (voir figures 16, 17 et 18) :

$$(\rho, u, v, p) = \begin{cases} (1,0,0,1) & \text{si } x \leq 0.5 \\ (0.125,0,0,0.1) & \text{si } x > 0.5 \end{cases}$$

Ce test a pour nom le tube à choc de Sod.

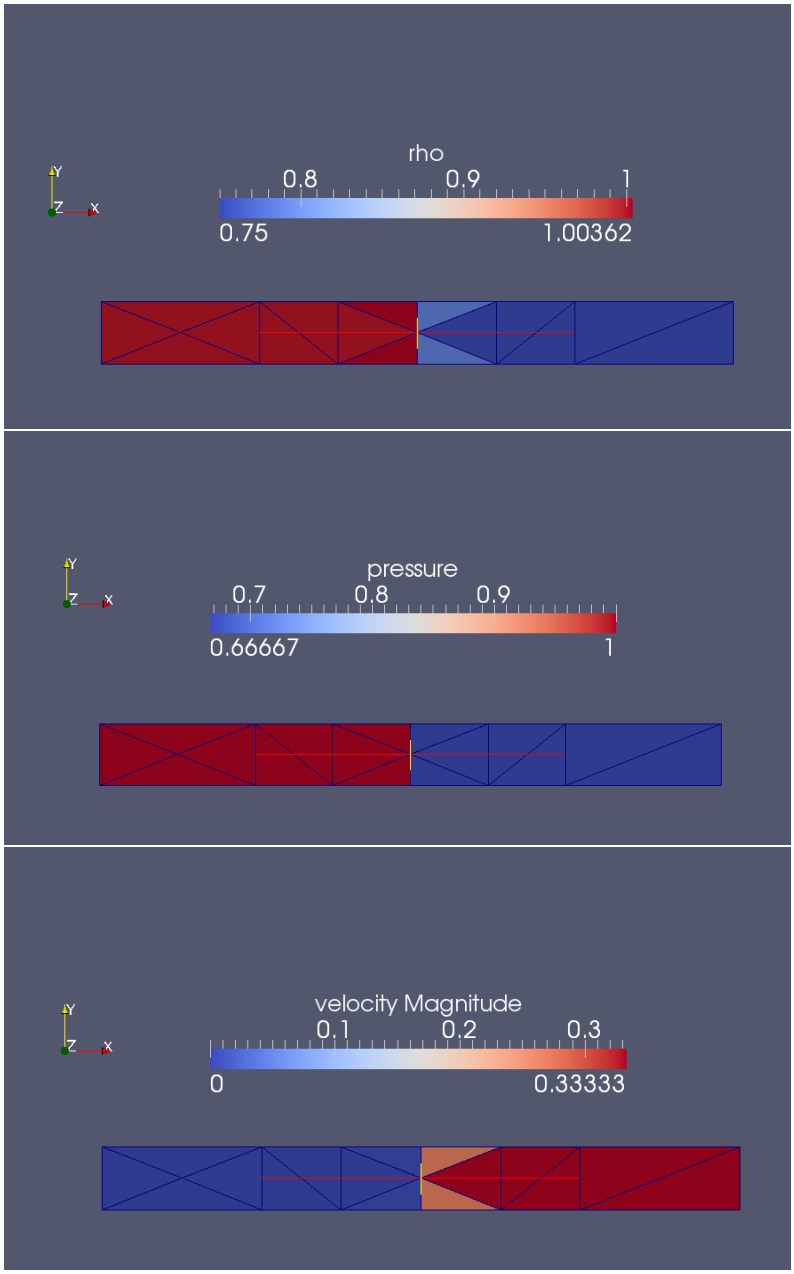


FIG. 12 – Valeurs de la densité, de la pression et de la norme de la vitesse selon le fichier `vtk`

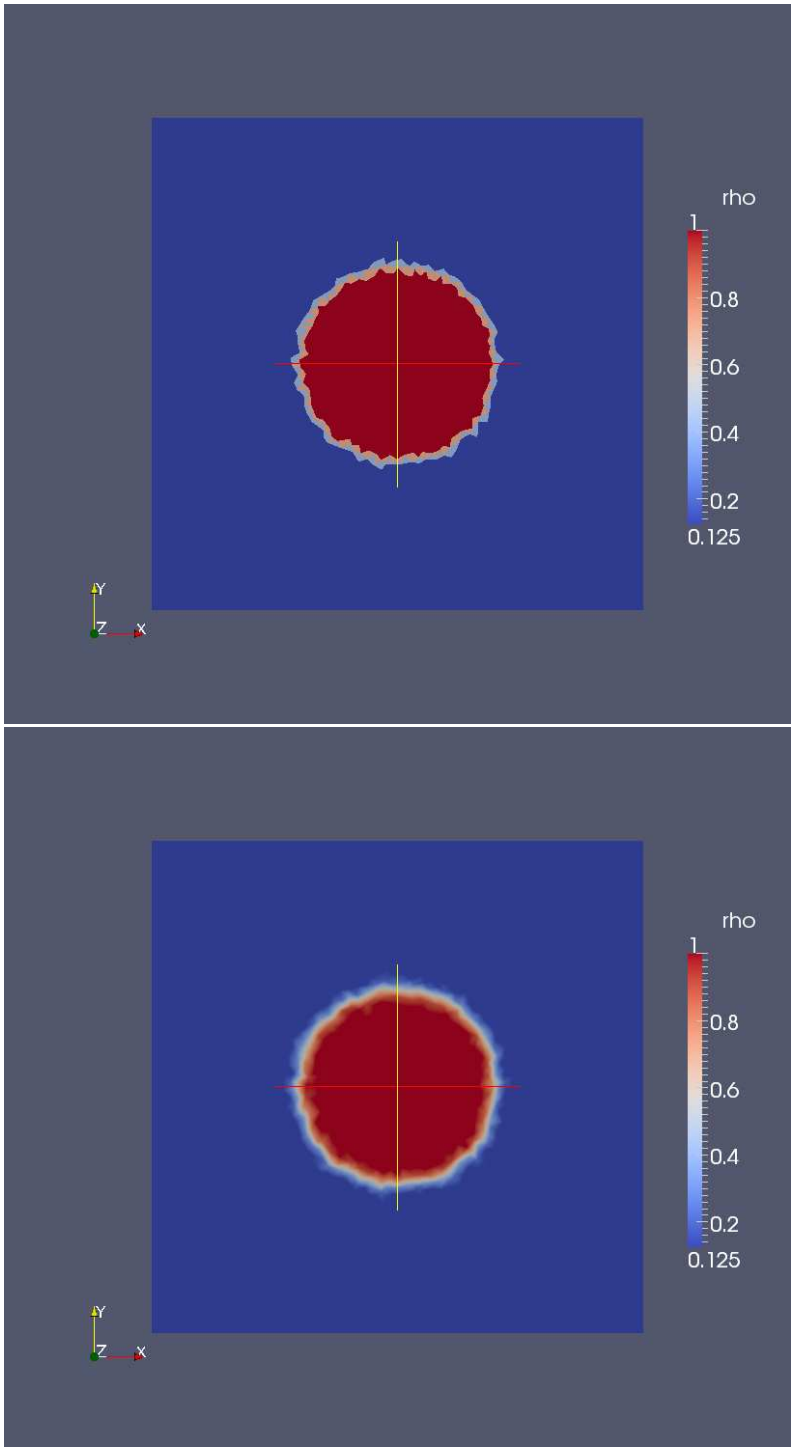


FIG. 13 – Valeurs de la densité associées aux cellules et associées aux noeuds du maillage

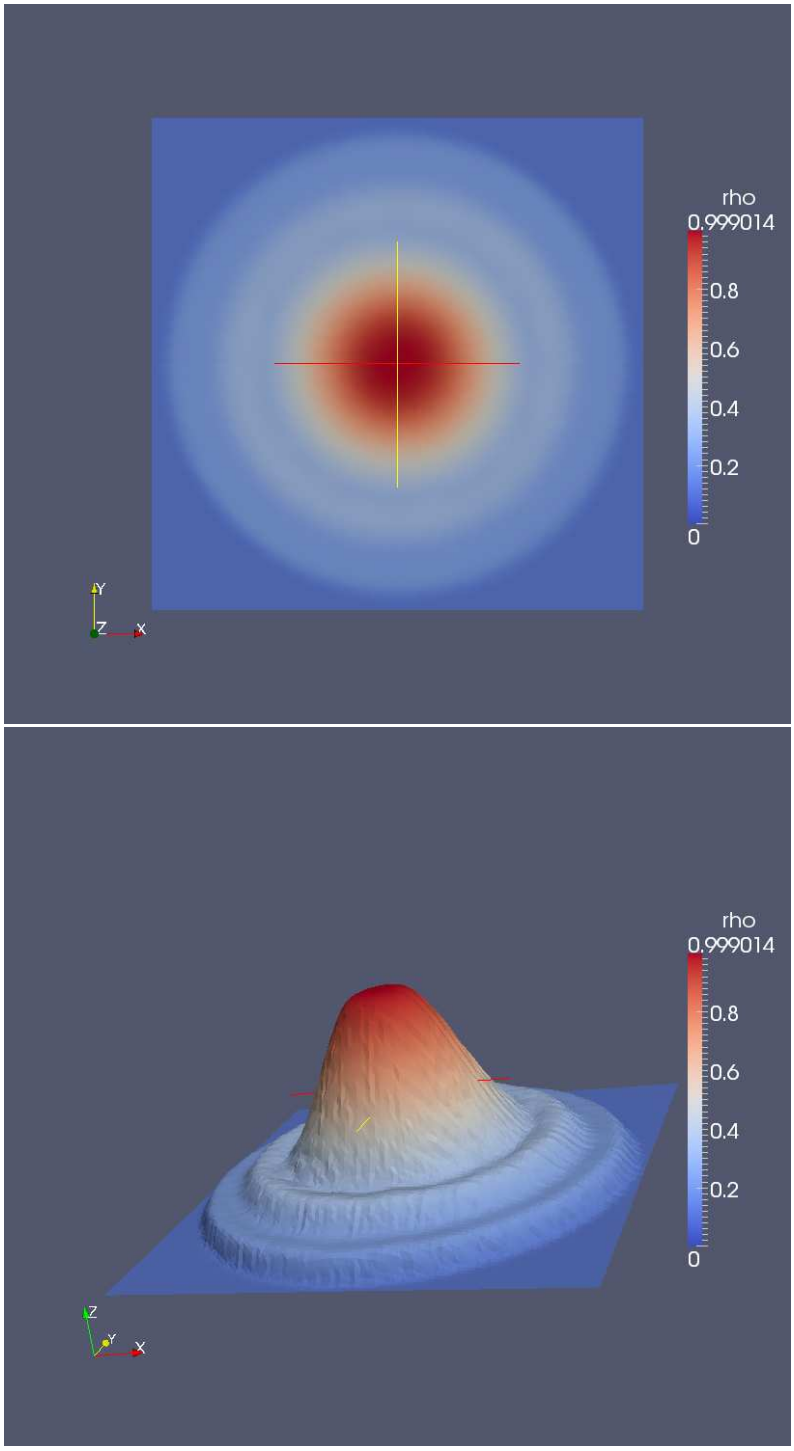


FIG. 14 – Visualisation des valeurs de la densité en 2D et en 3D

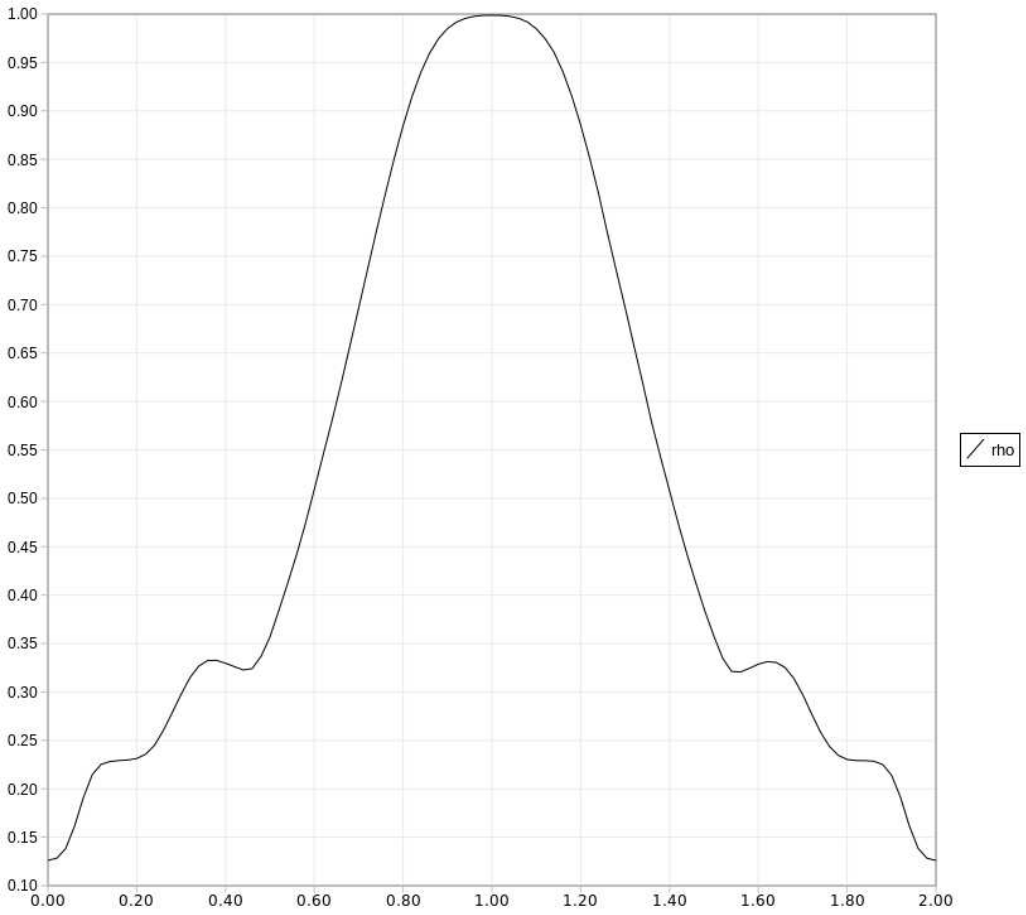


FIG. 15 – Visualisation en coupe 1D à $y = 1$ d'un problème où $(x, y) \in [0, 2] \times [0, 2]$

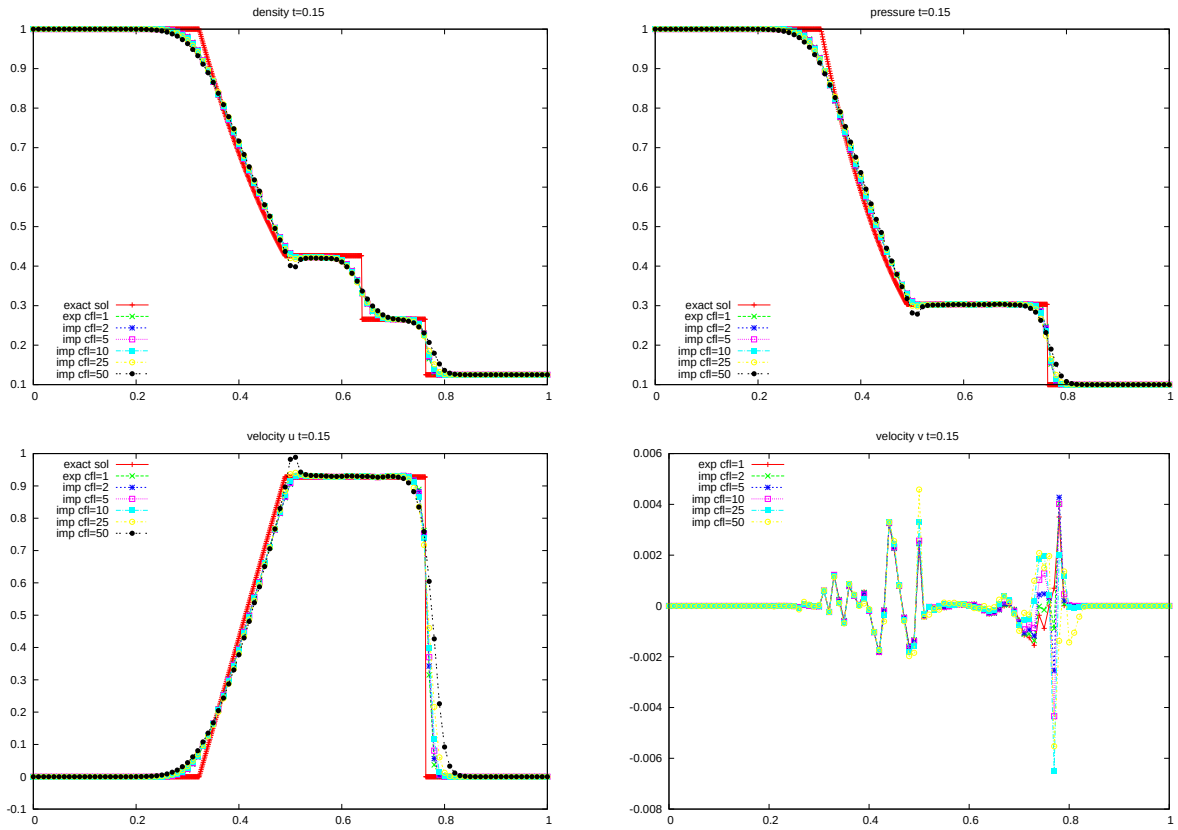


FIG. 16 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [2]

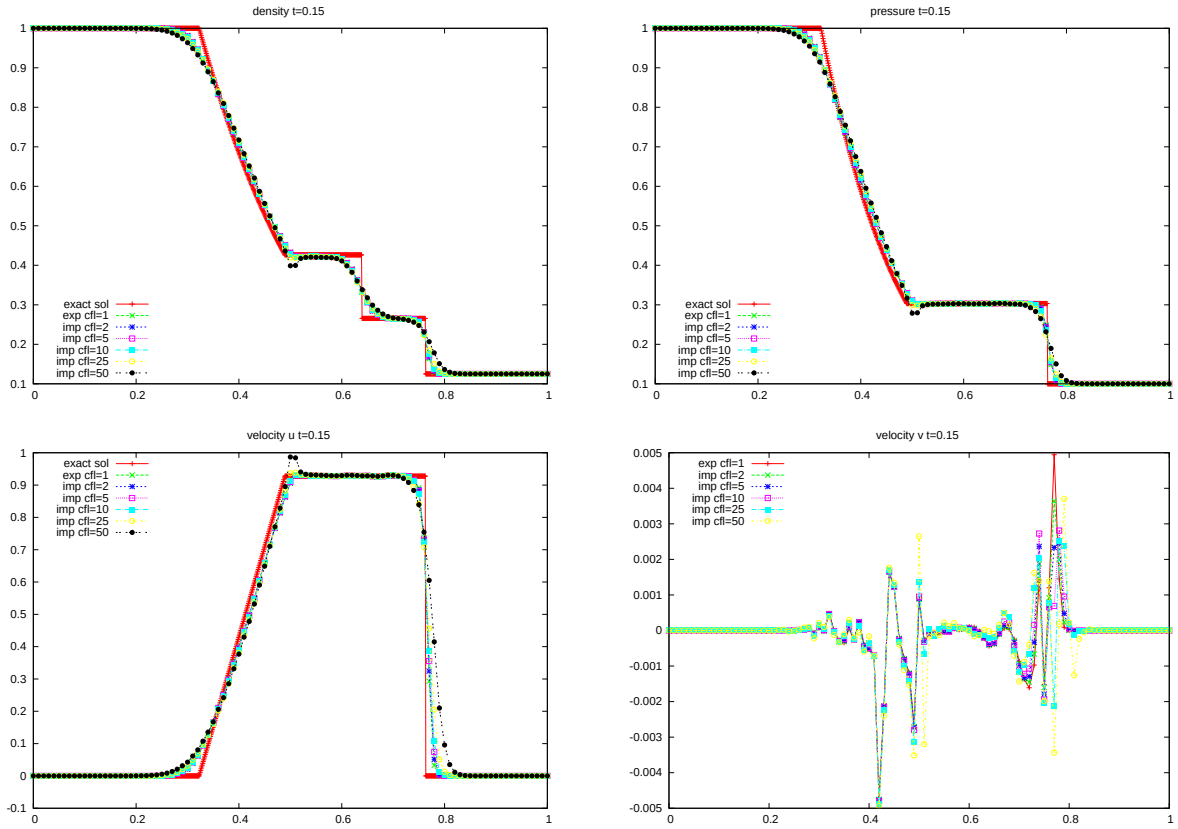


FIG. 17 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [2]

