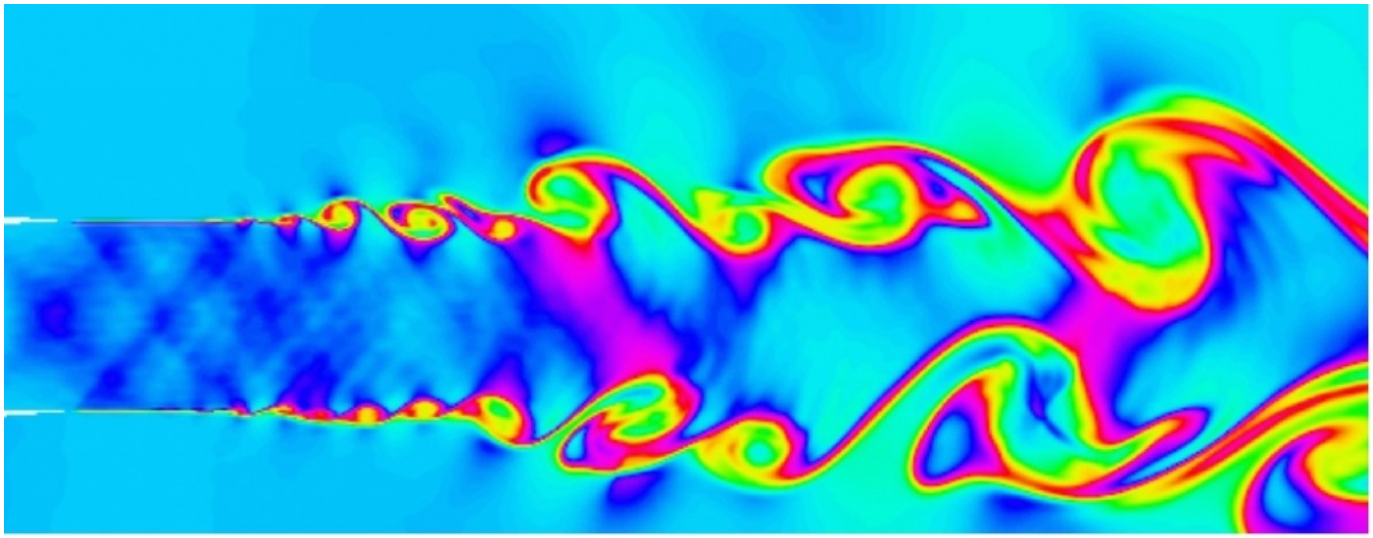
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet de Physique P6
STPI/P6/2023 - 027

Introduction au calcul scientifique : Solution numérique de l'équation de la dynamique des gaz



Étudiants :

Archibald BERNARD
Munh Baator BAT
Yassine ZAHOU
Rayane BENADJAOUD
Mathis VANNETZEL-GUITTET

Enseignant-responsable du projet :
Guido LODATO

Date de remise du rapport : 17/06/2023

Référence du projet : STPI/P6/2023 – 027

Intitulé du projet : Introduction au calcul scientifique : Solution numérique de l'équation de la dynamique des gaz

Type de projet : *modélisation, bibliographie.*

Objectifs du projet :

— Comprendre l'équation de la dynamique des gaz

— Développer un code de simulation en utilisant la méthode choisie pour résoudre l'équation de la dynamique des gaz

— Explorer les applications de la résolution numérique de l'équation de la dynamique des gaz

Mots-clefs du projet : *Simulation - dynamique des gaz - équation d'euler*

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN
DÉPARTEMENT SCIENCES ET TECHNIQUES POUR L'INGÉNIEUR
685 AVENUE DE L'UNIVERSITÉ BP 08- 76801 SAINT-ETIENNE-DU-ROUVRAY
TÉL : 33 2 32 95 66 21 - FAX : 33 2 32 95 66 31

Table des matières

Notations et Acronymes

Notations : — p : Pression
- U : Vitesse
- A : Section d'écoulement
- V : Volume d'écoulement
- ρ : Masse Volumique
- m : Masse
- P : Quantité de mouvement
- t : Temps
- F : Force
- E_T : Énergie totale
- e_T : Énergie totale massique
- W : Travail
- Q : Transferts thermiques
- T : Température
- e_i : Énergie interne

Acronymes : — PFD : Principe Fondamental de la Dynamique

1 Introduction

Présentation du sujet

La dynamique des fluides sert à expliquer beaucoup de phénomènes présents tout autour de nous. C'est un sujet vaste, décrit par une grande quantité d'équations. Celles que nous avons étudiées notamment sont les équations d'Euler ou équation de la dynamique des gaz.

Les équations d'Euler sont une simplification des équations de Navier-Stokes. Les équations de Navier-Stokes sont des équations aux dérivées partielles non linéaires permettant de décrire la manière dont se comportent les fluides Newtoniens. La résolution de celles-ci est extrêmement complexe et l'existence de leurs solutions n'est pas prouvée. Elles doivent leurs noms aux mathématiciens Henri Navier et Georges Gabriel Stokes qui les ont élaborées entre 1822 et 1845. Cependant, un grand nombre de scientifiques ont travaillé à l'avancement de ce sujet depuis maintenant 2 siècles. La résolution mathématique rigoureuse de ces équations constitue un des problèmes du prix du millénaire.

Les équations d'Euler découlent des équations de Navier-Stokes appliquées à un fluide (gaz ou liquide) sans échanges de chaleur avec l'extérieur et avec une viscosité nulle (pas de perte de quantité de mouvement par frottement).

Les équations d'Euler sont utilisées dès que les conditions réelles peuvent être approximées aux conditions d'utilisations de celles-ci. De plus, la relation de Bernoulli représente un cas particulier des équations d'Euler. Néanmoins, même si simplifiées, les équations d'Euler ne possèdent que très rarement des solutions analytiques, trouver une méthode de résolution numérique de celles-ci devient alors très intéressant.