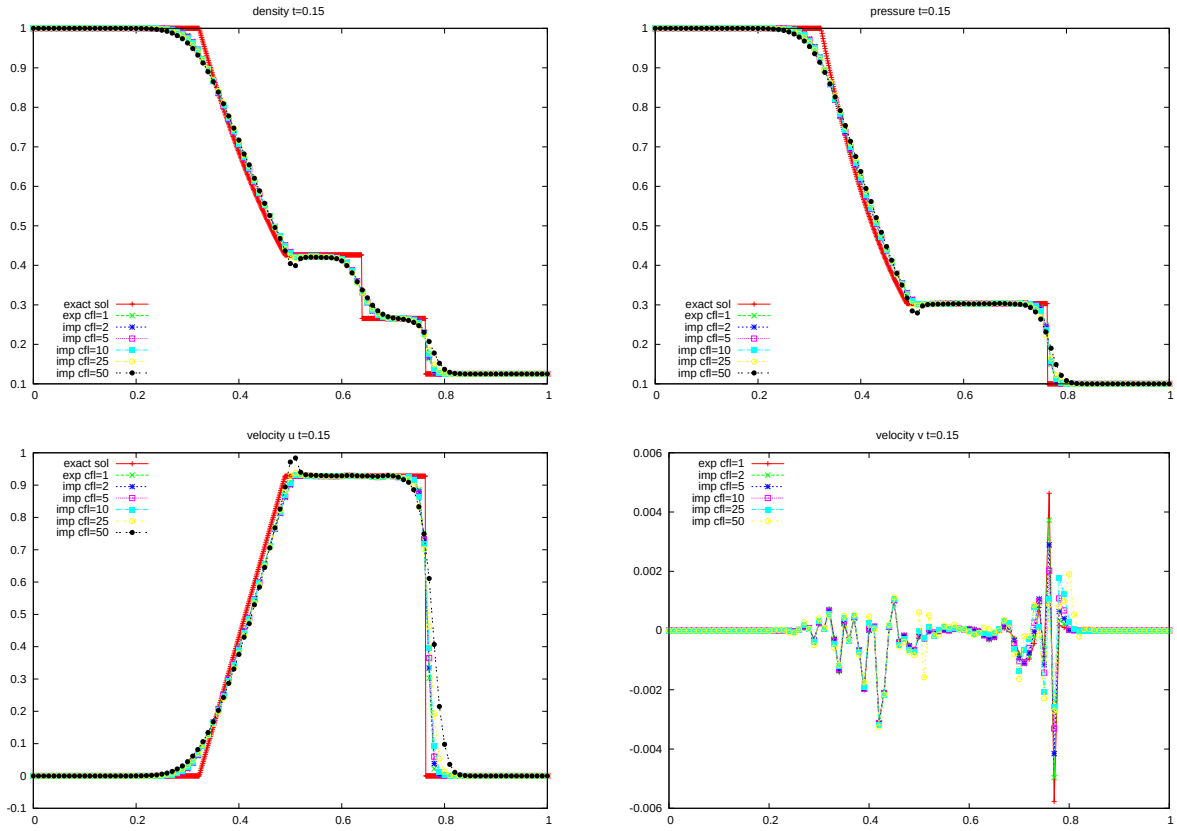


FIG. 18 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [2]

8.1.2 Test 2

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.2$, (voir figures 19, 20 et 21) :

$$(\rho, u, v, p) = \begin{cases} (1, 0.75, 0, 1) & \text{si } x \leq 0.3 \\ (0.125, 0, 0, 0.1) & \text{si } x > 0.3 \end{cases}$$

Ce test est une version modifiée du tube à choc de Sod utilisé pour le test 1.

8.1.3 Test 3

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.15$, (voir figures 22, 23 et 24) :

$$(\rho, u, v, p) = \begin{cases} (1, -2, 0, 0.4) & \text{si } x \leq 0.5 \\ (1, 2, 0, 0.4) & \text{si } x > 0.5 \end{cases}$$

8.1.4 Test 4

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.012$, (voir figures 25, 26 et 27) :

$$(\rho, u, v, p) = \begin{cases} (1, 0, 0, 1000) & \text{si } x \leq 0.5 \\ (1, 0, 0, 0.01) & \text{si } x > 0.5 \end{cases}$$

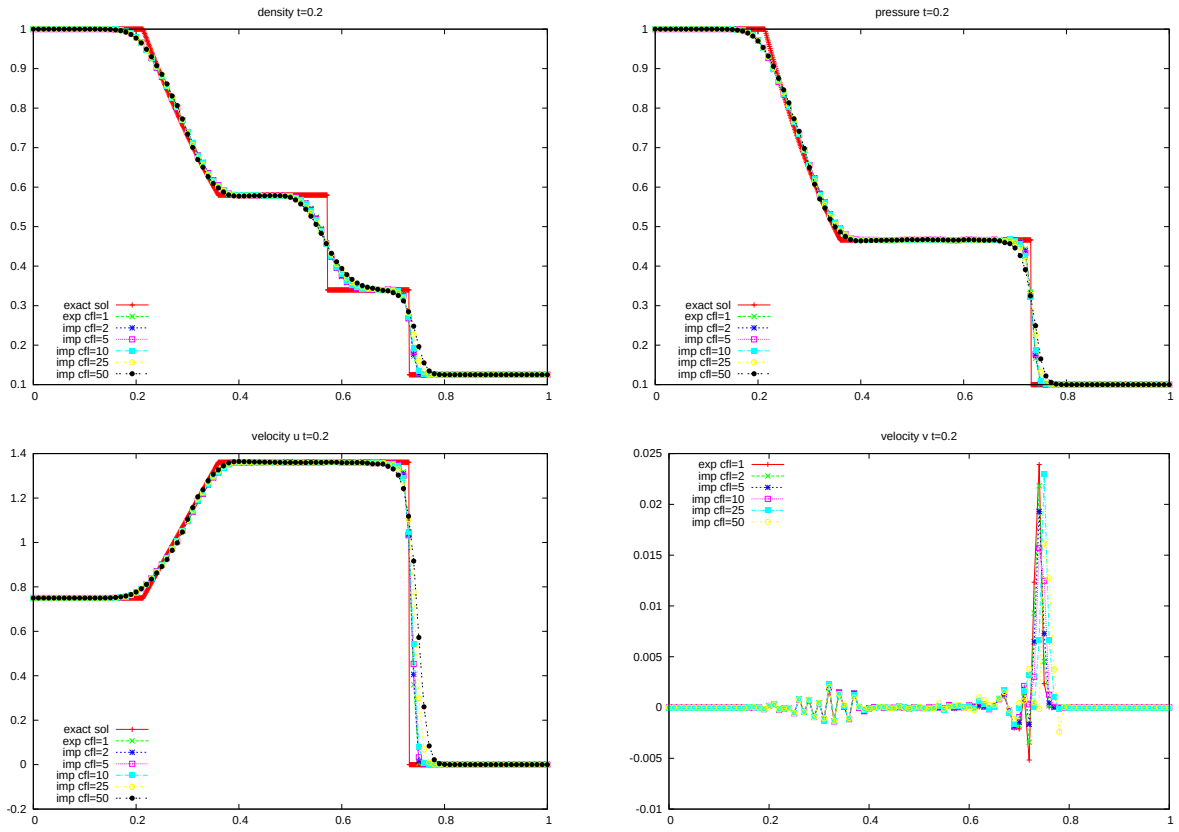


FIG. 19 – Test 2 : ρ, p, u et v à $t_f = 0.2$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [4]

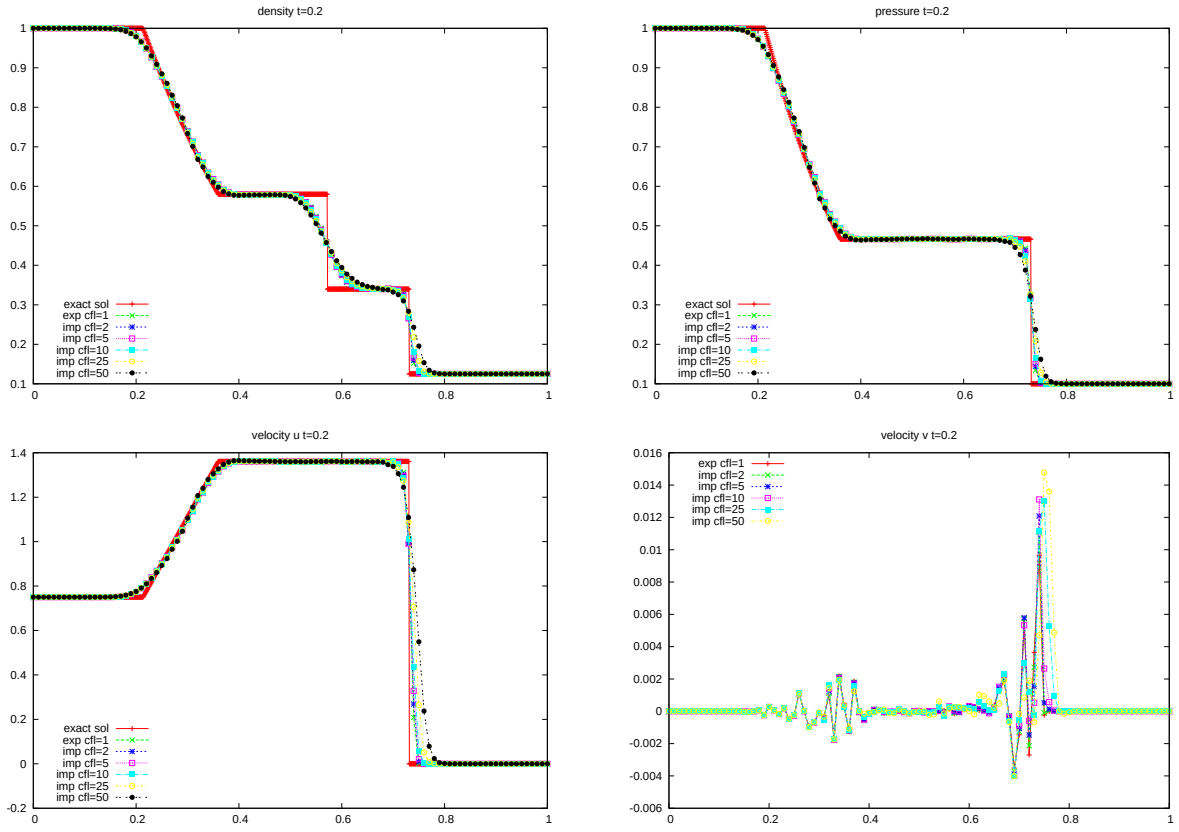


FIG. 20 – Test 2 : ρ, p, u et v à $t_f = 0.2$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [4]

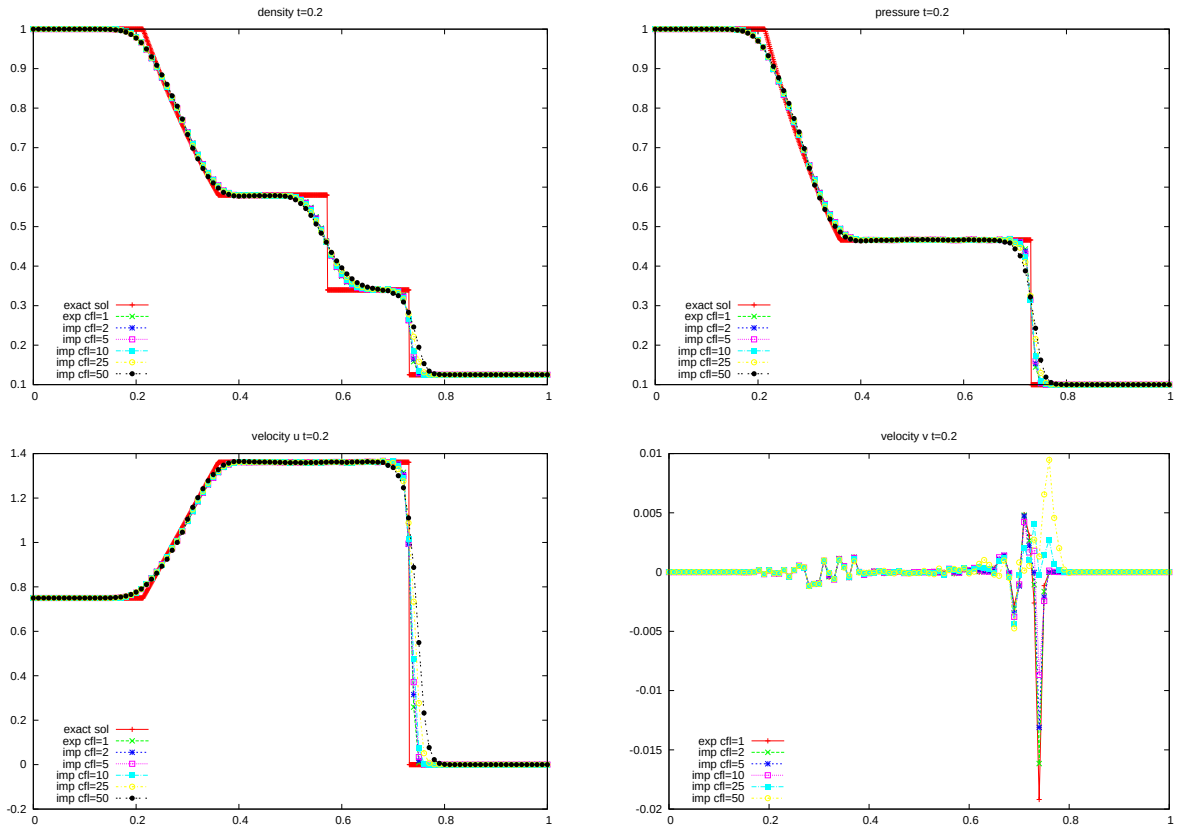


FIG. 21 – Test 2 : ρ, p, u et v à $t_f = 0.2$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [4]

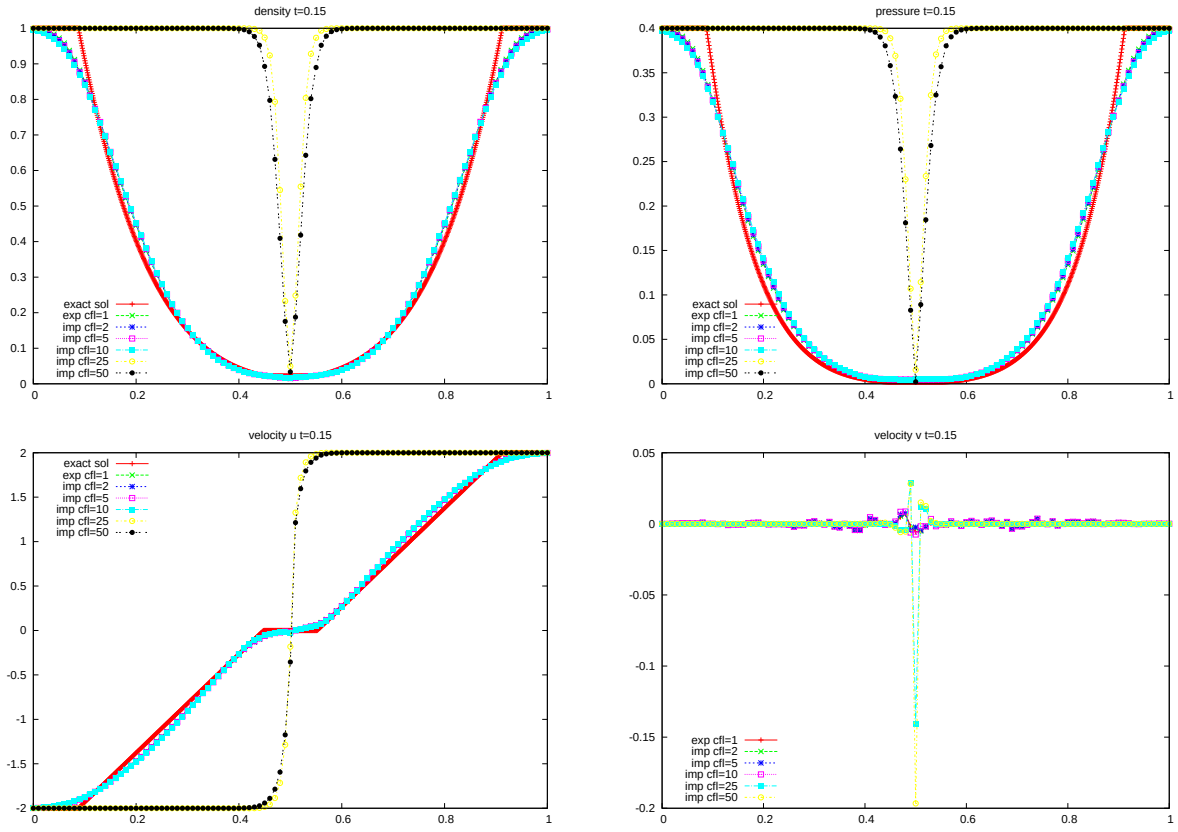


FIG. 22 – Test 3 : ρ, p, u et v à $t_f = 0.15$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [4]

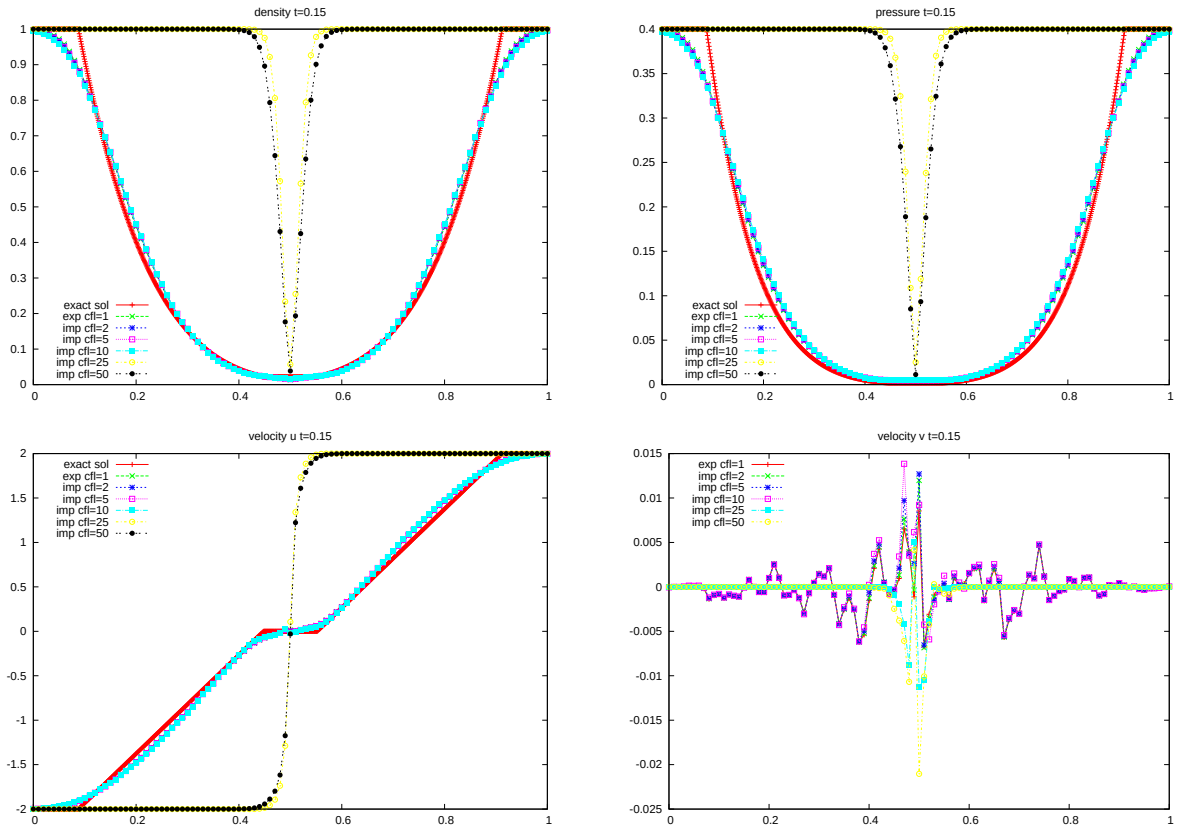


FIG. 23 – Test 3 : ρ, p, u et v à $t_f = 0.15$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [4]

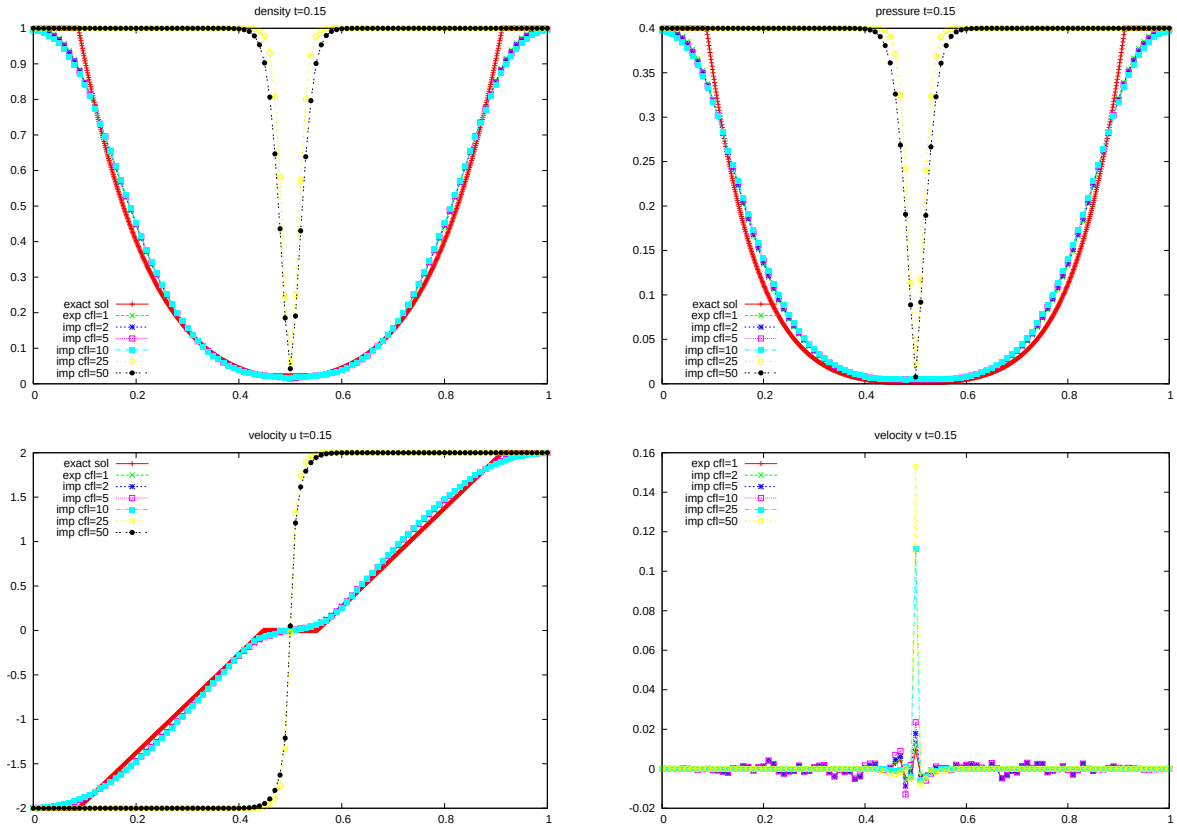


FIG. 24 – Test 3 : ρ, p, u et v à $t_f = 0.15$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [4]

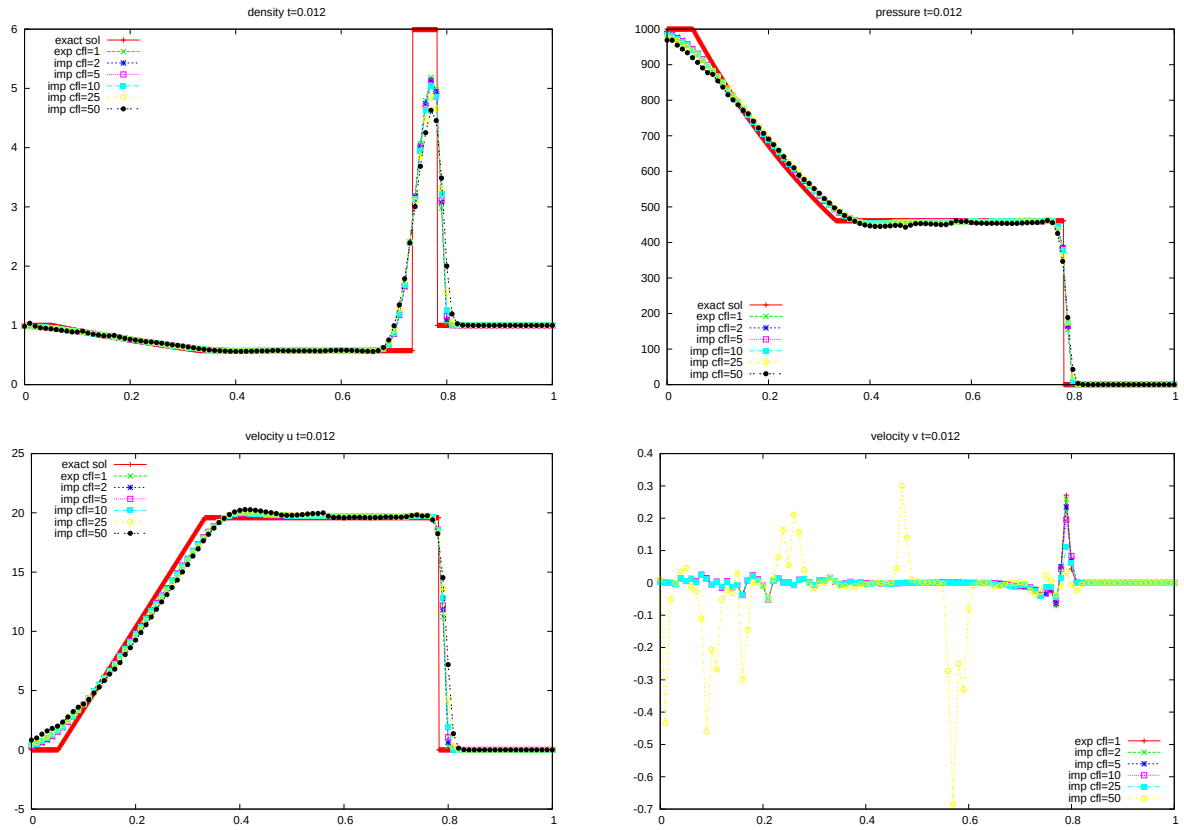


FIG. 25 – Test 4 : ρ, p, u et v à $t_f = 0.012$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [4]

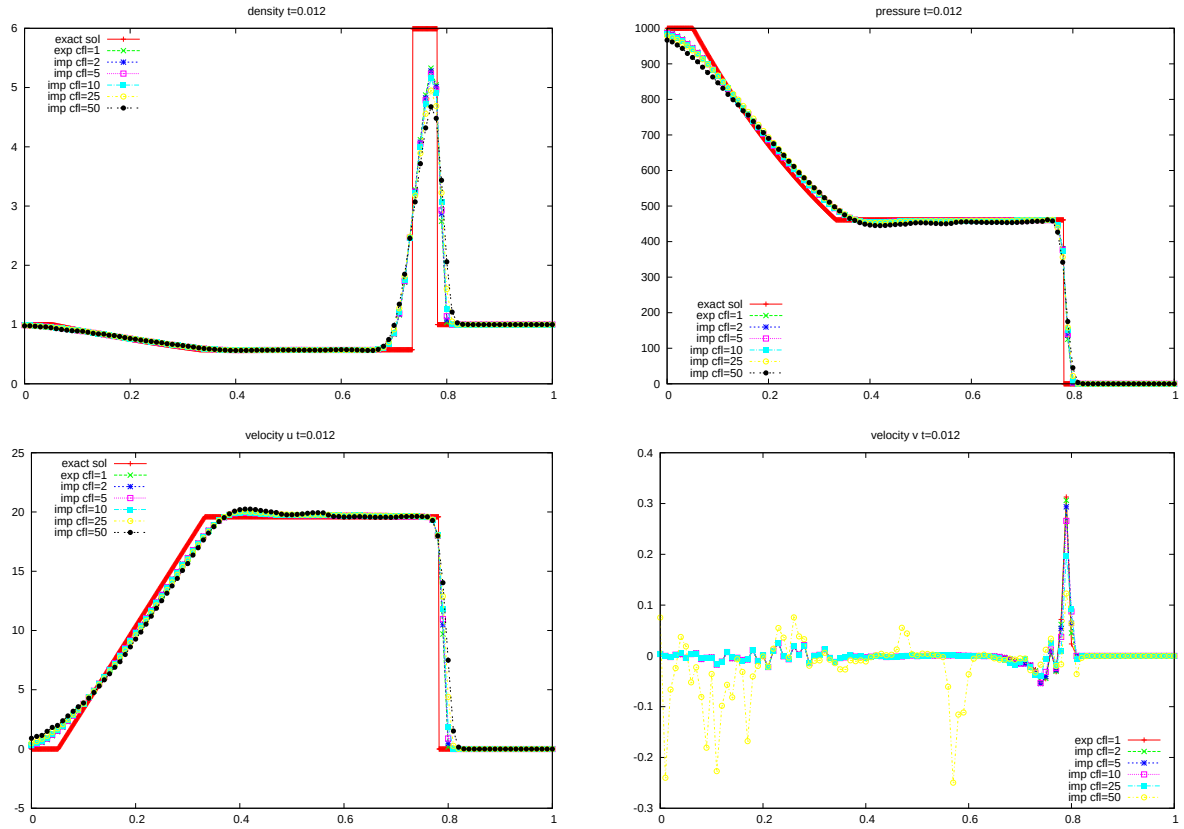


FIG. 26 – Test 4 : ρ, p, u et v à $t_f = 0.012$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [4]

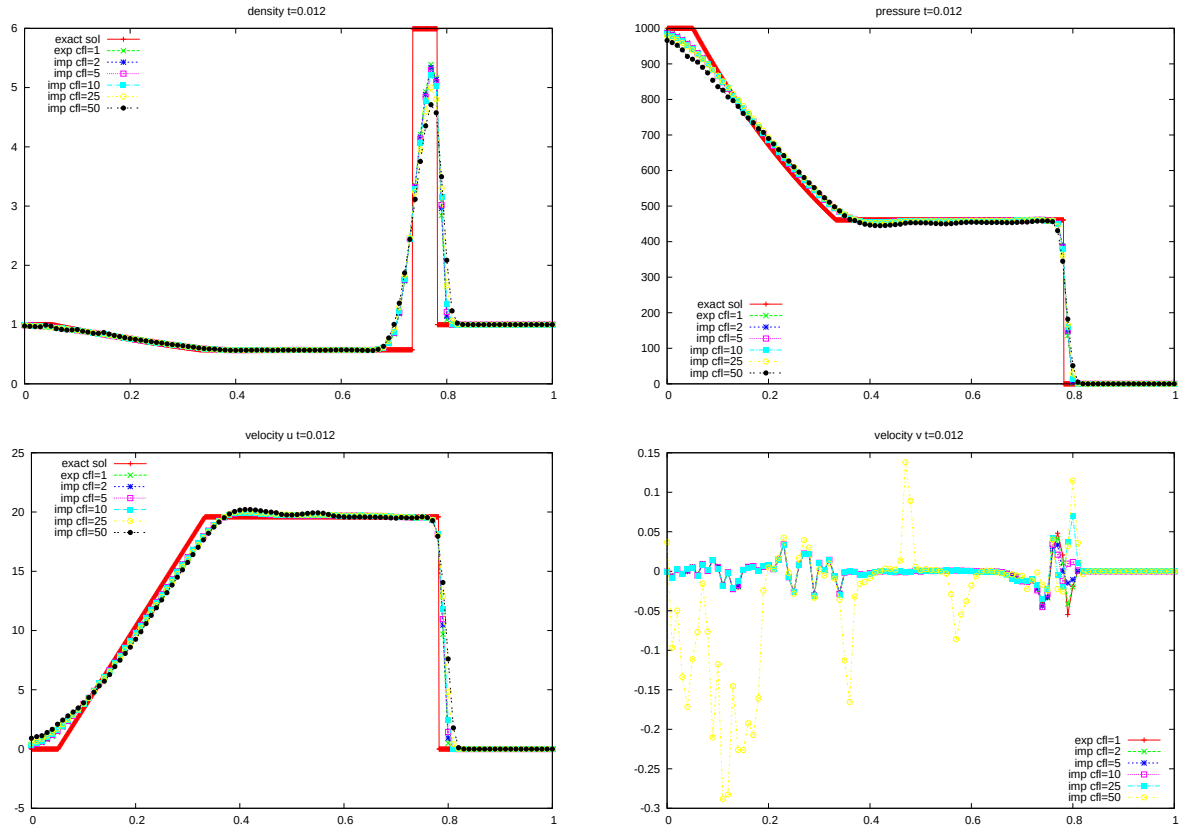


FIG. 27 – Test 4 : ρ, p, u et v à $t_f = 0.012$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [4]

8.1.5 Test 5

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.035$, (voir figures 28, 29 et 30) :

$$(\rho, u, v, p) = \begin{cases} (5.99924, 19.5975, 0, 460.894) & \text{si } x \leq 0.4 \\ (5.99242, -6.19633, 0, 46.0950) & \text{si } x > 0.4 \end{cases}$$

8.1.6 Test 6

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.012$, (voir figures 31, 32 et 33) :

$$(\rho, u, v, p) = \begin{cases} (1, -19.59745, 0, 1000) & \text{si } x \leq 0.8 \\ (1, -19.59745, 0, 0.01) & \text{si } x > 0.8 \end{cases}$$

8.1.7 Test 7

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.1$, (voir figures 34, 35 et 36) :

$$(\rho, u, v, p) = \begin{cases} (0.9, 3, 0, 2) & \text{si } x \leq 0.2 \\ (0.5, 2, 0, 1) & \text{si } x > 0.2 \end{cases}$$

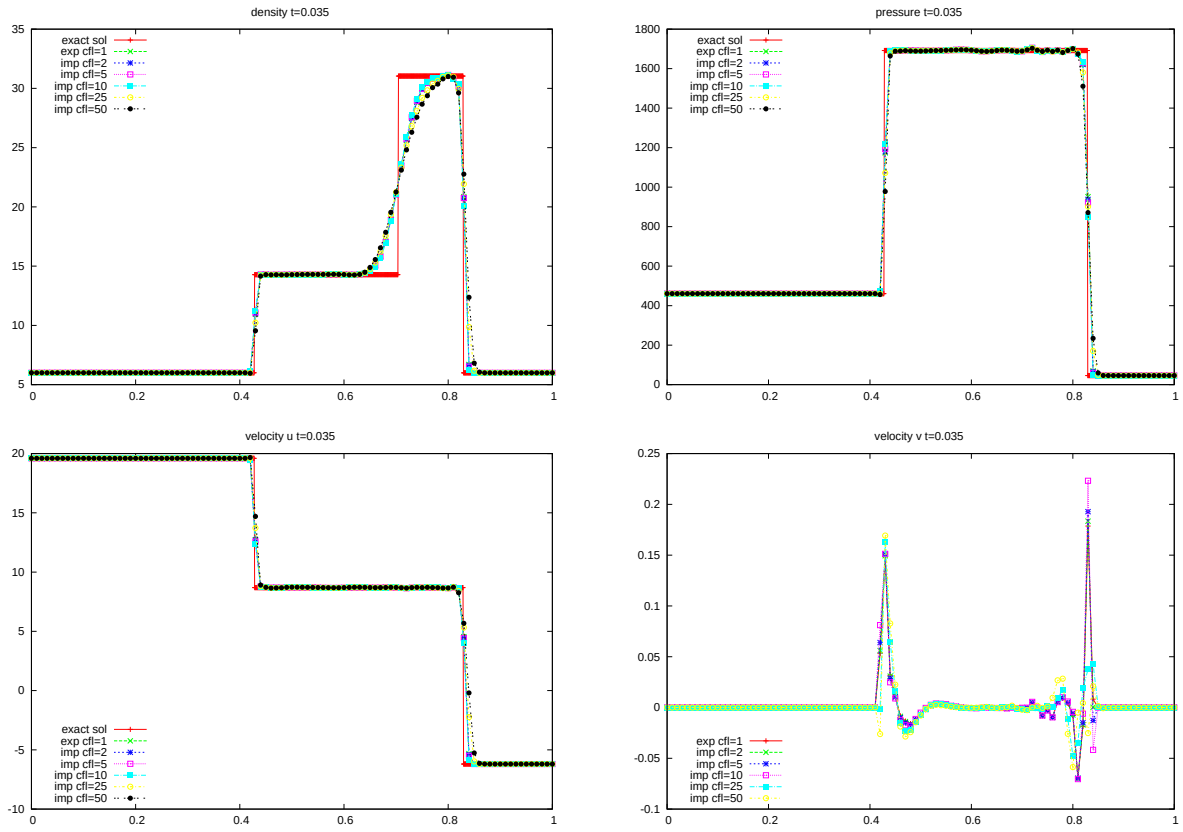


FIG. 28 – Test 5 : ρ, p, u et v à $t_f = 0.035$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [4]