Objectifs du projet

Dans ce projet, nous nous sommes donc attachés à produire un programme utilisant le langage GFortran capable de résoudre ces équations de manière numérique dans un espace à une dimension. Pour ce faire, nous avons étudié différentes méthodes numériques de résolution d'équations différentielles utilisant différents outils mathématiques, avant de choisir la plus efficace pour notre code.

2 Méthodologie, organisation du travail

Les premières séances étaient dédiées à la découverte et à l'étude des phénomènes physiques concernés par les équations d'Euler de la dynamique des gaz et les techniques utilisées pour les résoudre. Durant les séances suivantes, nous avons appris à utiliser le langage de programmation GFortran, afin de créer un code permettant de calculer les points flux et solutions des équations étudiées. Étant donné la nature très physique de l'aspect programmation, la majorité du groupe était concentrée sur le code, tandis qu'une personne se chargeait d'étudier la bibliographie sur le sujet. Les dernières séances du projet ont été utilisées pour vérifier le bon fonctionnement du programme et préparer les rendus de fin de projet tous ensemble (cf Figure 1).

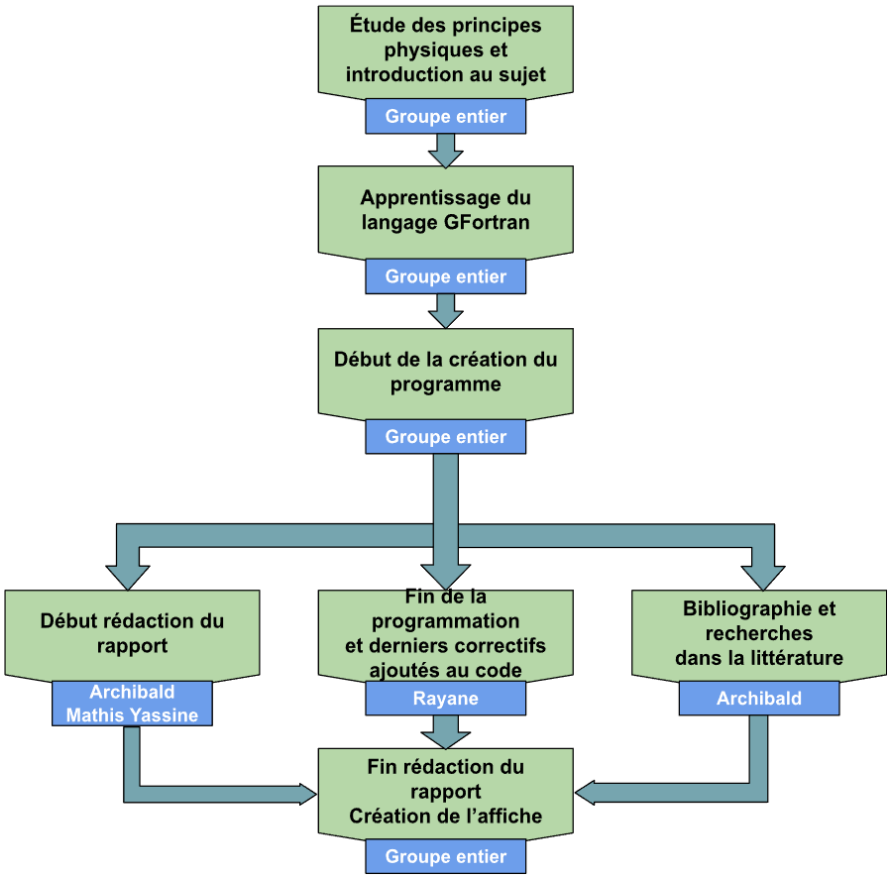


FIGURE 1 – Diagramme de Répartition du Travail

3 Travail réalisé et résultats

3.1 Fondements théoriques

3.1.1 Lois fondamentales de la dynamique de gaz

Afin d'analyser et d'établir les principes physiques appliqués à la résolution de l'équation de la dynamique des gaz, il faut étudier les trois principes fondamentaux de la physique au cœur du projet. Ces principes sont les suivants : principe de la conservation de la masse, principe de la conservation de la quantité de mouvement (deuxième principe de Newton) et principe de la conservation de l'énergie (premier principe de la thermodynamique). Le fluide étudié ici est considéré parfait (non visqueux) et en écoulement adiabatique. Étudions chacun de ces principes.

3.1.1.1 Principe de la conservation de la masse

Considérons une partie infinitésimale d'un écoulement de fluide suivant une direction selon l'axe des x (cf Figure 2).

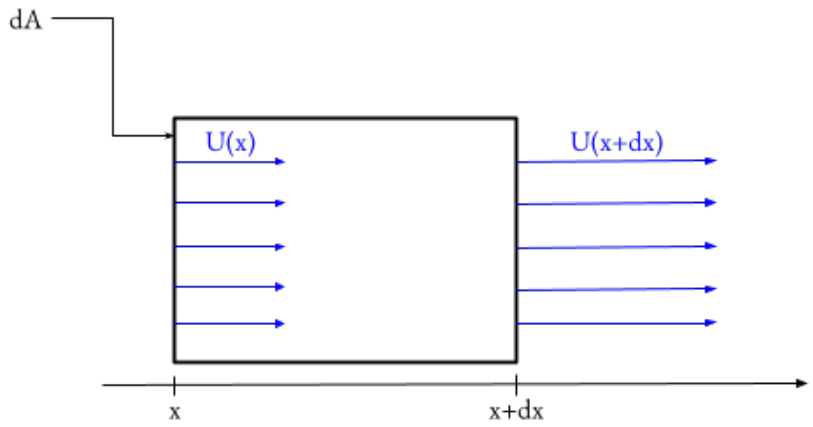


FIGURE 2 – Schéma d'un écoulement infinitésimal selon l'axe des x.