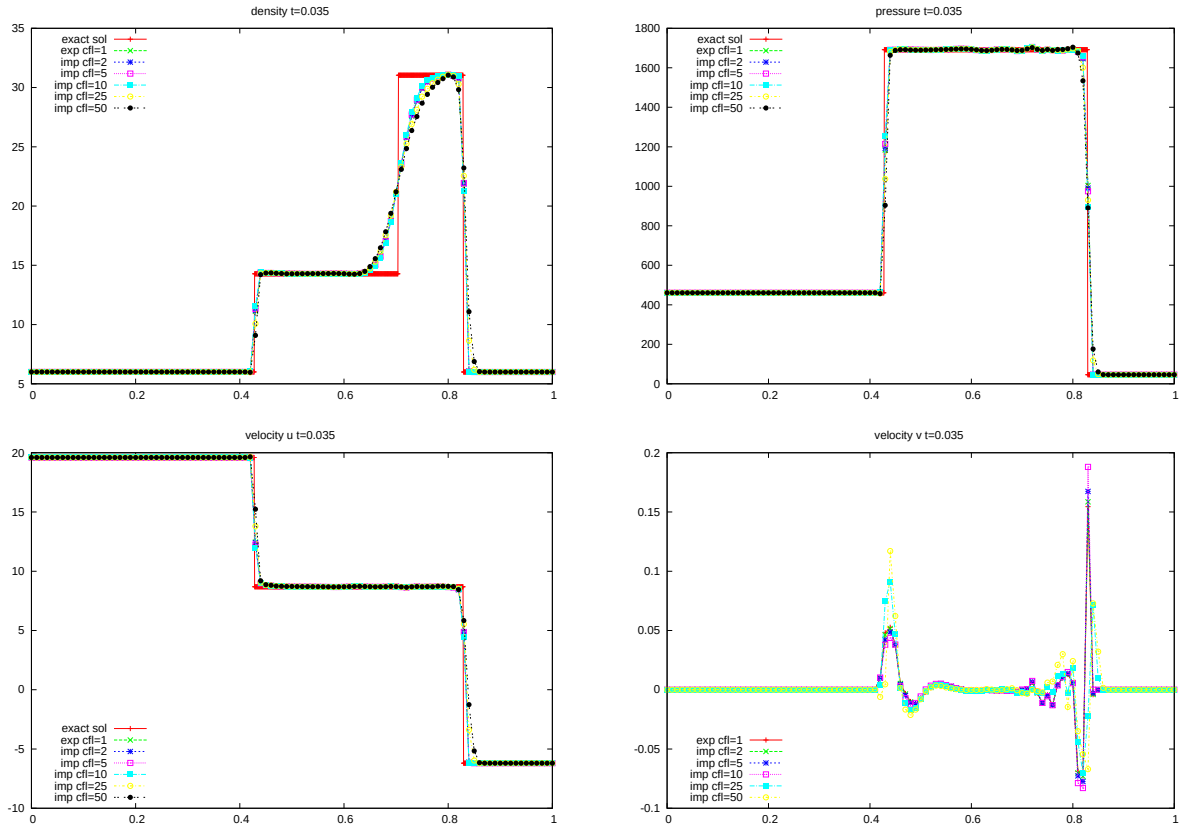


FIG. 29 – Test 5 : ρ, p, u et v à $t_f = 0.035$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [4]

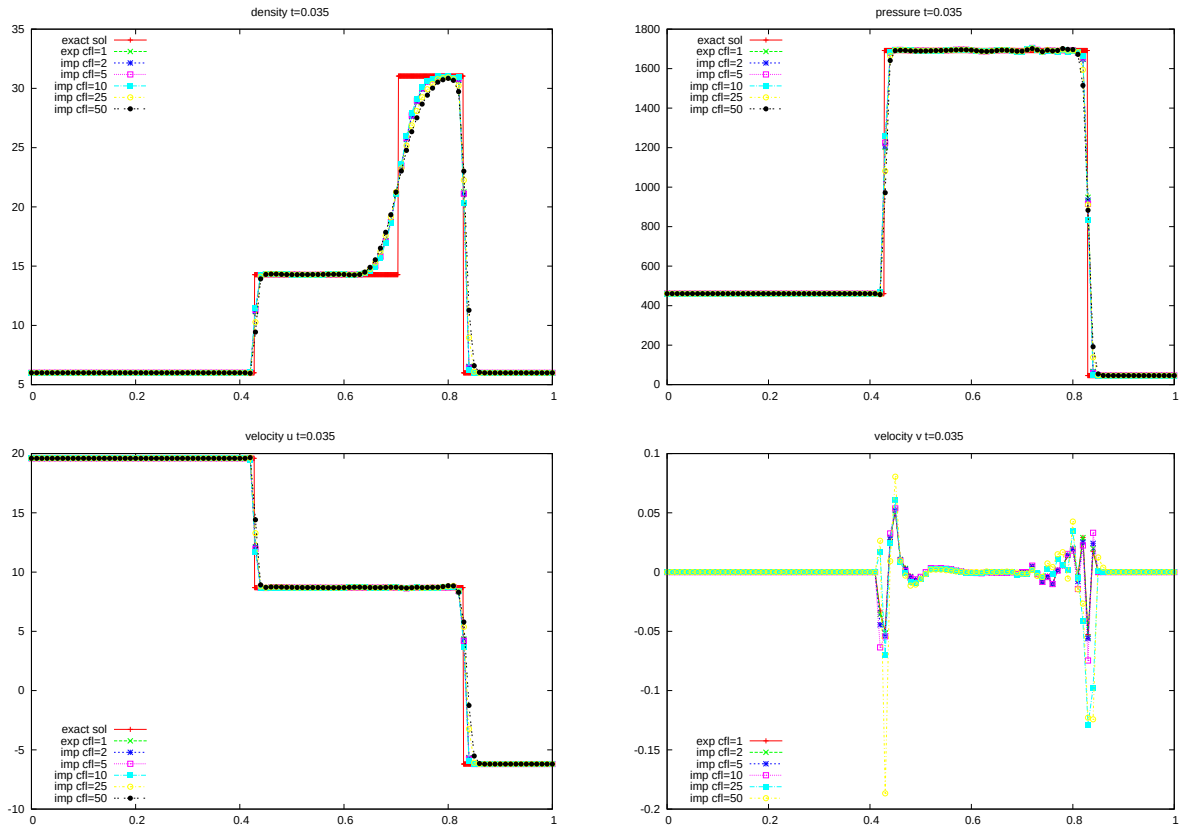


FIG. 30 – Test 5 : ρ, p, u et v à $t_f = 0.035$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [4]

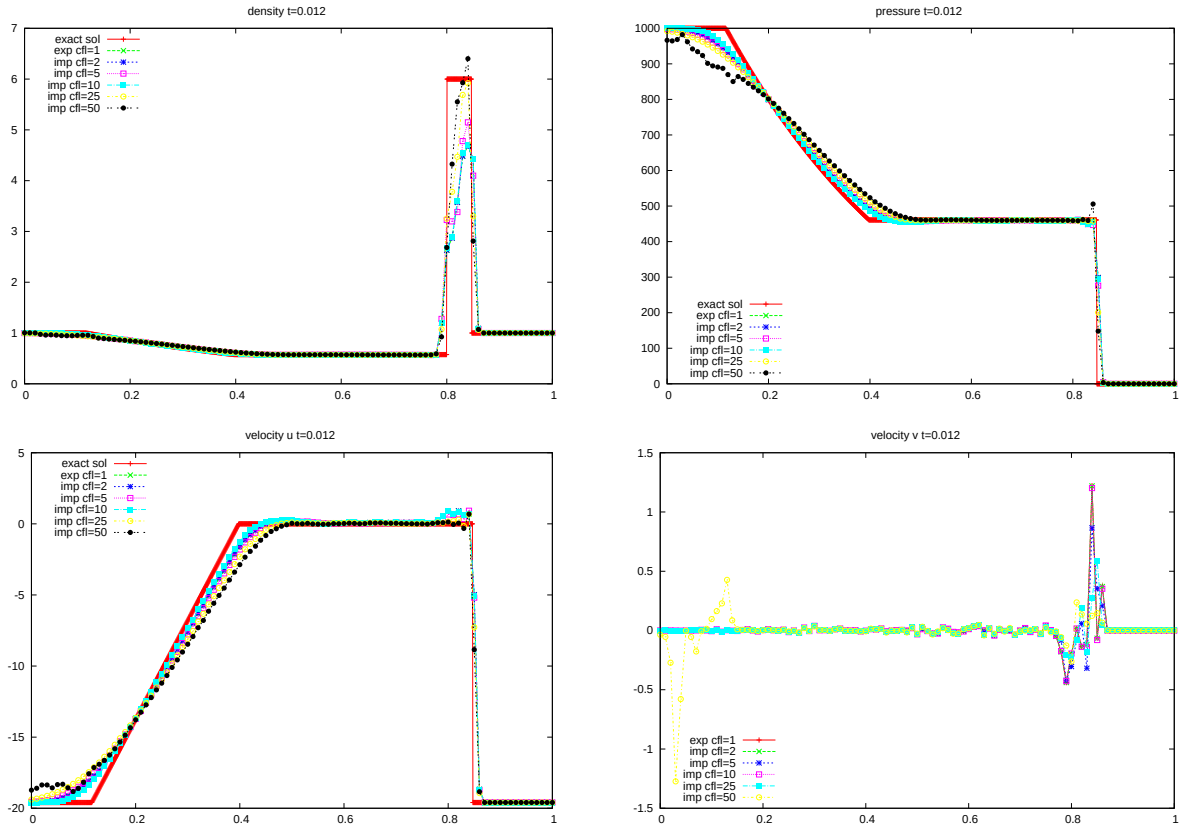


FIG. 31 – Test 6 : ρ, p, u et v à $t_f = 0.012$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [4]

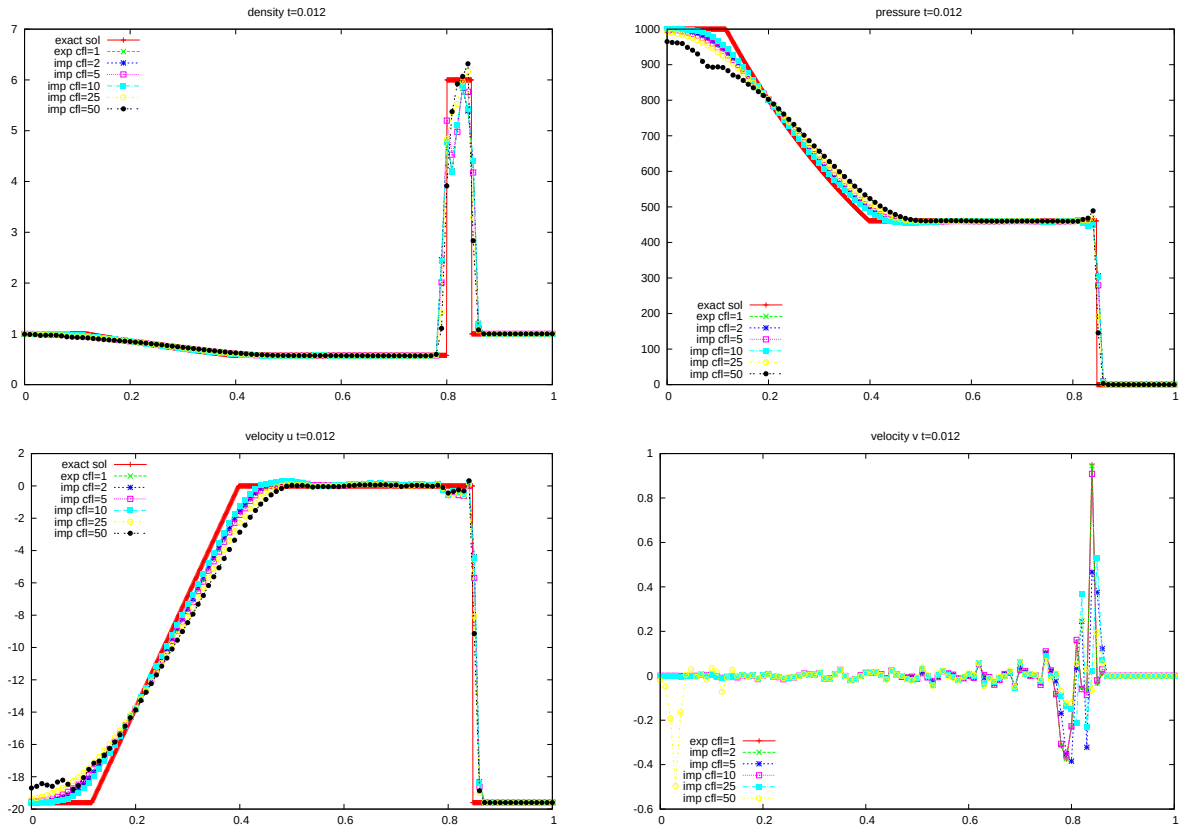


FIG. 32 – Test 6 : ρ, p, u et v à $t_f = 0.012$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [4]

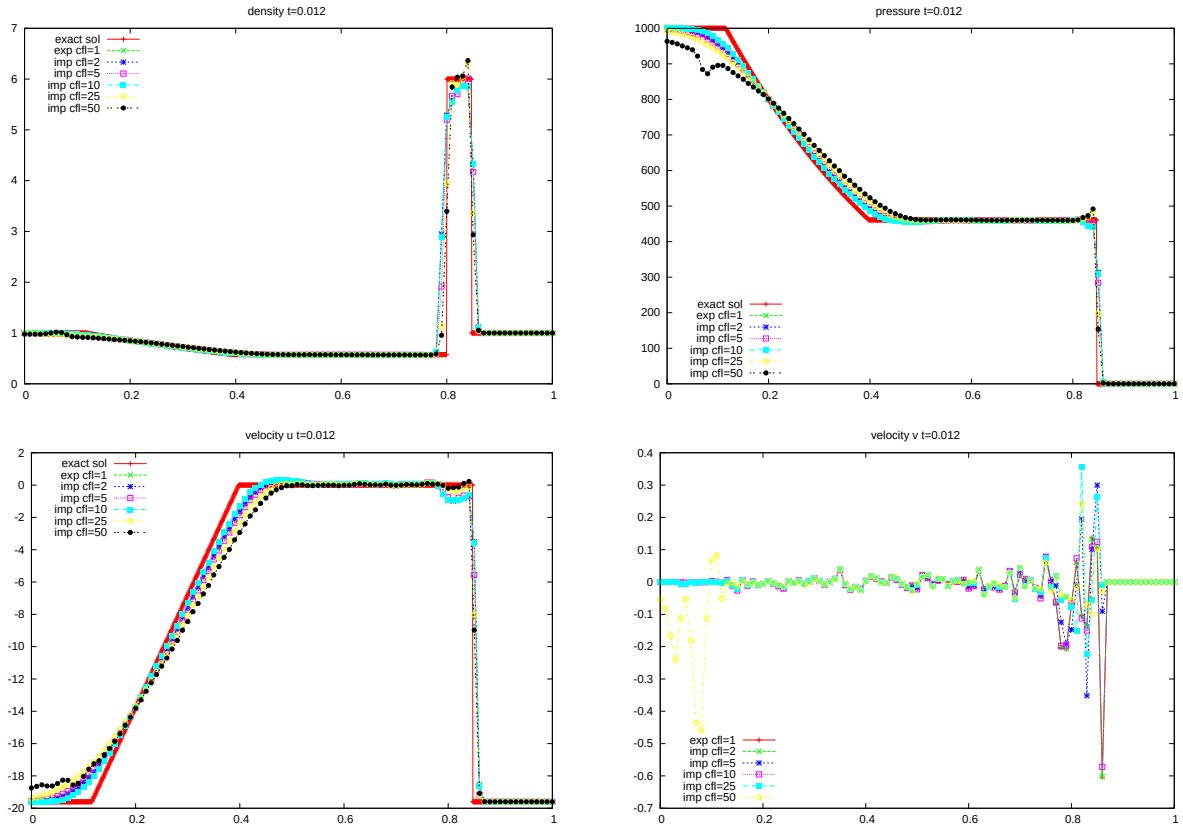


FIG. 33 – Test 6 : ρ, p, u et v à $t_f = 0.012$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [4]

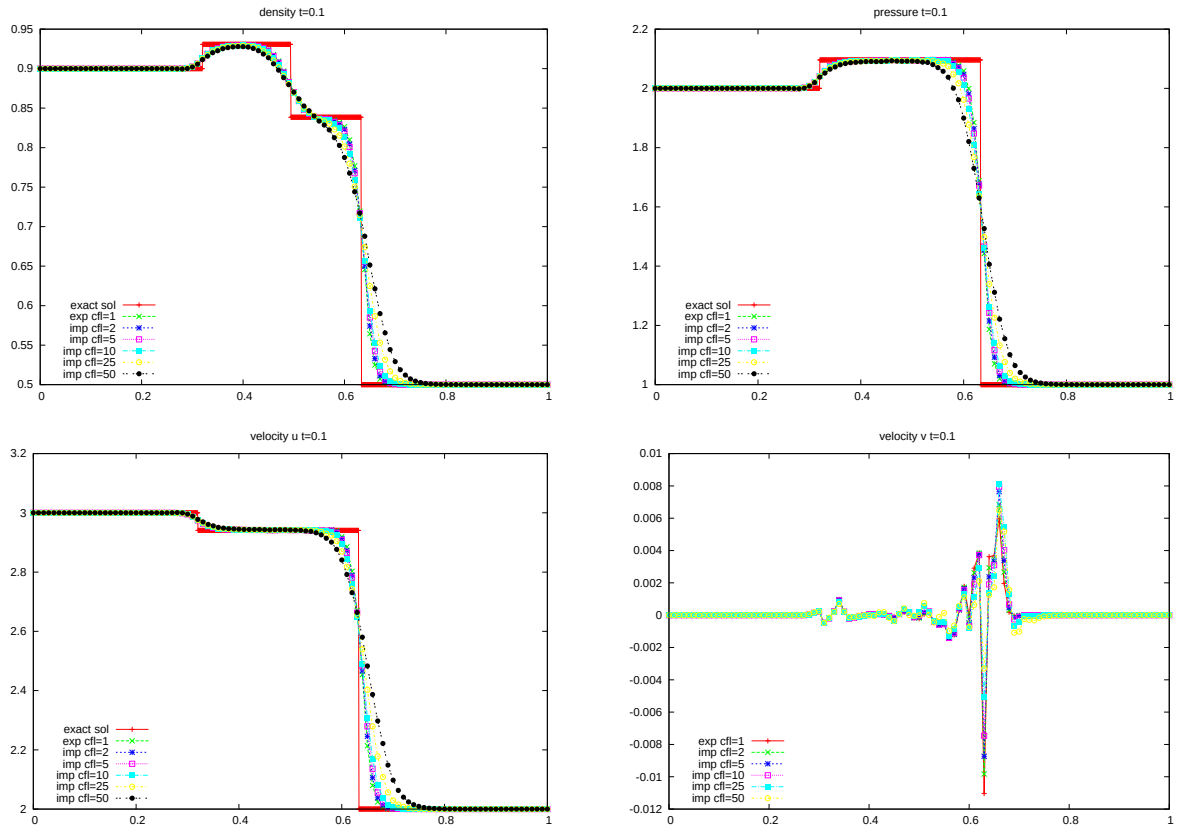


FIG. 34 – Test 7 : ρ, p, u et v à $t_f = 0.1$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [2]