On a, dans la section considérée, une masse $dm = \rho dV$, avec $dV = dAdx$. Ainsi, on introduit l'équation de l'invariance de la masse :

$$\frac{D(dm)}{Dt} = \frac{D\rho}{Dt}dV + \rho\frac{D(dV)}{Dt}$$

Avec $\frac{D(dm)}{Dt}$ la dérivée particulaire de dm, définie comme tel, pour une variable X :

$$\frac{D(X)}{Dt} = \frac{\partial X}{\partial x}\frac{dx}{dt} + \frac{\partial X}{\partial y}\frac{dy}{dt} + \frac{\partial X}{\partial z}\frac{dz}{dt} + \frac{\partial X}{\partial t}$$

Ici seulement selon x et on a donc $\frac{dx}{dt} = U$.

Considérant le fait que la vitesse $U(x)$ des surfaces dA de chaque côté de l'écoulement (en x et $x + dx$) peuvent varier, on a :

$$\frac{D(dV)}{Dt} = [UdA]_{x+dx} - [UdA]_x = \frac{\partial(UdA)}{\partial x}dx$$

Le volume peut donc changer si les deux surfaces délimitantes ne vont pas à la même vitesse.

On reprend l'équation en considérant l'invariance de la masse (avec $\frac{D(dm)}{Dt} = 0$) :

$$\frac{D\rho}{Dt}dV + \rho\frac{\partial U}{\partial x}\underbrace{dAdx}_{dV} = 0$$

En simplifiant, on obtient enfin l'équation de la conservation de la masse :

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho U)}{\partial x} = 0 \tag{1}$$

3.1.1.2 Principe de la conservation de la quantité de mouvement

Considérons le même système que précédemment (voir Figure 2). On va introduire la notion de quantité de mouvement $P = Udm = \rho U dV$.

On applique le Principe Fondamental de la Dynamique :

$$\frac{D(dP)}{Dt} = \sum dF$$

On fait l'hypothèse que le système parallélépipédique considéré ici n'est soumis qu'aux forces de pression $F_p = pdA$, exercées à droite et à gauche du système (en $x + dx$ et x respectivement). On a donc :

$$\sum dF = [pdA]_x - [pdA]_{x+dx} = -\frac{\partial(pdA)}{\partial x}dx$$

De plus :

$$\frac{D(dP)}{Dt} = \frac{D(Udm)}{Dt} = U\frac{D(dm)}{Dt} + dm\frac{DU}{Dt}$$

Or il y a conservation de la masse donc $\frac{D(dm)}{Dt} = 0$, ainsi, si on rassemble les deux expressions, on obtient :

$$dm\frac{DU}{Dt} = -\frac{\partial(pdA)}{\partial x}dx$$

En développant cette expression, on obtient l'équation de la conservation de la quantité de mouvement :

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

3.1.1.3 Principe de la conservation de l'énergie

On considère le même système que précédemment (voir Figure 2). D'après le premier principe de la thermodynamique, on a l'équation suivante :

$$\frac{DE_T}{Dt} = \dot{Q} + \dot{W}$$

On considère un écoulement adiabatique donc $\dot{Q} = 0$

L'énergie totale a pour expression :

$$E_T = \rho e_T dV$$

Avec $e_T = \underbrace{e_i}_{C_V T} + \frac{1}{2}U^2$

On a donc d'une part :

$$\frac{DE_T}{Dt} = \frac{D(\rho e_T dV)}{Dt} = \rho \frac{D(e_T)}{Dt} dV$$

D'autre part :

$$\dot{W} = \frac{D(pdAdx)}{Dt} = dF \frac{Dx}{Dt} = U \underbrace{pdA}_{dF}$$

En rassemblant les deux parties de l'équation, on obtient :

$$\rho dV \frac{D(e_T)}{Dt} = [pU]_x dA - [pU]_{x+dx} dA = -\frac{\partial(pU)}{\partial x} dAdx$$

En développant et simplifiant, on obtient l'équation de la conservation de l'énergie :

$$\frac{\partial(\rho e_T)}{\partial t} + \frac{\partial(U(\rho e_T + p))}{\partial x} = 0 \tag{3}$$

3.1.2 Bilan des équations

Considérons les équations ici. On a 4 inconnues : ρ, U, p et x, pour 3 équations. Le gaz étant idéal, on a $p = \rho h T$. On a, de plus :

$$e_T = c_T T + \frac{1}{2}U^2 = \frac{R}{\gamma - 1} T + \frac{1}{2}U^2 = \frac{p}{(\gamma - 1)\rho} T + \frac{1}{2}U^2$$

On obtient donc le bilan suivant, les équations d'Euler :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial(\rho U)}{\partial x} = 0 & (1) \\ \dfrac{\partial(\rho U)}{\partial t} + \dfrac{\partial(\rho U^2 + p)}{\partial x} = 0 & (2) \\ \dfrac{\partial(\rho e_T)}{\partial t} + \dfrac{\partial(U(\rho e_T + p))}{\partial x} = 0 & (3) \end{cases}$$

Pour simplifier l'écriture pour la suite et faciliter la résolution numérique, on utilise la notation vectorielle. On a :

$$\vec{Q} = \begin{pmatrix} \rho \\ \rho U \\ \rho e_T \end{pmatrix} \qquad \text{et} \qquad \vec{F} = \begin{pmatrix} \rho U \\ \rho U^2 + p \\ \rho U(\rho e_T + p) \end{pmatrix}$$

Le bilan devient donc :

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{F}}{\partial x} = \vec{0}$$

3.1.3 Outils mathématiques utilisés

Le but de cette partie du rapport est de vous présenter quelques outils mathématiques dont nous aurons besoin pour la résolution de notre problème. En effet, il est crucial de les connaître et de les comprendre pour pouvoir appréhender la manière dont nous avons procédé afin de résoudre notre problème ainsi que la structure de notre code. Nous commencerons donc par vous présenter notre utilisation des matrices jacobienne, puis le rôle qu'a la quadrature de Gauss dans notre méthode de résolution et enfin, nous présenterons le fonctionnement de la méthode de l'interpolation de Lagrange.