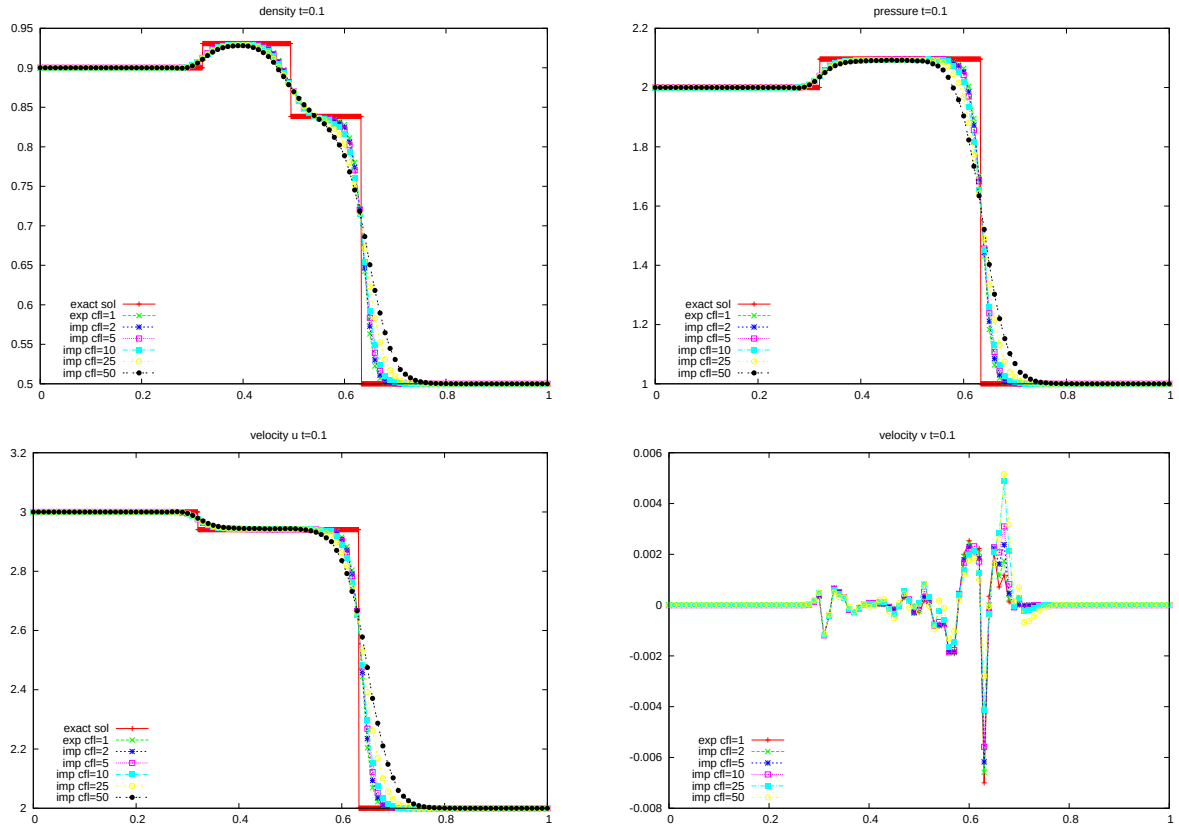


FIG. 35 – Test 7 : ρ, p, u et v à $t_f = 0.1$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [2]

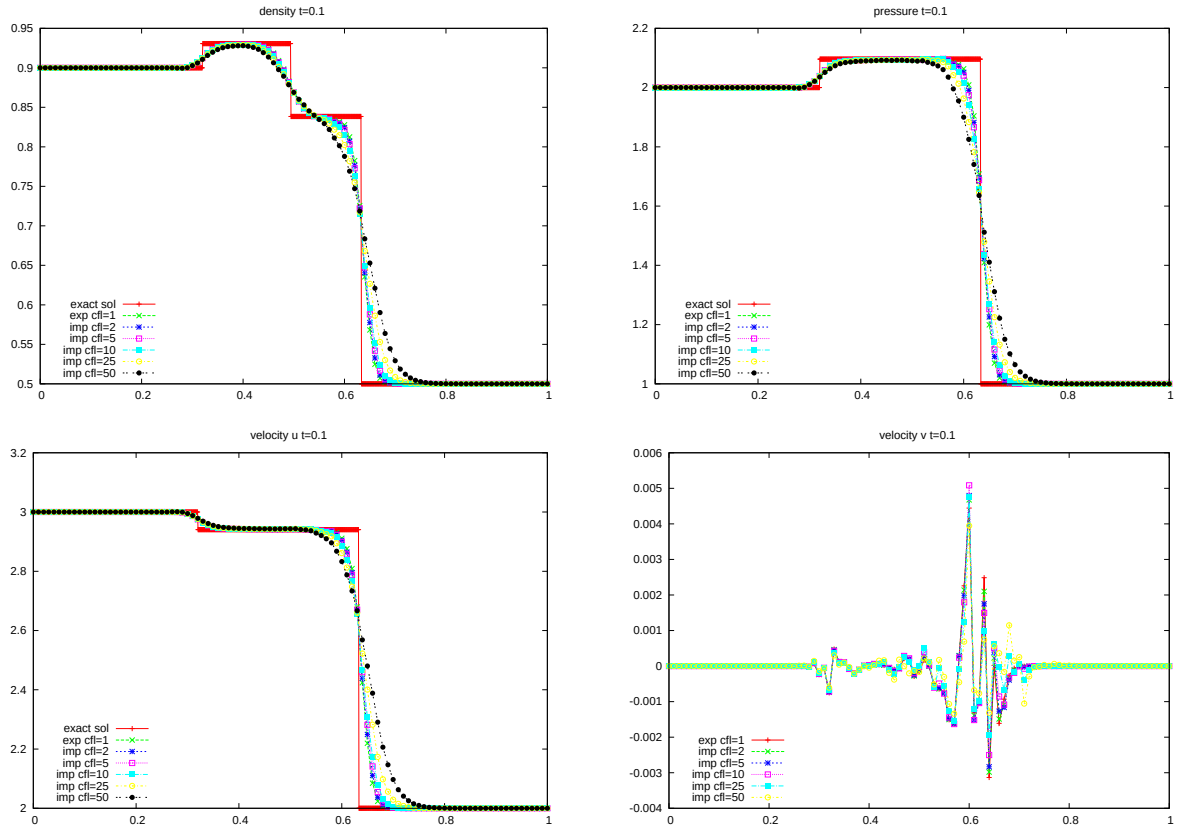


FIG. 36 – Test 7 : ρ, p, u et v à $t_f = 0.1$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [2]

8.1.8 Test 8

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.13$, (voir figures 37, 38 et 39) :

$$(\rho, u, v, p) = \begin{cases} (1, 1, 0, 2) & \text{si } x \leq 0.25 \\ (2, 2, 0, 2) & \text{si } x > 0.25 \end{cases}$$

8.1.9 Test 9

Pour un domaine $\mathcal{D} = [0, 1] \times [0, 0.002]$, on prend les conditions initiales suivantes pour $t_f = 0.035$, (voir figures 40, 41 et 42) :

$$(\rho, u, v, p) = \begin{cases} (1, 0, 0, 0.01) & \text{si } x \leq 0.5 \\ (1, 0, 0, 100) & \text{si } x > 0.5 \end{cases}$$

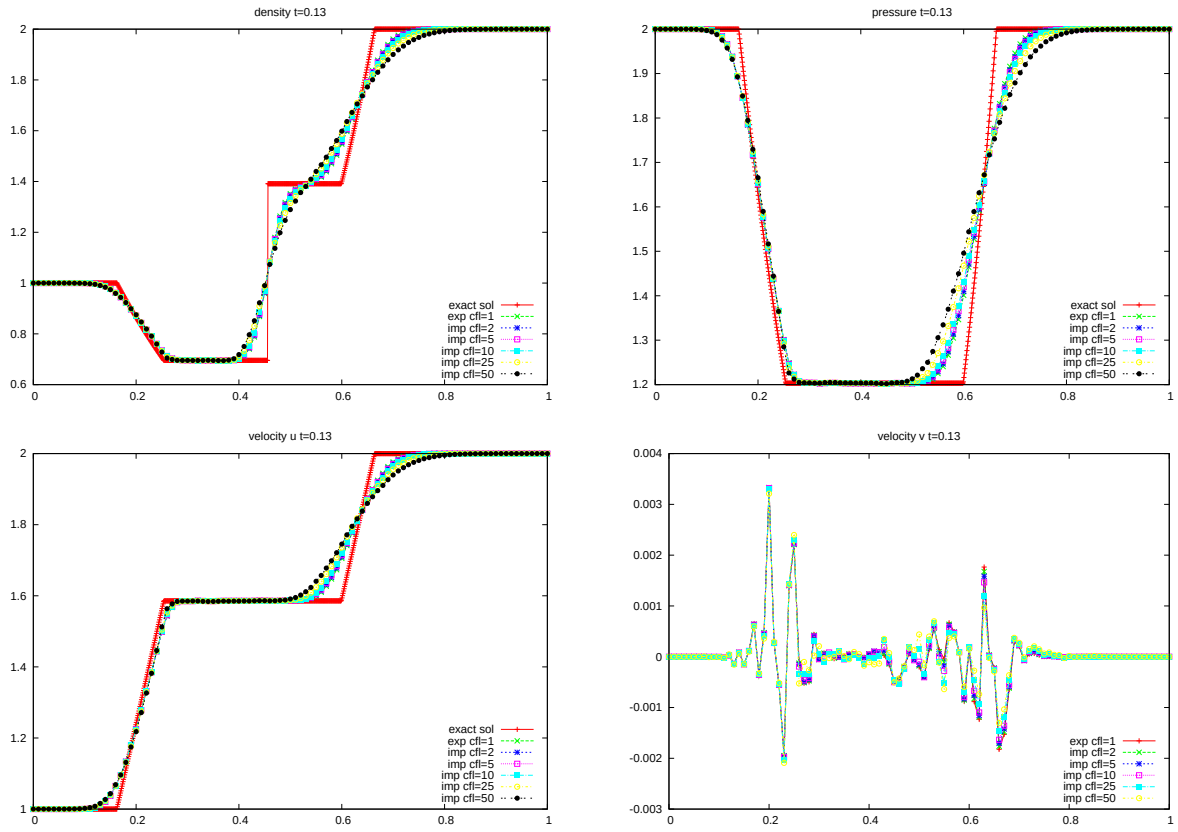


FIG. 37 – Test 8 : ρ, p, u et v à $t_f = 0.13$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [2]

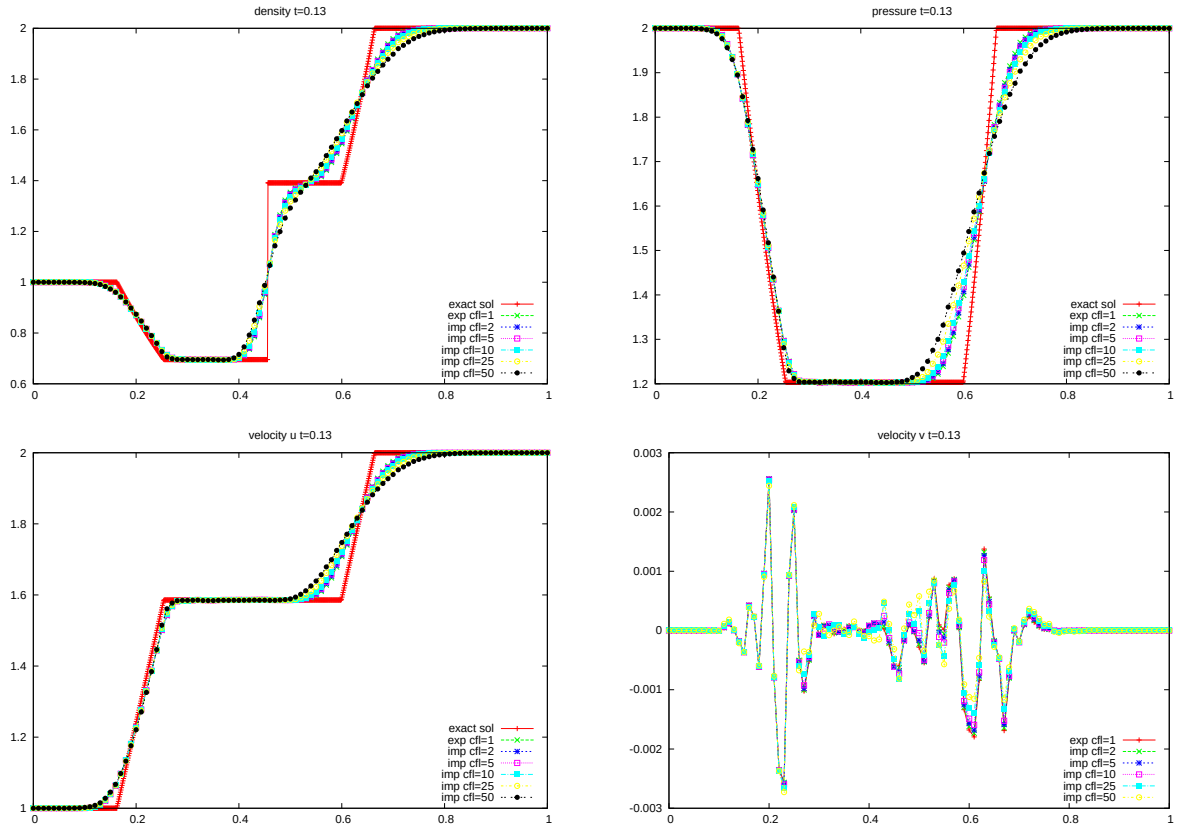


FIG. 38 – Test 8 : ρ, p, u et v à $t_f = 0.13$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [2]

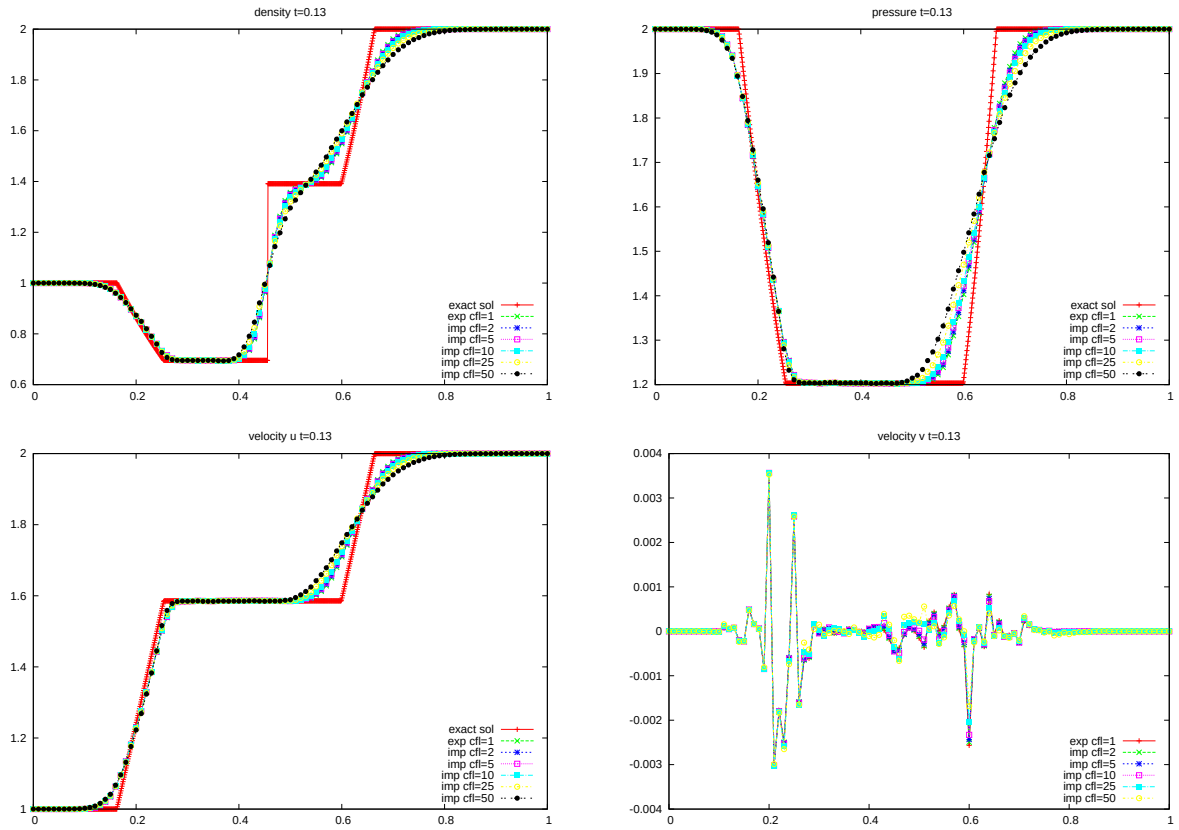


FIG. 39 – Test 8 : ρ, p, u et v à $t_f = 0.13$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [2]

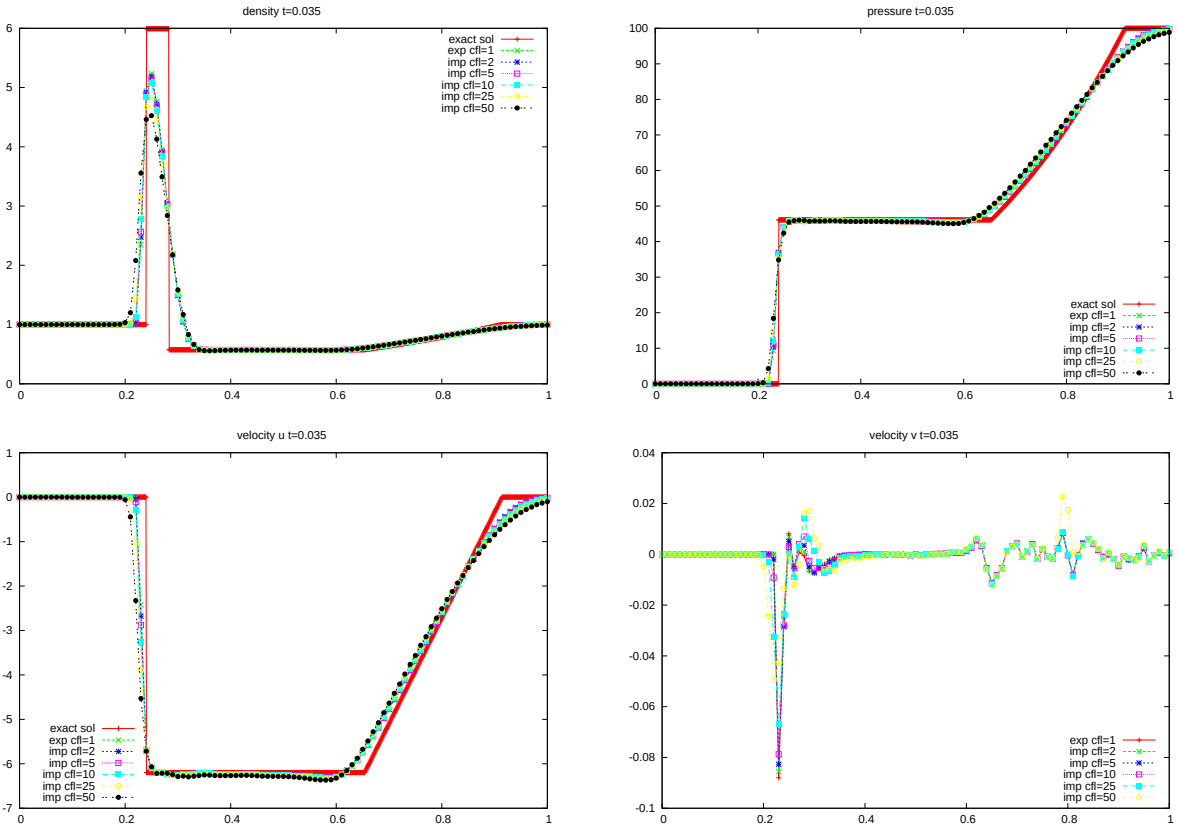


FIG. 40 – Test 9 : ρ, p, u et v à $t_f = 0.035$ pour $y = 0.005$ et $x \in [0, 1]$, réalisé dans [4]