3.1.3.1 Matrices Jacobiennes

En analyse vectorielle, on définit une matrice jacobienne comme la matrice des dérivées partielles du premier ordre d'une fonction vectorielle données. Dans notre cas, nous les utilisons dans un but bien précis. Les matrices jacobiennes nous servent à passer de l'espace physique à l'espace numérique. Pour ce faire, nous multiplions nos valeurs physiques par l'inverse de la Jacobienne en ce point-là.

3.1.3.2 Quadrature de Gauss

En analyse numérique, les quadratures sont des méthodes permettant d'approximer la valeur numérique d'une intégrale. La méthode de la quadrature de Gauss est une méthode de quadrature exacte d'un polynôme de degré $2n-1$.

Néanmoins, nous ne présenterons pas ici son fonctionnement, nous l'évoquons uniquement car pour résoudre notre problème nous nous basons sur les points de la quadrature de Gauss pour définir la matrice que nous utiliserons ensuite dans l'interpolation de Lagrange. En effet, procéder ainsi nous permet d'obtenir plus de stabilité dans les résultats.

3.1.3.3 Interpolation de Lagrange

L'interpolation numérique est une opération mathématique permettant de remplacer une fonction ou une courbe par une autre plus simple coïncidant avec la première en nombre défini de points. L'interpolation de Lagrange est une méthode d'interpolation polynomiale, c'est-à-dire que l'on cherche à remplacer notre fonction par un unique polynôme d'un degré aussi grand que nécessaire. Les interpolations polynomiales sont souvent utilisées dans le cas de fonctions continues car il est facile de dériver et d'intégrer des polynômes, de plus on obtient alors un autre polynôme, ce qui les rend pratiques pour simplifier un problème. Dans le cas de notre problème, nous utilisons une interpolation de Lagrange plutôt que les polynômes de Taylor car ces derniers sont faits de sorte à être le plus précis possible sur un seul point. À cause de cela, on pourrait perdre de la précision au fur et à mesure que l'on s'éloigne du dit point. On va donc lui préférer l'interpolation de Lagrange ou l'on établit au préalable un maillage constitué de plusieurs points par lequel notre polynôme devra passer. Ainsi, nous obtiendrons une meilleure précision sur notre ensemble de valeurs.

La formule de l'interpolation de Lagrange est en réalité assez simple, en effet, on peut l'écrire facilement :

$$f(x) \approx \sum_{i=1}^{n} l_i(x) f(x_i) \quad \text{avec} \quad l_i(x) = \prod_{k=1 \atop i \neq k}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

On peut voir ici que les $l_i(x)$ seront toujours égaux à 1 pour $x = x_i$ et 0 sinon. De ce fait, chaque produit $l_i(x)f(x_i)$ prendra la valeur de $f(x_i)$ au point x_i. Ainsi la fonction $f(x)$ étant la somme de tous ces produits sera le seul polynôme de degré n passant par tous les $n+1$ points x_i définis.

3.1.4 Méthodes numériques pour résoudre les équations différentielles

Dans cette partie du rapport, nous nous concentrerons sur les méthodes numériques pour résoudre les équations différentielles qui régissent la dynamique des gaz. Comme nous l'avons vu dans la section précédente, les lois fondamentales de la dynamique des gaz sont décrites par des équations différentielles complexes. La résolution analytique de ces équations n'est souvent pas possible, en particulier pour des problèmes réalistes et des conditions aux limites complexes. Par conséquent, il est essentiel de recourir à des méthodes numériques pour obtenir des solutions approchées.

Nous commencerons par présenter brièvement deux méthodes numériques couramment utilisées pour résoudre les équations différentielles : la méthode des différences finies et la méthode des volumes finis. Ensuite, nous nous concentrerons sur la méthode des éléments finis discontinus, qui sera la méthode principale sur laquelle nous travaillerons tout au long de la suite du rapport.

3.1.4.1 Méthode des différences finies

En général, la méthode des différences finies consiste à discréditer l'espace et le temps en un maillage régulier, et à approximer les dérivées spatiales et temporelles par des différences finies.

Afin d'appliquer cette méthode pour l'équation d'Euler, nous pouvons commencer par définir le problème continue, que l'on doit résoudre.

On a la fonction :

$$g(x,t) = \frac{\partial u(x,t)}{\partial t} + \frac{\partial f(u)}{\partial x}$$

Cette fonction correspond à la loi de conservation scalaire en dimension 1 avec u la variable d'état inconnue et $f(u)$ le flux de u avec les conditions limites suivantes à prendre en compte :

$$u(x,t)\big|_{t=0} = u_0(x), x \in \Omega$$

$$u(x,t)\big|_{x\in\partial\Omega} = u_B(t), t \in \mathbf{R}^+$$

On se place sur le segment que l'on discrétise à l'aide de la grille définie par ses sommets $x_k = kh$ avec $k \in [[1, K]]$ et h le pas de la grille, introduisant ainsi un maillage de K points tel que $x_1 < x_2 < .... < x_K$.

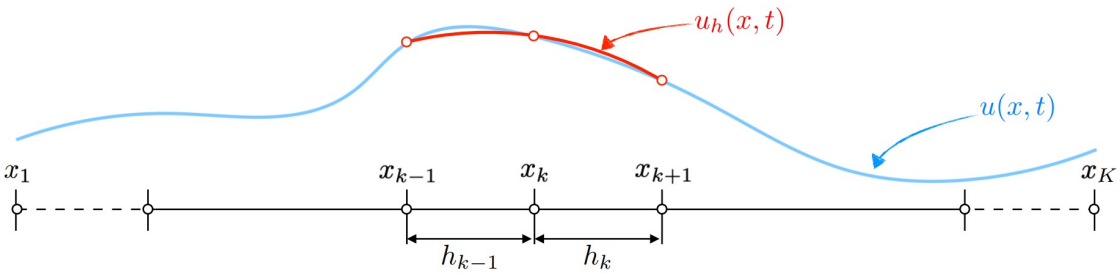


FIGURE 3 – Illustration de la méthode des différences finies

La méthode des différences finies suppose que la solution u(x, t) peut être approximée par des polynômes locaux 1D, dont les coefficients (a_i et b_i) sont déterminés en imposant que les fonctions approximatives interpolent les points de grille :

$$x \in [x_k : x_{k+1}] : u_h(x,t) = \sum_{i=0}^{2} a_i(t)(x - x_k)^i \quad , \quad f_h(x,t) = \sum_{i=0}^{2} b_i(t)(x - x_k)^i$$

Ainsi, pour obtenir une approximation de la solution du problème continu, il est nécessaire d'utiliser le résidu qui est défini comme la différence entre les termes temporels, spatiaux et les sources de l'équation de conservation. Le résidu est donné par :

$$R_h(x,t) = \frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - g(x,t)$$

Si la solution approximative était exacte, le résidu serait nul partout. Ainsi, pour formuler de manière discrète le problème, nous imposons que le résidu soit nul en chaque point de grille, c'est-à- dire lorsque $x = x_k$.

On obtient donc : $R_h(x,t) = \frac{\partial u_h(x_k)}{\partial t} + \frac{\partial f_h(x_k)}{\partial x} - g(x_k,t) = 0 \quad$ pour $k = 1, ..., K$

Maintenant, on procède à discrétiser les dérivées partielles $\frac{\partial f_h}{\partial x}$ en $x = x_k$ à l'aide de différences finies centrées, ce qui conduit à une expression en termes de $f_{k+1}(t)$ et $f_{k-1}(t)$, les évaluations du flux numérique aux points voisins :

$$\left.\frac{\partial f_h}{\partial x}\right|_{x=x_k} = b_1 = \frac{f_h(x_{k+1},t) - f_h(x_{k-1},t)}{h_k + h_{k-1}} \quad \text{si } h_k = h_{k-1}$$

On obtient enfin notre expression finale telle que :

$$g(x_k,t) = \frac{du_k(t)}{dt} + \frac{f_{k+1}(t) - f_{k-1}(t)}{h_k + h_{k-1}}$$

En résumé, la méthode des différences finies approxime la solution en utilisant des polynômes locaux 1D et impose que le résidu soit nul en chaque point de grille. Cela conduit à un ensemble d'équations aux différences finies qui décrivent l'évolution de la solution dans le domaine unidimensionnel Ω.

3.1.4.2 Méthode des volumes finis

Comparée à la méthode des différences finies, qui approxime les dérivées à l'aide de valeurs nodales, la méthode de volume fini évalue les expressions exactes de la valeur moyenne de la solution sur un certain volume et utilise ces données pour construire des approximations de la solution dans les cellules.

Pour commencer, comme dans la méthode des éléments finis, un maillage est construit, qui consiste en une partition du domaine où vit la variable d'espace.

En utilisant la même fonction précédente, ainsi que les conditions au limites, on a :

$$g(x,t) = \frac{\partial u(x,t)}{\partial t} + \frac{\partial f(u)}{\partial x} \quad \text{avec} \quad \begin{cases} u(x,t)\big|_{t=0} = u_0(x), x \in \Omega \\ u(x,t)\big|_{x\in\partial\Omega} = u_B(t), t \in \mathbf{R}^+ \end{cases} \tag{4}$$

Si nous supposons que l'équation (4) représente un milieu en écoulement de surface constante, nous pouvons subdiviser le domaine spatial, x, en volumes finis ou en cellules avec des centres de cellule indexés par k. Ainsi, pour une cellule donnée, k, nous pouvons définir la valeur moyenne en volume de $u_k(t) = u_h(x,t)$ au temps t , tel que $x \in D_k$ par :

$$\bar{u}_k(t) = \frac{1}{h_k}\int_{D_k} u_h(x,t)\, dx, \quad \text{avec} \quad h_k = x_{k+\frac{1}{2}} - x_{k-\frac{1}{2}}$$

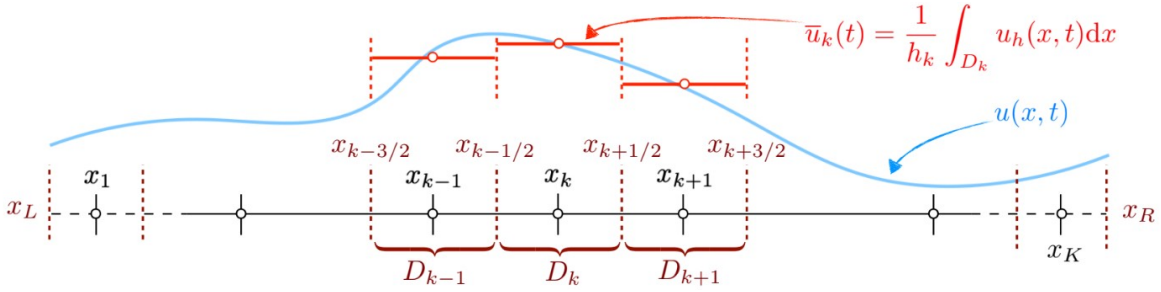


FIGURE 4 – Illustration de la méthode des volumes finis

Comme pour la méthode précédente, on utilise le résidu. Or l'objectif de la méthode des volumes finis est de trouver une solution approximée qui satisfasse certaines propriétés. Pour cela, on impose que l'intégrale du résidu sur chaque cellule soit nulle. Cela garantit la conservation de la quantité physique représentée par l'équation aux dérivées partielles. En d'autres termes, on a l'équation suivante :