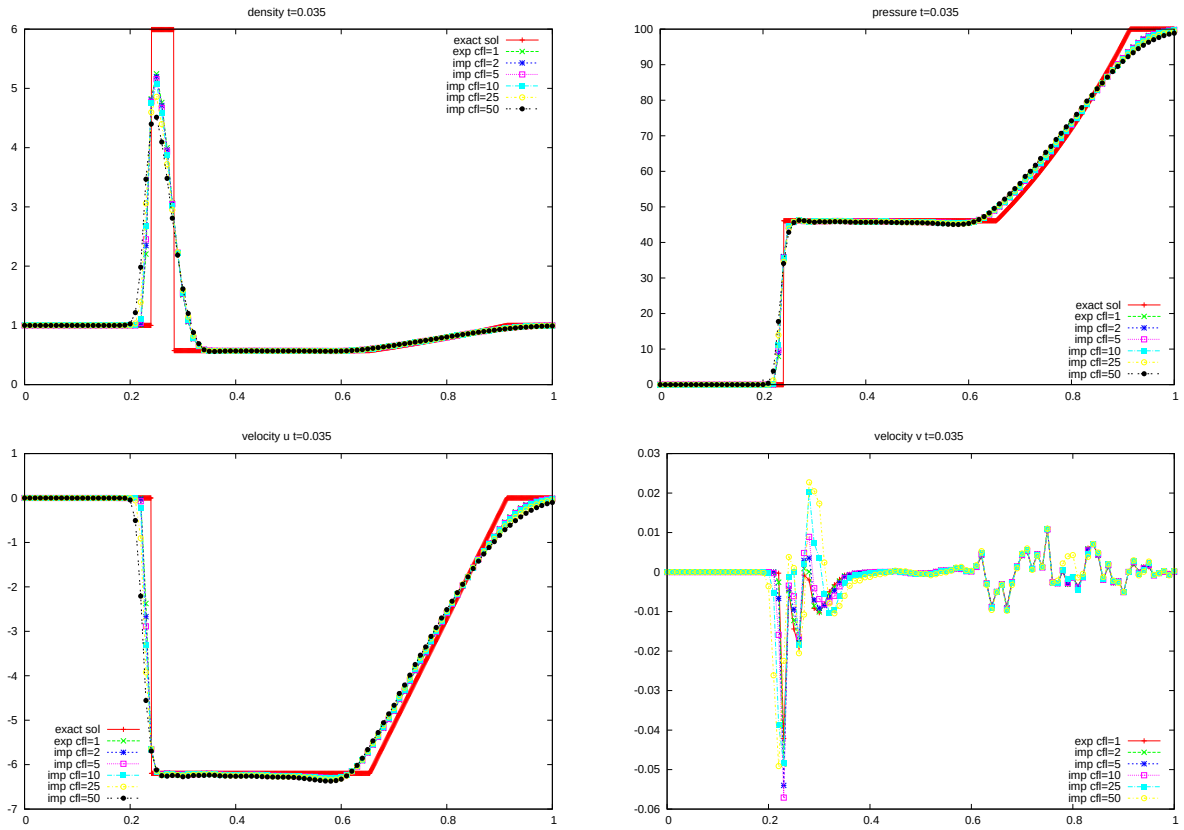


FIG. 41 – Test 9 : ρ, p, u et v à $t_f = 0.035$ pour $y = 0.01$ et $x \in [0, 1]$, réalisé dans [4]

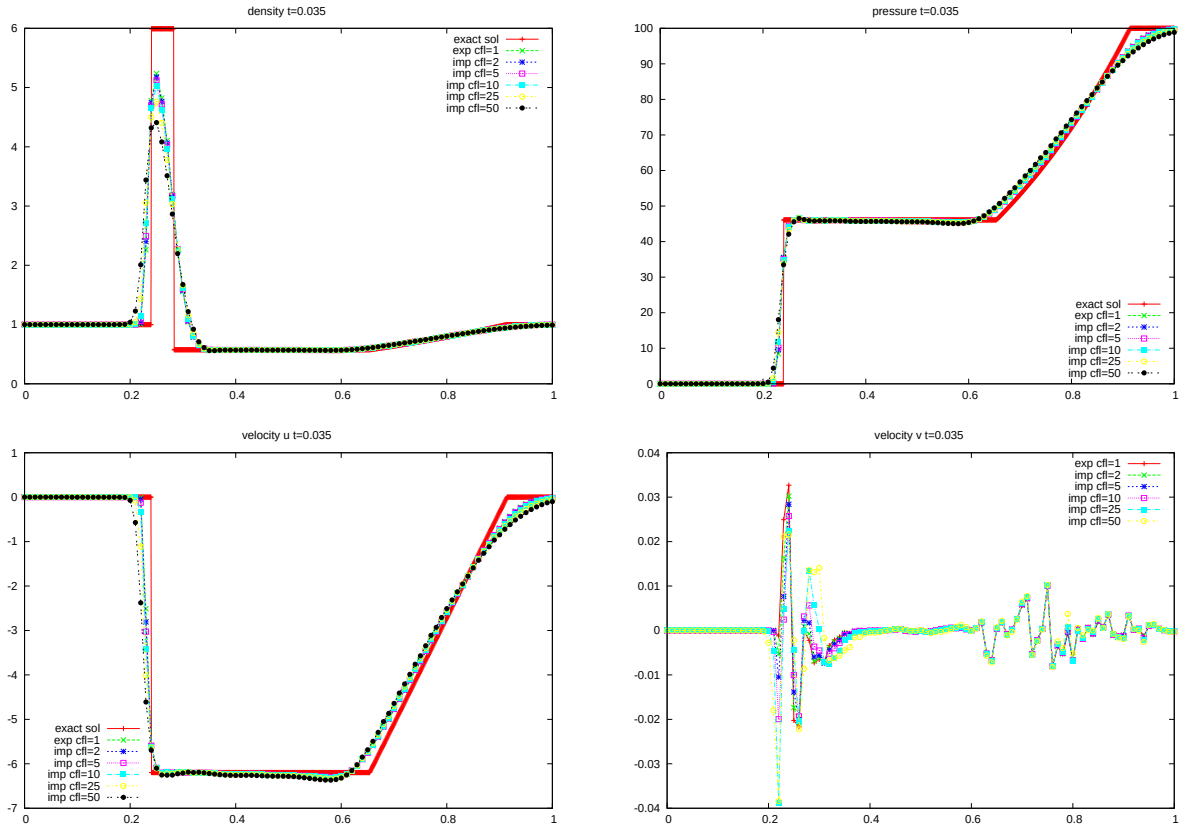


FIG. 42 – Test 9 : ρ, p, u et v à $t_f = 0.035$ pour $y = 0.015$ et $x \in [0, 1]$, réalisé dans [4]

8.2 Tests 2D

8.2.1 Test 1

Pour un domaine carré $\mathcal{D} = [0, 2] \times [0, 2]$, on prend les conditions initiales suivantes pour $t_f = 0.15$:

$$(\rho, u, v, p) = \begin{cases} (1, 0, 0, 1) & \text{si } r \leq 0.5 \\ (0.125, 0, 0, 0.1) & \text{si } r > 0.5 \\ r & \text{le rayon du cercle de centre } (1, 1) \end{cases}$$

Le premier maillage utilisé pour ce test contient 20402 mailles, 30805 arêtes et 10408 sommets (voir figures 43, 45, 46, 47 et 48). Le second maillage contient 80802 mailles, 121605 arêtes et 40808 sommets (voir figures 44, 49, 50, 51 et 52).

9 Conclusion

Pour arriver au lancement de l'executable issu du projet, il a fallu franchir plusieurs grandes étapes comme la gestion du maillage, la construction des structures en C, le développement en C proprement dit du solveur, l'initialisation de la résolution du problème à partir d'un fichier et enfin la visualisation des résultats du code. La suite du travail a été consacrée à la validation du code fraîchement élaboré sur des cas tests connus. Le temps utilisé à coder le solveur a été court tandis que certaines tâches comme la récupération des données du maillage et le post-traitement qui suit ou la phase de validation du code ont demandé bien plus de patience et de temps. Il a donc fallu déjouer les bugs qui parfois découlent d'un problème de syntaxe. Mais ils peuvent aussi être le fruit d'une erreur de raisonnement. Dans le premier cas, des bonnes habitudes de rédaction ou des outils comme Valgrind ou gdb sont alors d'un grand secours. Dans la seconde éventualité, le programmeur doit éplucher les étapes de son calcul et s'assurer ligne par ligne qu'il ne s'écarte pas du « bon » chemin. C'est sans doute l'exercice le plus difficile sur l'ensemble du travail à fournir. L'expérience acquise pendant ce stage permet de mesurer l'inertie qu'il existe entre l'idée émise et sa mise en application effective.

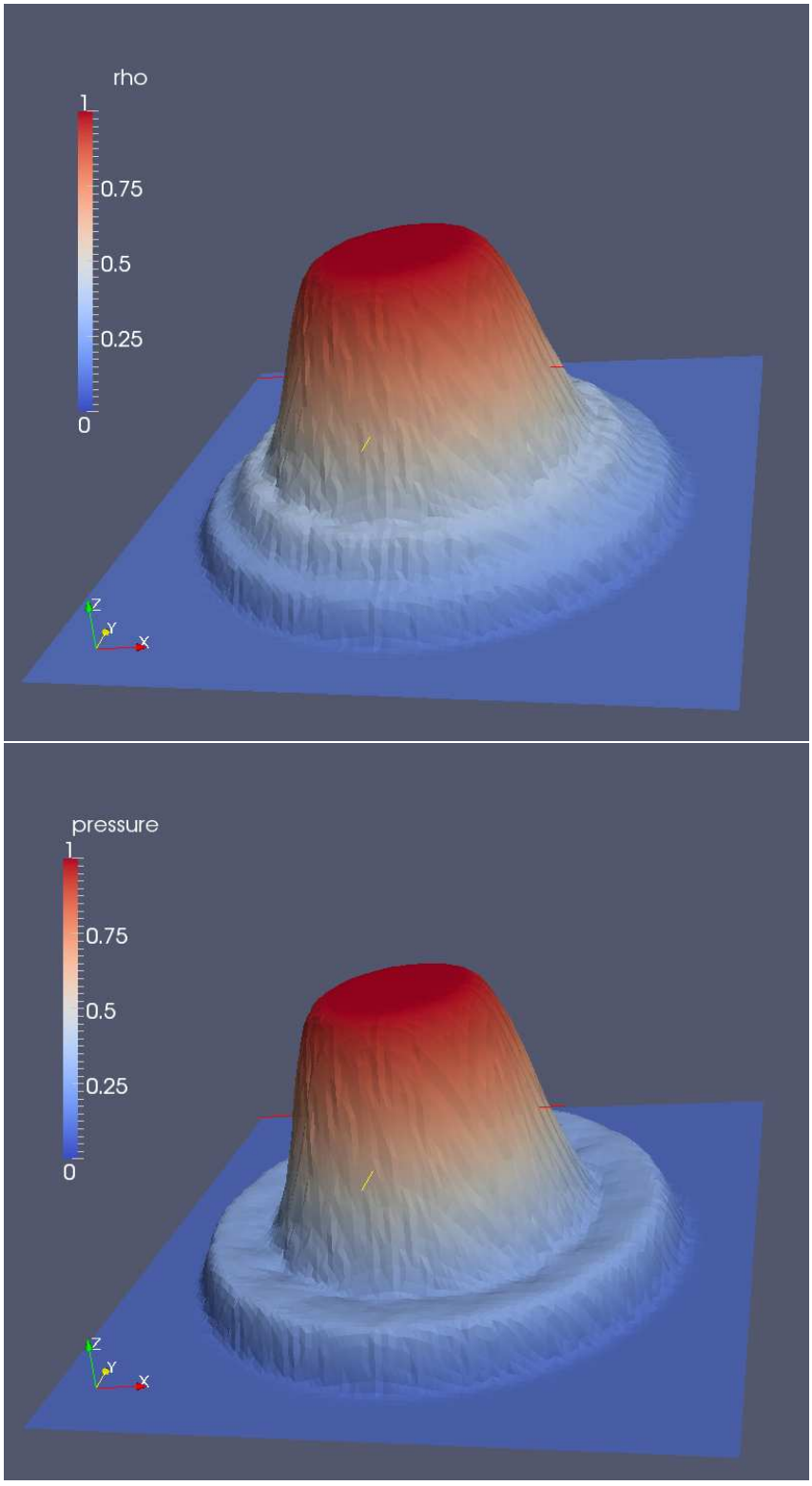


FIG. 43 – Test 1 : ρ et p à $t_f = 0.15$ en explicite avec 20402 mailles

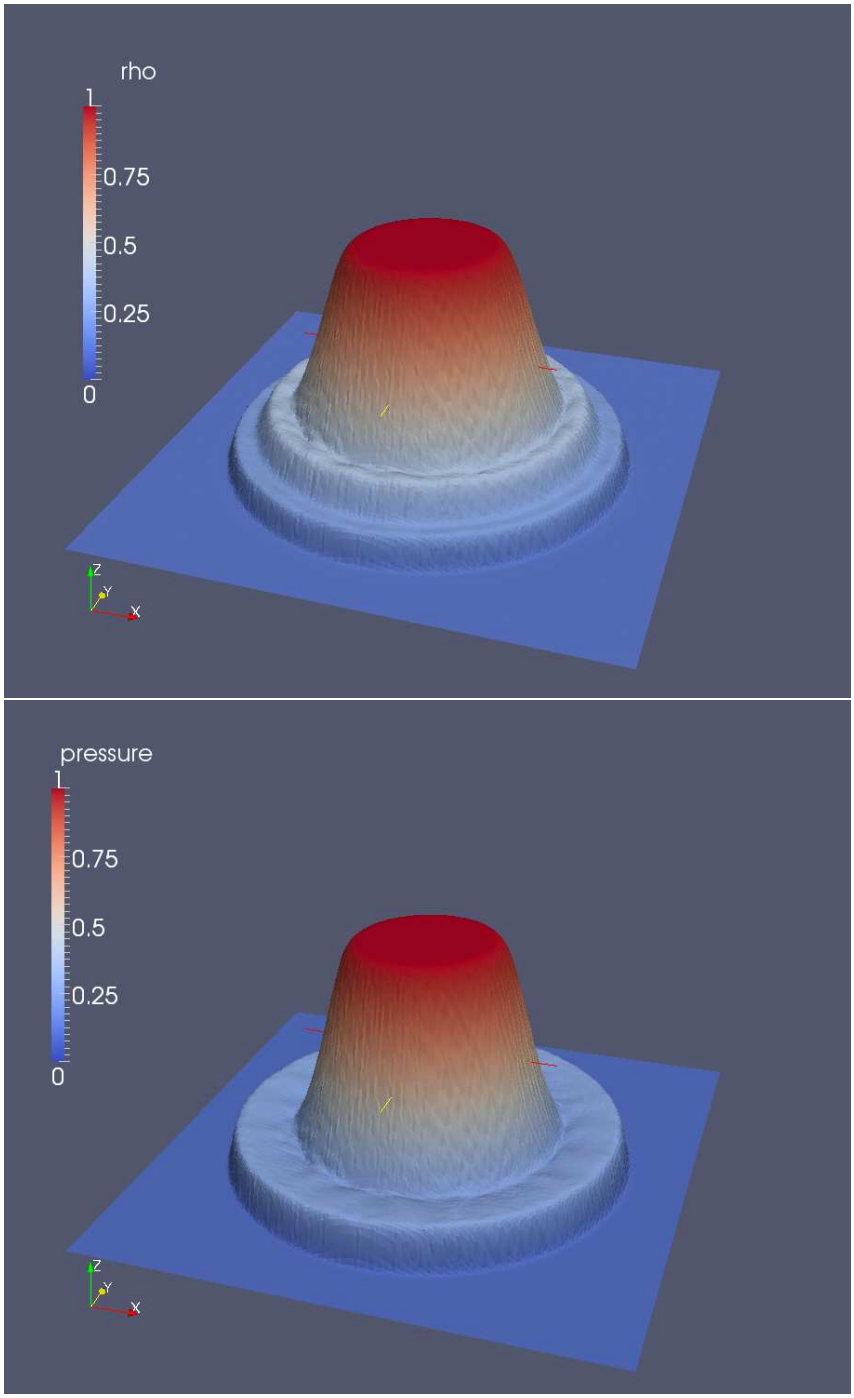


FIG. 44 – Test 1 : ρ et p à $t_f = 0.15$ en explicite avec 80802 mailles

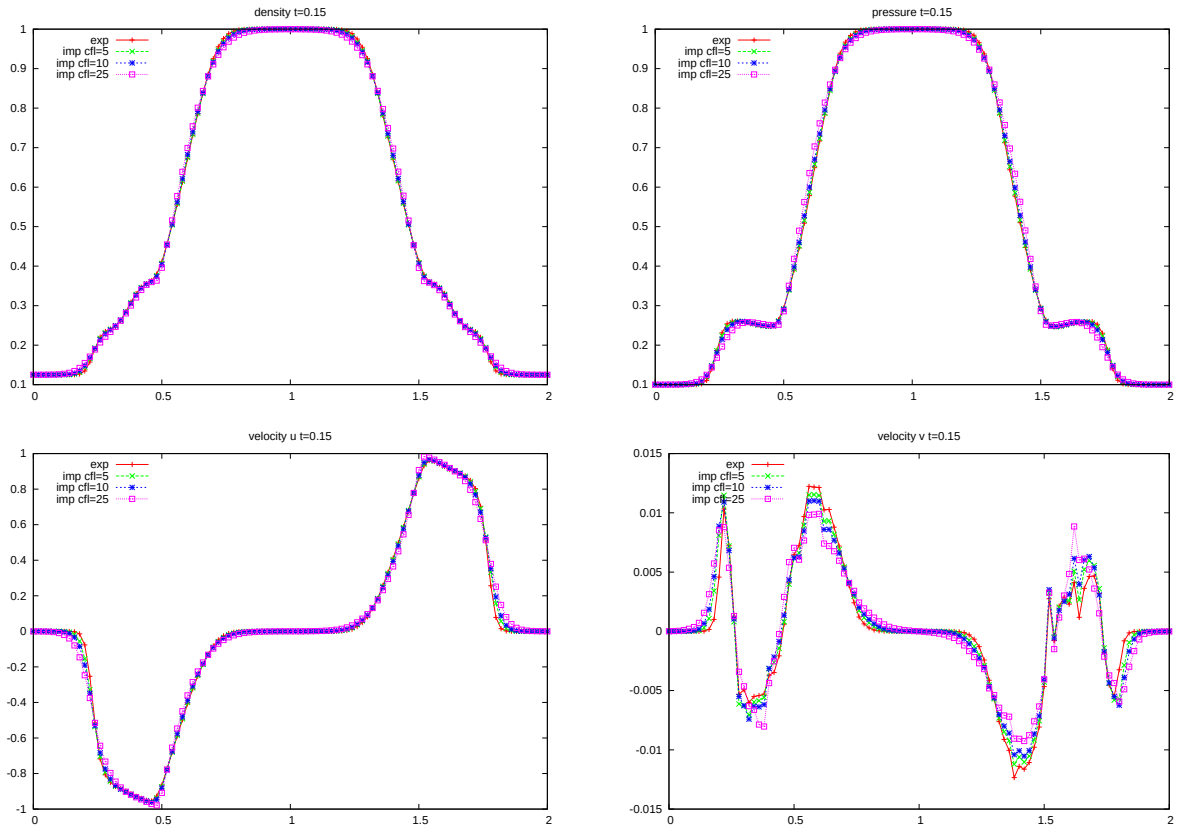


FIG. 45 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = 1$ et $x \in [0, 2]$ avec 20402 mailles