$$\int_{D_k} R_h(x,t)dx = 0$$

$$\Leftrightarrow \int_{D_k}\left[\frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - g(x,t)\right]dx = \frac{d\bar{u}_k}{dt}h_k + \Big[f(u_h)\Big]_{x_{k-\frac{1}{2}}}^{x_{k+\frac{1}{2}}} - \bar{g_k}(t)h_k = 0 \ \texttt{pour} \ \ k = 1,...,K$$

Ainsi en appliquant le théorème de la divergence on a : $f_{k+\frac{1}{2}} = f(u_{k+\frac{1}{2}})$ avec $u_{k+\frac{1}{2}} = \frac{1}{2}(\bar{u}_k + \bar{u}_{k+1})$ ce qui donne :

$$\bar{g}_k(t) = \frac{d\bar{u}_k}{dt} + \frac{f_{k+\frac{1}{2}} - f_{k-\frac{1}{2}}}{h_k}$$

En résumé, la méthode des volumes finis subdivise le domaine en cellules, approxime la solution par des valeurs constantes par morceaux à l'intérieur de chaque cellule, définit le résidu dans chaque cellule, impose l'intégrale nulle du résidu sur chaque cellule pour assurer la conservation.

3.1.4.3 Méthode des éléments finis discontinus

La méthode des éléments finis discontinus est basée sur la discrétisation de l'espace en éléments finis, mais contrairement à la méthode des éléments finis classique, elle permet des discontinuités dans la solution. La méthode des éléments finis discontinus utilise des fonctions de base discontinues pour représenter la solution dans chaque élément fini. Ces fonctions de base sont définies localement dans chaque élément fini et sont généralement polynomiales. Les fonctions de base sont choisies de manière à satisfaire les conditions de continuité aux interfaces entre les éléments finis.

Ainsi, Comme précédemment, nous combinons la structure cellulaire (élémentaire) des volumes finis pour obtenir une flexibilité géométrique, avec la représentation polynomiale de la solution des différences finies pour obtenir une approximation d'ordre élevé :
À l'intérieur de chaque élément, la solution est développée en utilisant une base locale de fonctions de base $\psi k_i(x)$. La solution approximative dans l'élément k à l'instant t est représentée comme suit :

$$u_k^h(x,t) = \sum (a_i(t)\psi k_i(x)) \quad \text{pour} \quad x \in D_k$$

Dans cette expression, $a_i(t)$ est le coefficient associé au nœud i de l'élément k, et $\psi k_i(x)$ est la fonction de base locale associée à ce nœud.

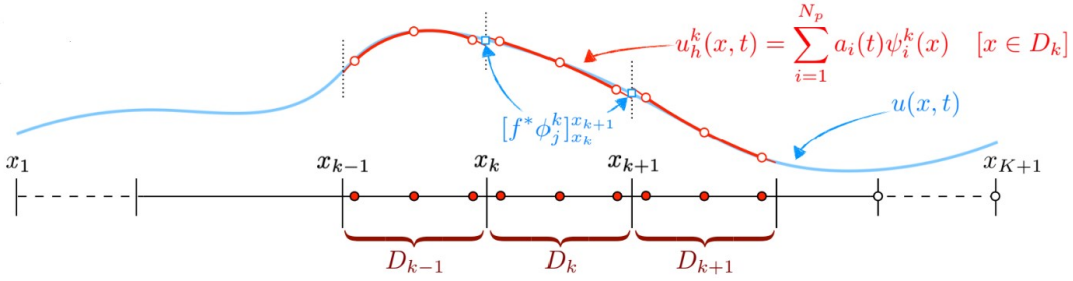


FIGURE 5 – Illustration de la méthode des éléments finis discontinus

Par ailleurs, une caractéristique clé de la méthode des éléments finis discontinus est la possibilité de discontinuité de la solution aux interfaces entre les éléments. Cela signifie que la solution peut varier brusquement d'un élément à un autre. Cette discontinuité est prise en compte et peut être différente d'un côté à l'autre de l'interface.

$$u_k^h(x_{k+},t) \neq u_k^h(x_{k-},t)$$

où x_{k+} est le point à l'interface sur le côté positif de l'élément k, et x_{k-} est le point à l'interface sur le côté négatif de l'élément k.

Comme précédemment, on utilise l'orthogonalité du résidu, défini localement dans chaque élément k comme suit :

$$R_k^h(x,t) = \frac{\partial u_h}{\partial t} + \frac{\partial f_h}{\partial x} - g(x,t)$$

Pour obtenir une formulation discrète du problème, le résidu $R_k^h(x,t)$ doit être orthogonal à toutes les fonctions de test $\phi_j^k(x)$ dans un espace V_h. Cela est formulé mathématiquement comme suit :

$$\int_{D_k} R_k^h(x,t)\phi_j^k(x)\,dx = 0 \quad \text{pour tout } k \text{ et } j$$

Cette condition d'orthogonalité garantit que le résidu est "annulé" lorsqu'il est projeté sur les fonctions de test $\phi_j^k(x)$ à l'intérieur de chaque élément.

Or, aux interfaces entre les éléments, un flux numérique spécifique f* est utilisé pour assurer la stabilité dans les problèmes dominés par les ondes. L'équation d'orthogonalité du résidu peut être réécrite en tenant compte de ce flux numérique :

$$\int_{D_k} \frac{\partial u_{kh}}{\partial t}\phi_j^k(x)\,dx - \int_{D_k} \frac{\partial f_{kh}}{\partial x}\phi_j^k(x)\,dx - \int_{D_k} g(x,t)\phi_j^k(x)\,dx = -\left[f^*\phi_j^k(x)\right]_{x_k}^{x_{k+1}} \quad \text{pour tout } k \text{ et } j$$

Cette équation établit la connexion entre les éléments adjacents, où $-\int_{[x_{k-},x_{k+}]} f^*\phi_{kj}(x)\,dx$ représente le flux numérique à l'interface entre les éléments k et $k+1$. Ce terme permet de transmettre l'information entre les éléments voisins de manière cohérente.

En résumé, la méthode des éléments finis discontinus combine l'expansion polynomiale de la solution approximative, la possibilité de discontinuité de la solution aux interfaces, la formulation du résidu localement dans chaque élément, l'orthogonalité du résidu par rapport aux fonctions de test, et l'utilisation d'un flux numérique spécifique aux interfaces pour assurer la stabilité. Ces aspects mathématiques sont essentiels pour la formulation et la résolution des problèmes avec la méthode des éléments finis discontinus.

3.2 Mise en œuvre des méthodes numériques

Le code est constitué d'un fichier principal nommé main et de 7 subroutines qui seront détaillés ci-dessous : (voir code entier en annexe 4)

- `Jacobian` : Permet de créer un vecteur qui transforme une discrétisation quelconque (espace physique) sur le domaine [0 ;1] (espace numérique). En une dimension 1 cela consiste à diviser chaque élément par sa taille. Le vecteur ainsi obtenu permet de passer de la solution physique a la solution numérique et l'inverse de chaque élément le constituant permet de passer de l'espace numérique à l'espace physique.
- `GetXsXf` : Cette fonction obtient les coordonnées des points solutions et points flux codé en dur en fonction de l'ordre de simulation voulu. Ceci ont été trouvés grâce à la quadrature de Gauss.
- `GetLmatMmat` : Est appelée pour créer deux matrices `Lmat` permet de réaliser l'interpolation de Lagrange et `Mmat` qui elle fait la dérivée du polynôme de Lagrange. Pour le calcul de `Lmat` deux boucles `do` sont nécessaires dans lequel on retranscrit le calcul détaillé dans la partie "Interpolation de Lagrange :" Le calcul de `Mmat` se fait avec trois boucles do qui réalise le calcul du résidu détaillé dans la partie 3.1.4.3. Les valeurs de Xs et Xf sont déjà connues grâce à `GetXsXf` déjà présenté.
- `FluxIntérieur` : Elle calcule en appliquant directement l'expression du Flux a la solution numérique sur chaque point flux.
- `FluxInterface` : Permet le calcul du flux aux interfaces entre les éléments grâce à la méthode du Roe Solver, cette subroutine contient également une autre subroutine `entropyFix` qui permet de corriger certaines erreurs de stabilité au niveau des interfaces, notamment lorsque des chocs se propagent.
- `Compresid` : Cette subroutine calcule le flux résiduel $\frac{df}{dx_i}$ qui sera ensuite additionné à chaque itération de `Rk3`. Pour ce faire il y a plusieurs étapes :
  1. On multiplie (produit matriciel) la solution physique par la Lmat pour faire une interpolation et obtenir la solution aux points flux.
  2. On utilise la solution aux points flux pour faire le calcul des flux grâce à `FluxIntérieur` et `FluxInterface`.
  3. On rassemble les différentes composantes du flux dans un seul tableau.
  4. On calcule enfin la résiduel en multipliant (produit matriciel) le tableau rassemblant les flux par la `Mmat`.
- `Rk3` : Fonction permettant de faire l'itération dans le temps. Il s'agit d'appliquer la méthode créée par GOTTLIEB S. et SHU C. Dans la boucle `do`, on fera appel à `Compresid` pour récupérer la résiduel, puis on calcule le pas de temps en fonction de la vitesse, la vitesse du son et la taille du domaine. Grâce à ces deux informations, on peut calculer la solution numérique aux points solution que l'on transformera en solution physique juste après grâce à l'inverse de La Jacobienne. Cette opération est répétée 3 fois suivant le modèle runge kutta 3. À la fin, on obtient en sortie la solution à l'instant T+1.

— `Main` : Il s'agit du programme principal, on commence par déclarer les variables globales. Puis par allouer les différents tableau puis on calcule la taille du domaine (ligne 49) et on crée l'espace physique dans le vecteur "X", on fait appel au fonctions `Jacobian`, `getXsXf` et `GetLmatMmat` pour obtenir respectivement la jacobienne et son inverse (`Jac` et `InvJ`), les coordonnées des points solutions et points flux, la `Lmat` et `Mmat`. On initialise ensuite les différents cas test en donnant à chaque élément une valeur précise suivant le cas que l'on veut réaliser.
On crée ensuite le tableau de la solution physique, puis la solution numérique en le multipliant par la jacobienne (`Jac`). On commence ensuite les itérations avec une boucle do while, la première ligne consiste à vérifier que `check = 0` pour éviter une boucle infini (le nombre d'itération maximale étant définis et si dépasser alors `check = 1`), ensuite une autre boucle intervient pour enregistrer dans des fichier les résultats obtenus à certaines itérations. enfin, on fait appel à rk3 pour le calcul de la solution à l'instant i+1 autant de fois que nécessaire suivant le pas de temps, la simulation s'arrête lorsque t = 0.2s.