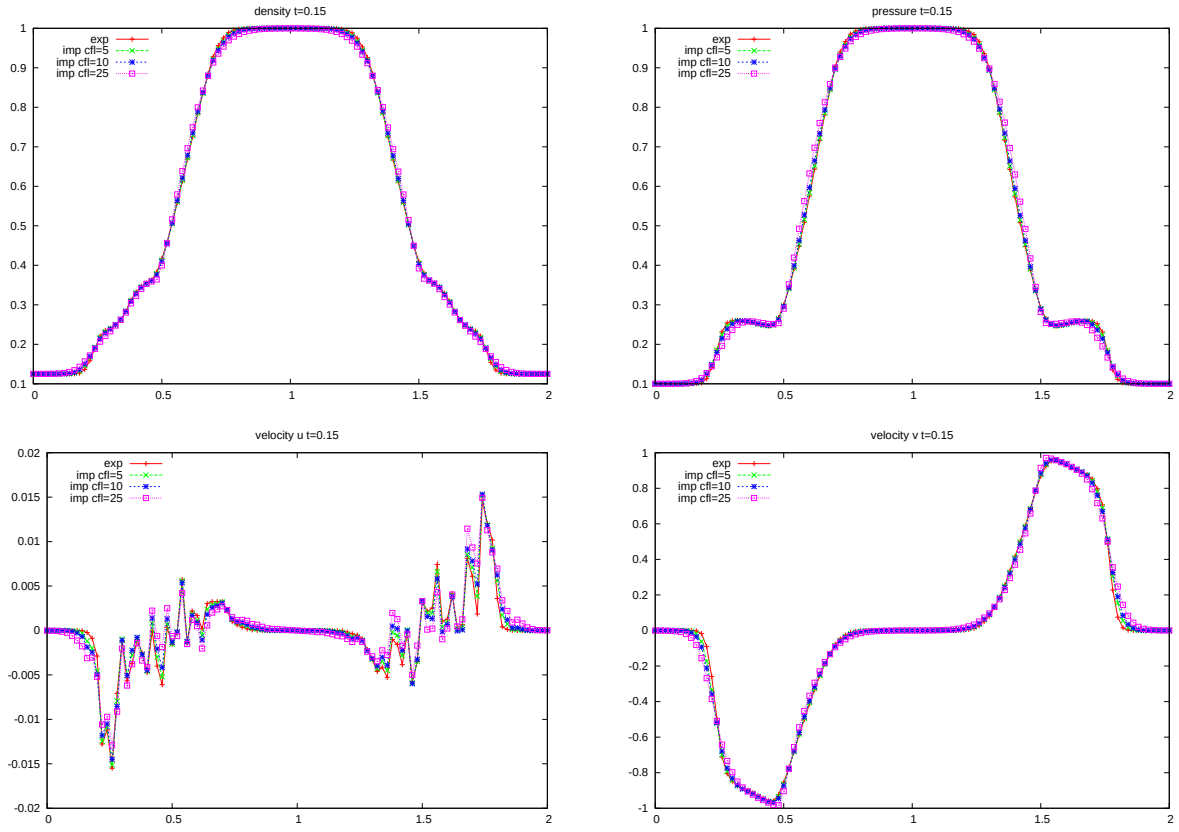


FIG. 46 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $x = 1$ et $y \in [0, 2]$ avec 20402 mailles

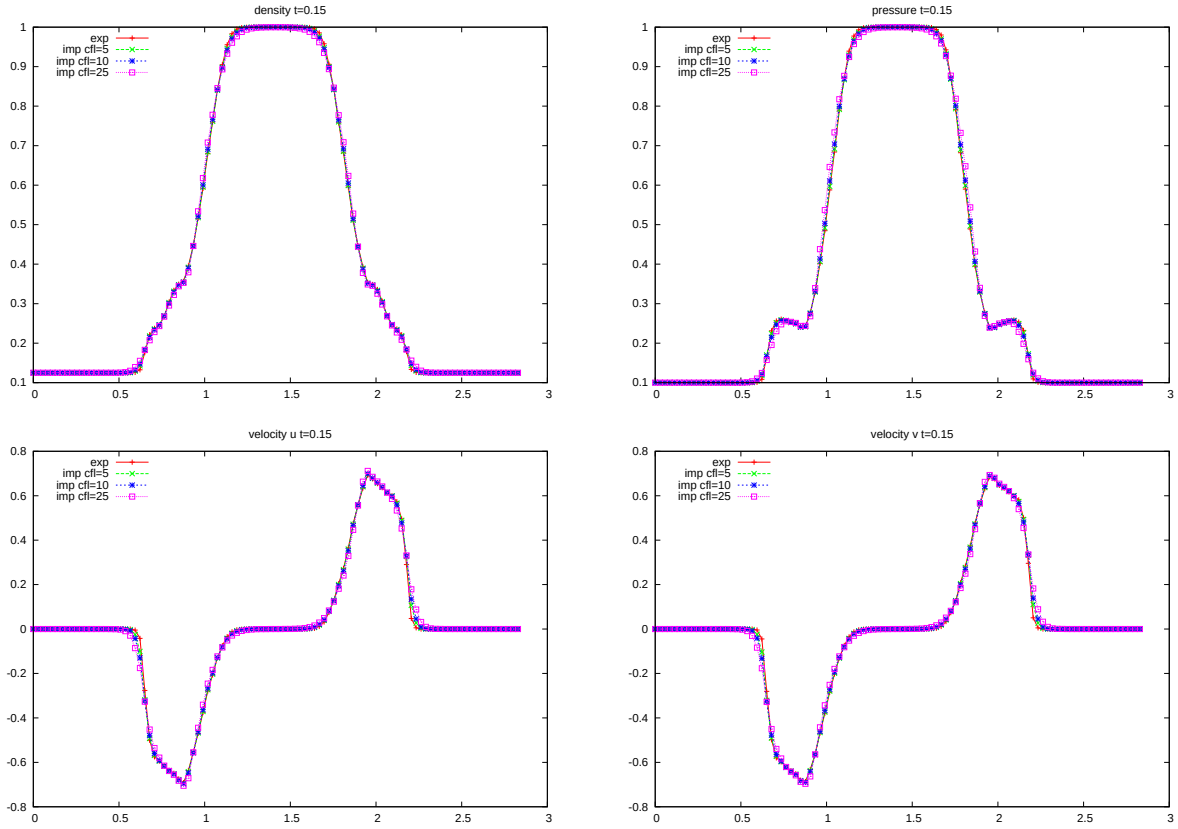


FIG. 47 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = x$ et $x \in [0, 2]$ (la première diagonale) avec 20402 mailles

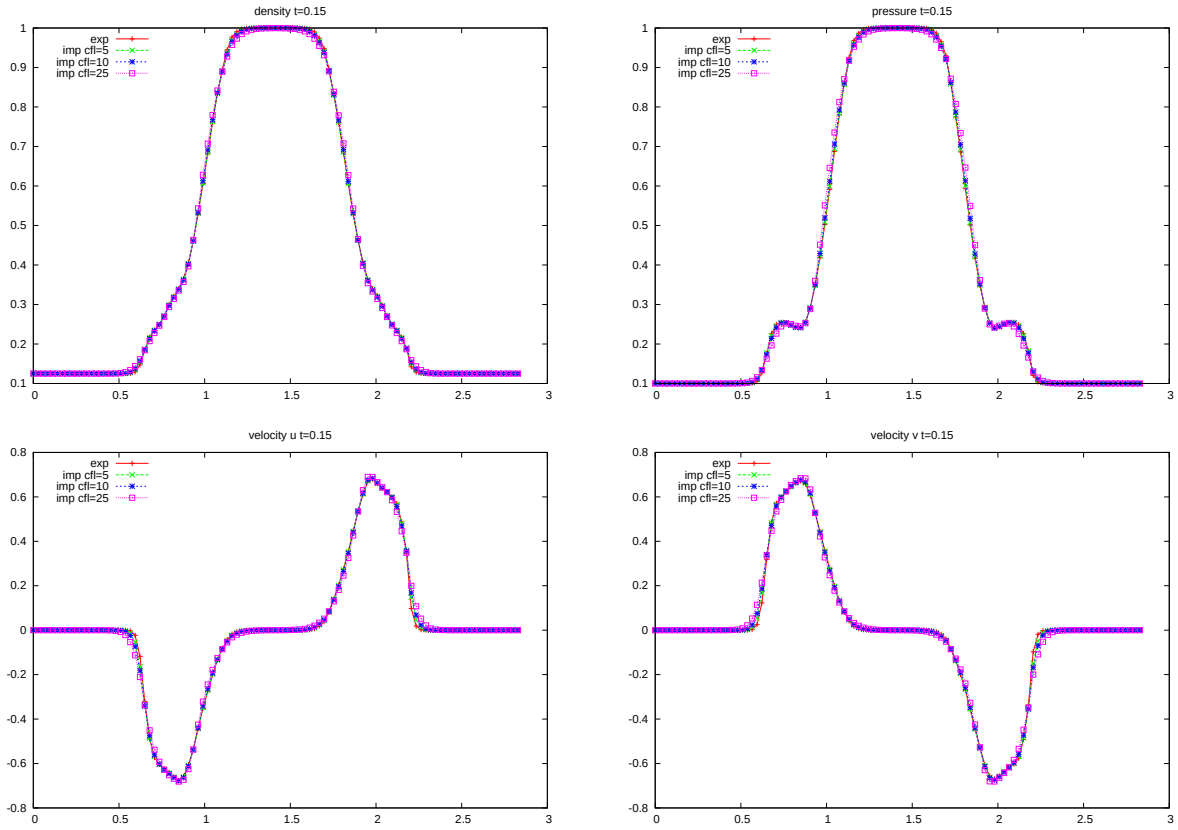


FIG. 48 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = -x + 2$ et $x \in [0, 2]$ (la seconde diagonale) avec 20402 mailles

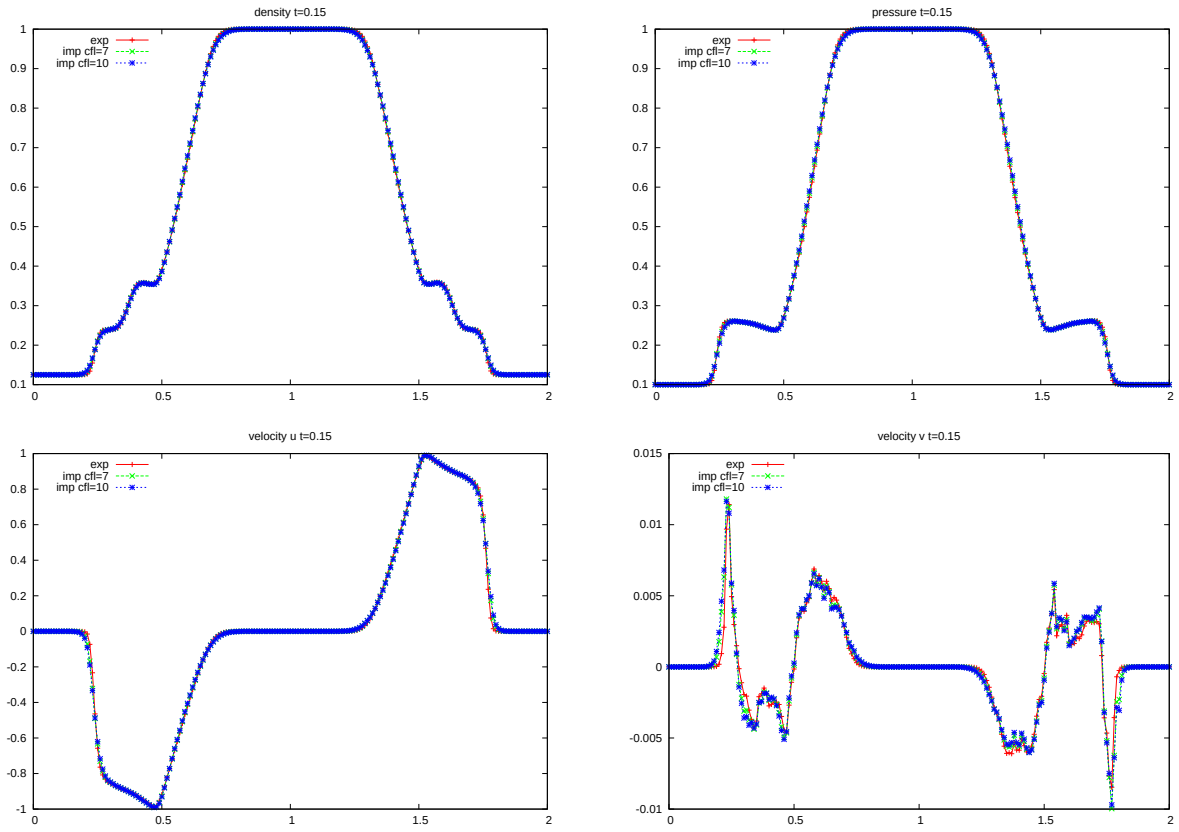


FIG. 49 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = 1$ et $x \in [0, 2]$ avec 80802 mailles

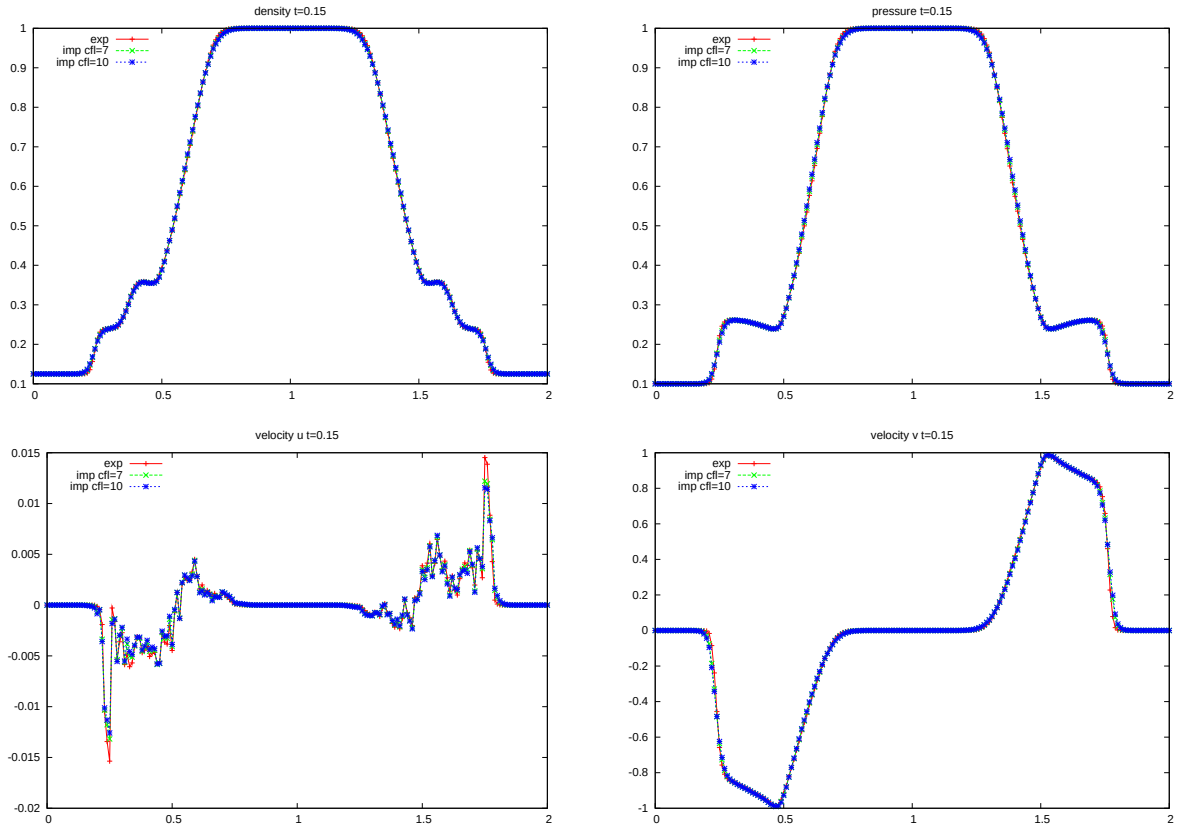


FIG. 50 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $x = 1$ et $y \in [0, 2]$ avec 80802 mailles

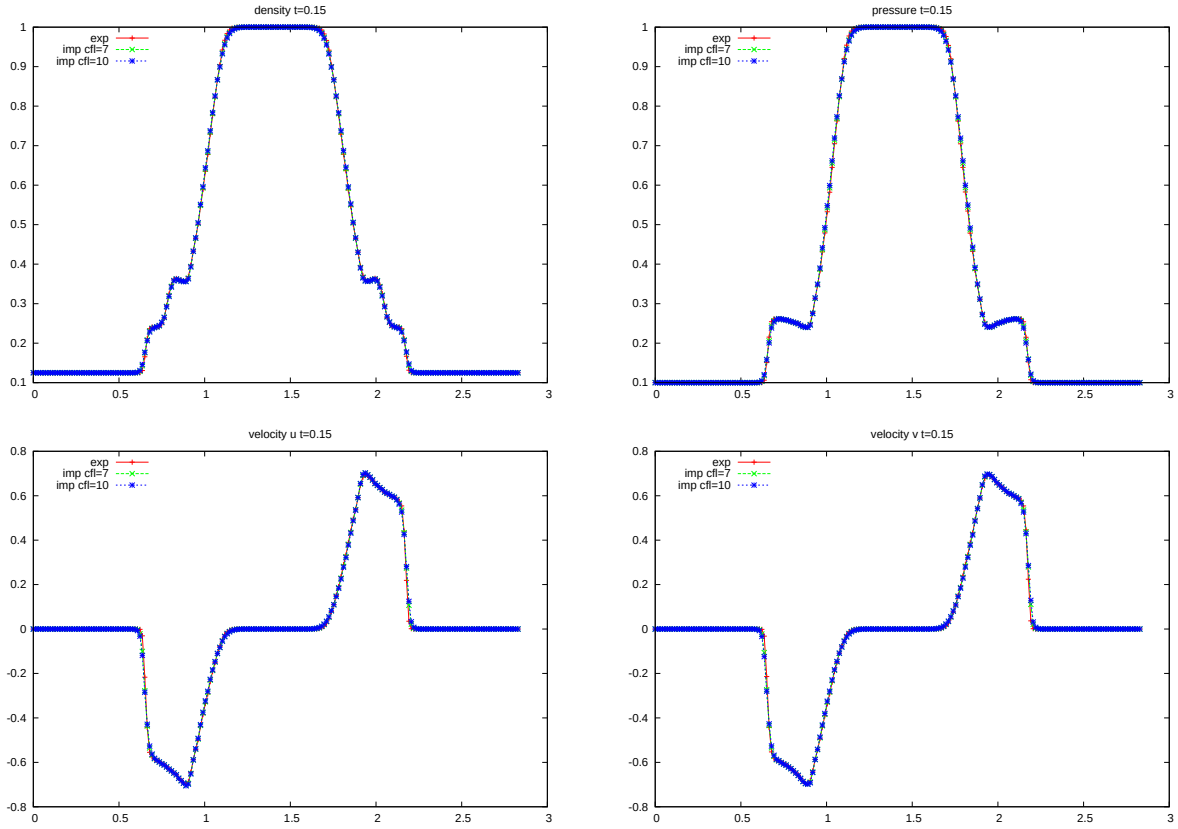


FIG. 51 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = x$ et $x \in [0, 2]$ (la première diagonale) avec 80802 mailles

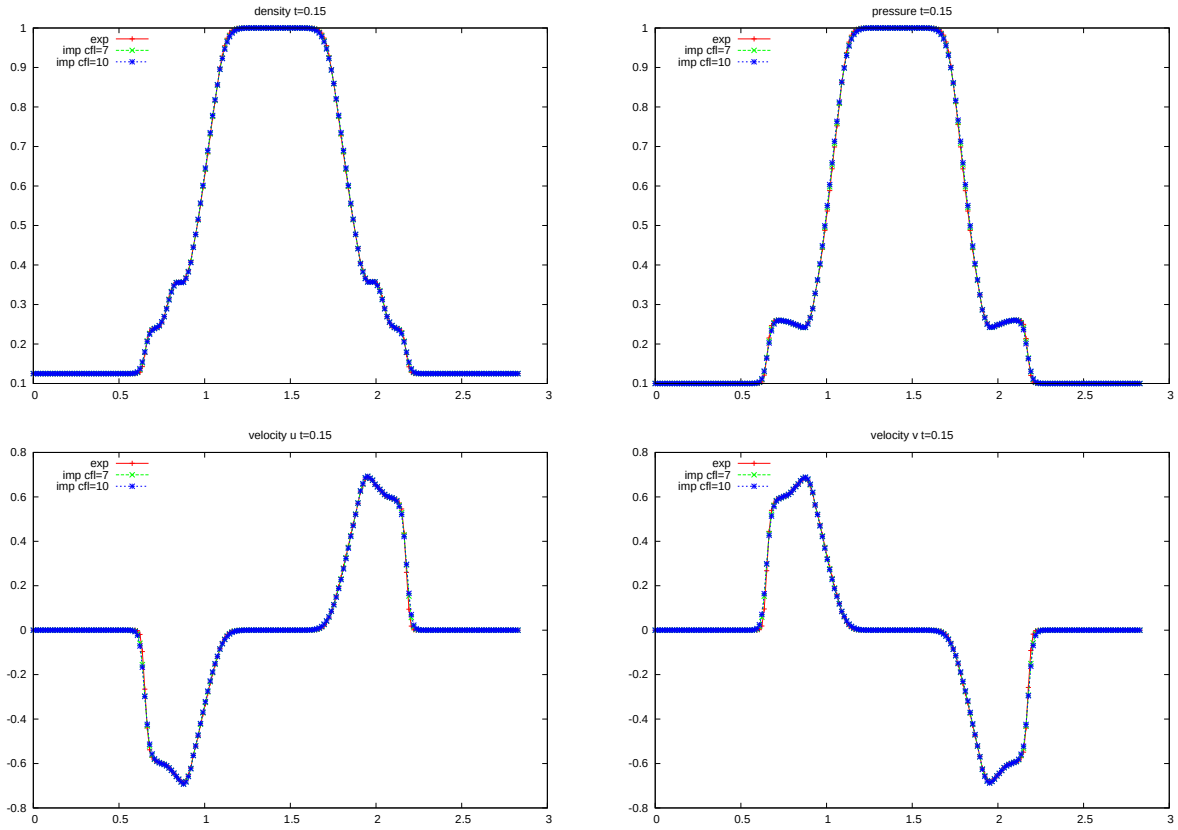


FIG. 52 – Test 1 : ρ, p, u et v à $t_f = 0.15$ pour $y = -x + 2$ et $x \in [0, 2]$ (la seconde diagonale) avec 80802 mailles

A Calculs liés aux triangles

A.1 Calcul de l'aire d'un triangle

La manière classique pour calculer l'aire d'un triangle est de multiplier la longueur de sa base par celle de sa hauteur et de la diviser par deux. Mais si l'on connaît uniquement les coordonnées de ses sommets, on peut directement retrouver ce résultat à l'aide du calcul d'un déterminant. Soit ABC un triangle quelconque et $\mathcal{A}$ son aire. On désigne par x_A l'abscisse de A et y_A l'ordonnée de A. $\vec{u} \wedge \vec{v}$ représente le produit vectoriel de $\vec{u}$ par $\vec{v}$ et $||\vec{u}||$ la norme de $\vec{u}$.

$$\begin{aligned}
\mathcal{A} &= \frac{1}{2}||\vec{AB} \wedge \vec{AC}|| \\
&= \frac{1}{2}\left|\det\begin{pmatrix} x_B - x_A & x_C - x_A \\ y_B - y_A & y_C - y_A \end{pmatrix}\right| \\
&= \frac{1}{2}|x_B y_C - y_B x_C - x_A y_C + x_C y_A + x_A y_B - x_B y_A|
\end{aligned}$$

A.2 Centre de gravité d'un triangle

Une manière de caractériser le centre de gravité G d'un triangle ABC est d'utiliser l'égalité vectorielle

$$\vec{GA} + \vec{GB} + \vec{GC} = \vec{0}$$

De cette égalité découle les deux égalités scalaires suivantes :

$$\begin{aligned}
x_A - x_G + x_B - x_G + x_C - x_G &= 0 \\
y_A - y_G + y_B - y_G + y_C - y_G &= 0
\end{aligned}$$

On a donc :

$$\begin{aligned}
x_G &= \tfrac{1}{3}(x_A + x_B + x_C) \\
y_G &= \tfrac{1}{3}(y_A + y_B + y_C)
\end{aligned}$$

A.3 Calcul de l'intégrale d'une surface

Dans l'initialisation du code développé ici, on est amené à évaluer des intégrales sur des triangles. Pour le triangle ABC d'aire $\mathcal{A}$, on approche une intégrale de surface de la manière suivante :

$$\int_{ABC} f(x,y)dxdy \approx \frac{\mathcal{A}}{3}\left[f(x_A,y_A) + f(x_B,y_B) + f(x_C,y_C)\right]$$

Cette formule d'approximation est exacte pour les polynômes de degré inférieur ou égal à 1.

B Détermination de l'orientation d'une normale

Il nous faut à une étape du code établir l'orientation des normales aux arêtes pour chaque cellule du maillage. A ce moment précis, nous ne connaissons que les vecteurs normaux et les triangles et par conséquent leurs arêtes. Plaçons-nous dans la situation d'un triangle qu'on appelle K. Soient e l'une de ses arêtes, $\vec{n}_e$ la normale à e, $\vec{n}_{e,K}$ la normale à e orientée par rapport à K (*ie* sortante de K), P le sommet opposé à e et I le

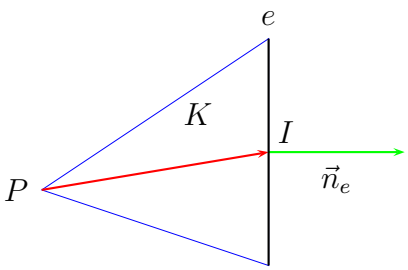


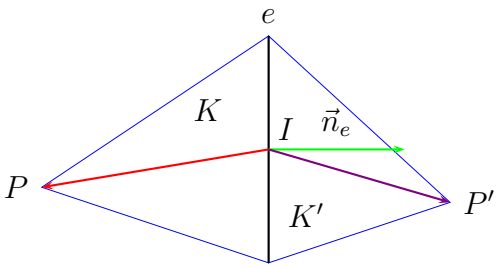
FIG. 53 – Orientation d'une normale

milieu de e. Voir figure 53.
La procédure pour évaluer l'orientation est la suivante : on réalise le produit scalaire entre la normale et le vecteur $\vec{PI}$ et on récupère son signe. On a alors $\vec{n}_{e,K} = \mathsf{sign}(\langle \vec{n}_e, \vec{PI} \rangle)\vec{n}_e$

C Position des triangles partageant la même arête

Une autre étape du code exige de connaître le positionnement des cellules W (Ouest) et E (Est) pour chaque arête du maillage. Nous ne connaissons alors que les deux triangles et l'arête qu'ils ont en commun. Plaçons-nous dans la situation d'une arête qu'on appelle e. Soient K et K' les deux triangles situés de part et d'autre de e. P et P' sont respectivement les sommets opposés à e et appartenant à K et K'. Par défaut, la normale $\vec{n}_e$ à e a une orientation qui va de l'Ouest vers l'Est.

Cas où l'arête n'est pas horizontale.
En se basant sur un produit scalaire du même type que le paragraphe précédent, on peut donc situer le positionnement de K. En effet, si $(\langle \vec{n}_e, \vec{IP} \rangle < 0)$, alors $\{K = W$ et $K' = E\}$ sinon $\{K = E$ et $K' = W\}$.
Attention, on pourrait croire qu'on peut simplifier le critère d'identification en se conten-



$\langle \vec{n}_e, \vec{IP} \rangle < 0, K = W$ et $\langle \vec{n}_e, \vec{IP'} \rangle > 0, K' = E$

FIG. 54 – Identification des cellules W et E par rapport à e

tant de comparer l'abscisse de P avec celle de P' pour déterminer qui sont E et W : si $(x_P < x_{P'})$ alors $\{K = W$ et $K' = E\}$ sinon $\{K = E$ et $K' = W\}$. Ce test semble marcher si on se base sur la figure 54 mais des contre-exemples existent comme l'exemple

de la figure 55. De la même manière effectuer ce test en se fiant non pas à P et P' mais aux centres de gravité de K et K' n'est pas une bonne solution non plus.

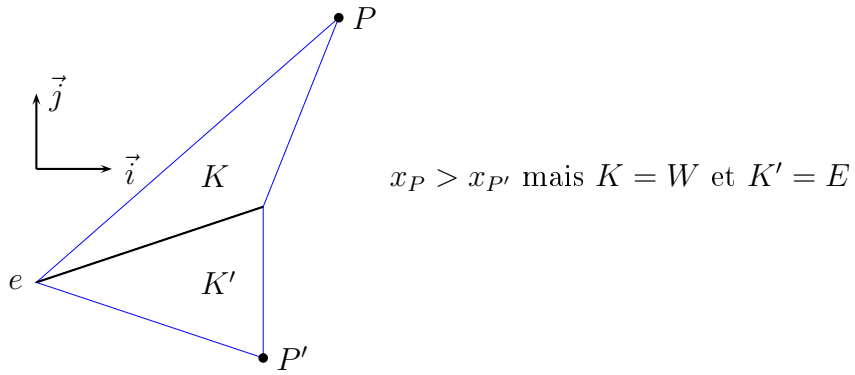


FIG. 55 – Identification des cellules W et E : contre-exemple

Cas où l'arête est horizontale.

Dans ce cas particulier, la normale non orientée $\vec{n}_e$ est toujours calculée ascendante par rapport à $\vec{j}$, allant toujours de l'Ouest vers l'Est. Le critère le plus naturel est donc : si $(y_{P'} > y_P)$ alors $\{K = W \text{ et } K' = E\}$ sinon $\{K = E \text{ et } K' = W\}$.

D Utilisation de PETSc

D.1 Création d'une matrice

On peut créer une matrice de différentes façons avec PETSc. On va évoquer ici deux possibilités dans le cadre des matrices creuses.

Cas général.

Dans le cas où on ne connaît pas la structure de la matrice (*ie* on ne sait pas où sont

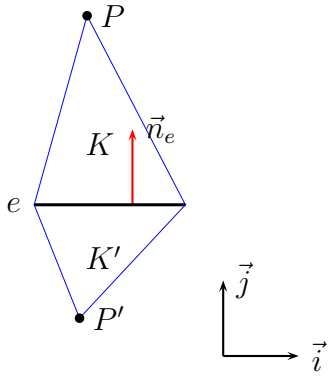


FIG. 56 – Identification des cellules W et E pour une arête horizontale

placés les éléments non nuls de la matrice), une manière de déclarer une matrice est la suivante :

```
Mat A ;
int matrixSize=1000 ;
int blockSize=5 ;
MatCreate(PETSC_COMM_WORLD,&A) ;
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,matrixSize,matrixSize) ;
MatSetFromOptions(A) ;
MatSetBlockSize(A,blockSize) ;
```

L'utilisation de la fonction `MatCreate()` est liée à la fonction `MatSetFromOptions()` qui permet de configurer le type de matrice attendue (creuse, dense, séquentielle, parallèle). Ici, nous faisons appel à une matrice creuse pour une utilisation séquentielle. Néanmoins cette formulation présente un inconvénient : PETSc ne connaît pas à cette étape la place mémoire que va prendre la matrice pendant l'execution du code. C'est au fur et à mesure de l'assemblage, que PETSc devra prévoir l'agrandissement de l'espace d'allocation à chaque ajout d'un élément dans la matrice. Par ailleurs, la dernière ligne permet une manipulation par bloc de la matrice, ce qui nous convient ici car nous construisons la matrice par ajouts successifs de matrices-blocs.

Cas optimisé.

On peut éviter le désavantage du cas précédent en utilisant la commande `MatCreateSeqBAIJ()`, ce qui suppose bien entendu de connaître la structure de la matrice.

```
Mat A ;
int matrixSize=1000 ;
int blockSize=5 ;
int nonZeroBlocks=3 ;
MatCreateSeqBAIJ(PETSC_COMM_SELF,BlockSize,matrixSize,
...                matrixSize,nonZeroBlocks,PETSC_NULL,&A) ;
return A ;
```

Tout l'enjeu est alors de situer l'emplacement des matrices-blocs dans la matrice finale. Dans notre problème, chaque cellule a au plus 3 voisins. Nous avons donc à prévoir au plus pour chaque cellule i l'emplacement de 4 matrices-blocs de taille 4×4 (n_k désigne l'indice de la cellule voisine k de la cellule i avec $k \in \{0, 1, 2\}$) :

$$\dots \underbrace{\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}}_{A[i,n_0]} \dots \underbrace{\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}}_{A[i,i]} \dots \underbrace{\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}}_{A[i,n_1]} \dots \underbrace{\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}}_{A[i,n_2]} \dots$$

Dès lors, la déclaration d'une matrice dans le cadre de notre projet peut se faire de la manière suivante :

```
Mat A ;
int nbCells=100 ;
int blockSize=4 ;
int matrixSize=blockSize*nbCells ;
int nonZeroBlocks=4 ;
MatCreateSeqBAIJ(PETSC_COMM_SELF,blockSize,matrixSize,
...                matrixSize,nonZeroBlocks,PETSC_NULL,&A) ;
```

```
return A ;
```
Avec cette déclaration, toute la mémoire exigée pour construire la matrice est allouée directement, ce qui permet un gain de temps dans l'assemblage de la matrice. On peut remarquer cependant qu'une partie de la mémoire n'est pas utilisée car il existe des cellules pour qui le nombre de voisins est inférieur à 3.

D.2 Assemblage d'une matrice par bloc

Pour comprendre l'assemblage par bloc d'une matrice avec PETSc faisons appel à un exemple. Soit A la matrice que l'on souhaite construire. Supposons que notre matrice-bloc est de taille 3×3 dont les valeurs ont été copiées (ligne par ligne) dans un tableau d'une ligne de taille 9 qu'on appelle b. La commande :

```
MatSetValuesBlocked(Mat A,int 1,int* i,int 1,int* j,
...                    PetscScalar b[],ADD_VALUES) ;
```

a alors pour équivalent en pseudo-code :

$$
\begin{array}{llllllllll}
A[& 3i & , & 3j &] & \leftarrow & b(0) + A[& 3i & , & 3j &] \\
A[& 3i & , & 3j+1 &] & \leftarrow & b(1) + A[& 3i & , & 3j+1 &] \\
A[& 3i & , & 3j+2 &] & \leftarrow & b(2) + A[& 3i & , & 3j+2 &] \\
A[& 3i+1 & , & 3j &] & \leftarrow & b(3) + A[& 3i+1 & , & 3j &] \\
A[& 3i+1 & , & 3j+1 &] & \leftarrow & b(4) + A[& 3i+1 & , & 3j+1 &] \\
A[& 3i+1 & , & 3j+2 &] & \leftarrow & b(5) + A[& 3i+1 & , & 3j+2 &] \\
A[& 3i+2 & , & 3j &] & \leftarrow & b(6) + A[& 3i+2 & , & 3j &] \\
A[& 3i+2 & , & 3j+1 &] & \leftarrow & b(7) + A[& 3i+2 & , & 3j+1 &] \\
A[& 3i+2 & , & 3j+2 &] & \leftarrow & b(8) + A[& 3i+2 & , & 3j+2 &]
\end{array}
$$

D.3 Création d'un vecteur

La manipulation des vecteurs est similaire à celle des matrices avec la question sur le caractère « creux » de l'objet en moins. On peut les déclarer de la manière suivante :

```
Vec x ;
int vecSize=1000 ;
int blockSize=4 ;
VecCreate(PETSC_COMM_WORLD, &x) ;
VecSetSizes(x,PETSC_DECIDE, vecSize) ;
VecSetFromOptions(x) ;
VecSetBlockSize(x,blockSize) ;
return x ;
```

L'utilisation de la fonction `VecCreate()` impose l'utilisation de la fonction `VecSetFromOptions()` qui fait appel à un vecteur « classique » ou la fonction `VecSetType()` qui permet de définir d'autres types de vecteur. Ici, nous travaillons uniquement en séquentiel il n'est donc pas necessaire de faire appel à `VecSetType()` qui permet, par exemple, de gérer la configuration d'un vecteur adapté au calcul parallèle (mpi).

D.4 Remplissage d'un vecteur par bloc

Nous allons évoquer le remplissage d'un vecteur par bloc sur un exemple. On suppose le bloc de taille 3. *val* est un tableau d'une ligne de taille 3 qui contient les valeurs du bloc. La commande :

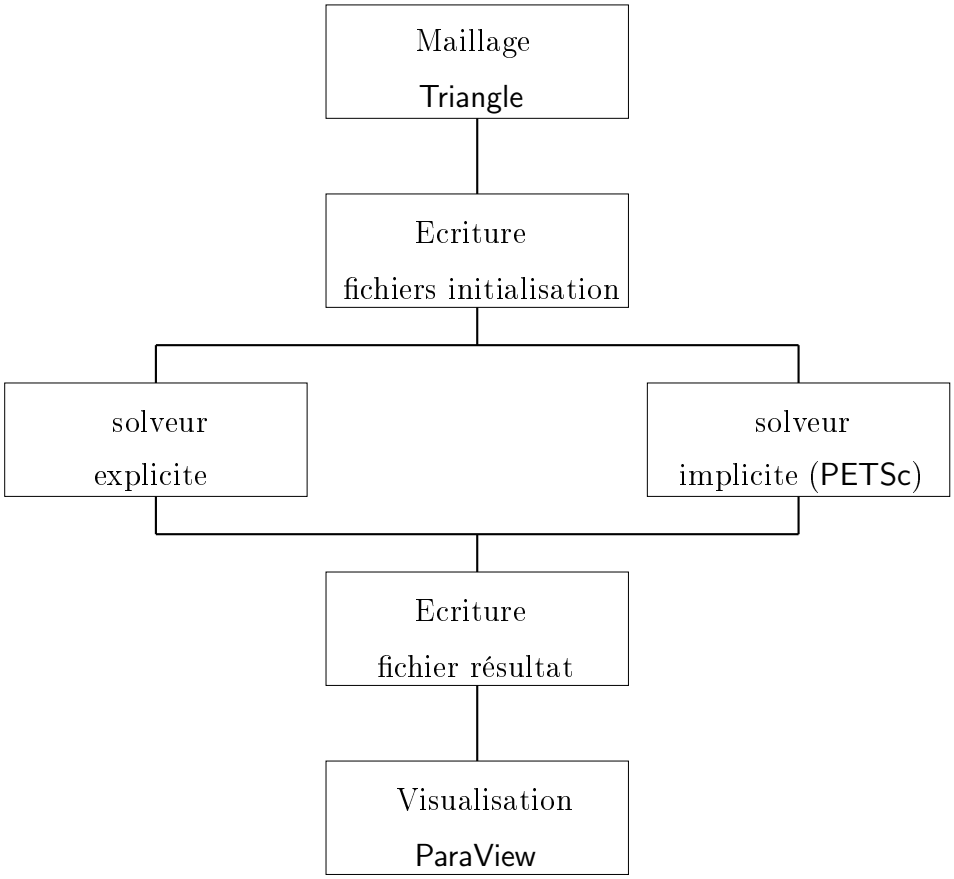


FIG. 57 – Principales étapes du projet

```
VecSetValuesBlocked(Vec b,int 1,int* i,PetscScalar val[],
...                          INSERT_VALUES) ;
```

a alors pour équivalent en pseudo-code :

$$
\begin{array}{llcll}
b[& 3i &] & \leftarrow & val[\ 0\ ] \\
b[& 3i+1 &] & \leftarrow & val[\ 1\ ] \\
b[& 3i+2 &] & \leftarrow & val[\ 2\ ]
\end{array}
$$

Table des figures

Références

[1] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page, 2009. http ://www.mcs.anl.gov/petsc.

[2] C. Chalons and J.-F. Coulombel. Relaxation approximation of the euler equations. 2007.

[3] J. R. Shewchuk. Triangle : Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, 1996. http ://www.cs.cmu.edu/ quake/triangle.html.

[4] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics.* Springer, 1999.