3.3 Applications pratiques et résultats

Appliquons maintenant les méthodes numériques à des cas particuliers, possédant chacun des conditions initiales différentes. Toutes les variables sont mises en rapport à une variable de référence, donc elles sont sans unité. Le domaine est alors le méme et sera de longueur 1 et la durée de simulation est de 0.2.

3.3.1 Sod Shock Tube

Le premier cas-test est celui du tube à choc, un instrument utilisé pour reproduire et concentrer les ondes de détonation afin de simuler des explosions et leurs effets, il permet aussi d'étudier des phénomènes aérodynamiques difficiles à obtenir en soufflerie. Pour produire l'onde de choc on partage le tube en une ou plusieurs cellules séparées par une fine membrane, la masse volumique et la pression sont différentes dans chacune d'entre elles. À t = 0 on brise les frontières ce qui cause une propagation de choc et c'est ce que nous allons simuler. Notre modèle contient deux cellules de taille identique, les conditions initiale sont : $X \in [0; 0.5]$, $P = 1$, $U = 0$, $\rho = 1$. Et on a pour $X \in [0.5; 1]$, $P = 0.1$, $U = 0$, $\rho = 0.125$

Au bout de 0.2s, on obtient l'état suivant : (voir Figures 6, 7)

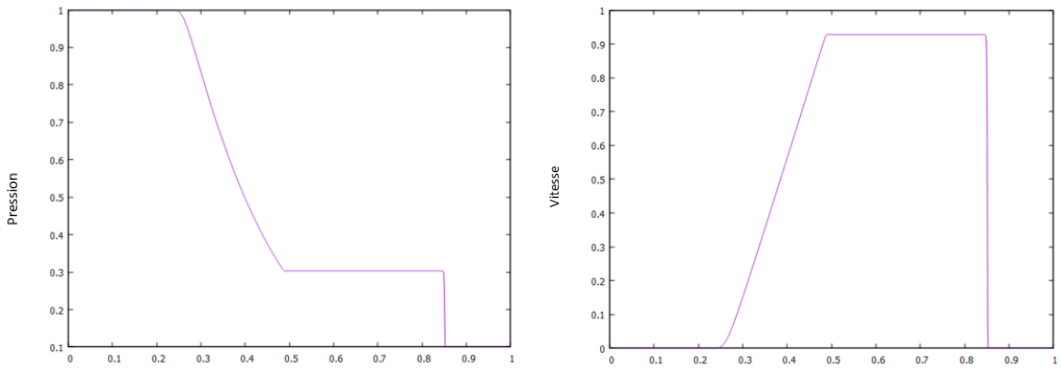


FIGURE 6 – Diagrammes d'évolution spatiale de la Pression et de la Vitesse

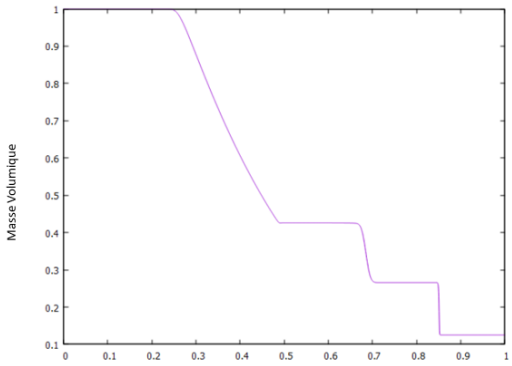


FIGURE 7 – Diagramme d'évolution spatiale de la masse volumique ρ Sod Shock Tube

3.3.2 Advection pure d'un profil de masse volumique

Le second cas est une advection pure d'une vitesse et d'une pression constantes et d'une masse volumique sinusoïdale. L'installation est semblable à un tor de grande taille, les conditions aux extrémités gauche et droite sont identiques. Les conditions initiales sont les suivantes : $X \in [0; 1]$, $P = 1$, $U = 1$, $\rho = 1 + 0.2 \sin(2\pi x)$.

Dans ce cas, la masse volumique se distribue suivant une onde sinusoïdale qui se déplace dans l'espace à la vitesse u, la pression et la vitesse sont constantes dans le temps et l'espace. On obtient donc les résultats suivants à $t_0 = 0$ et à $t_f = 0.2$: (voir Figures 8,9)

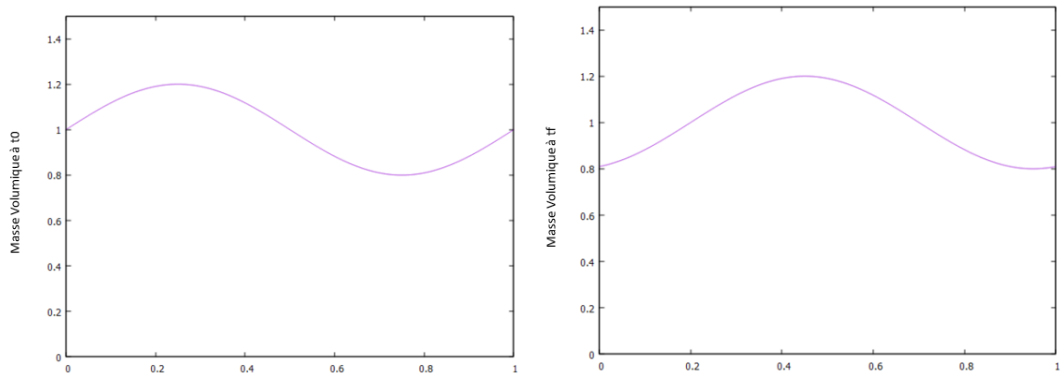


FIGURE 8 – Diagrammes d'évolution spatiale de la masse volumique ρ à t_0 et à t_f pour Advection pure d'un profil de masse volumique

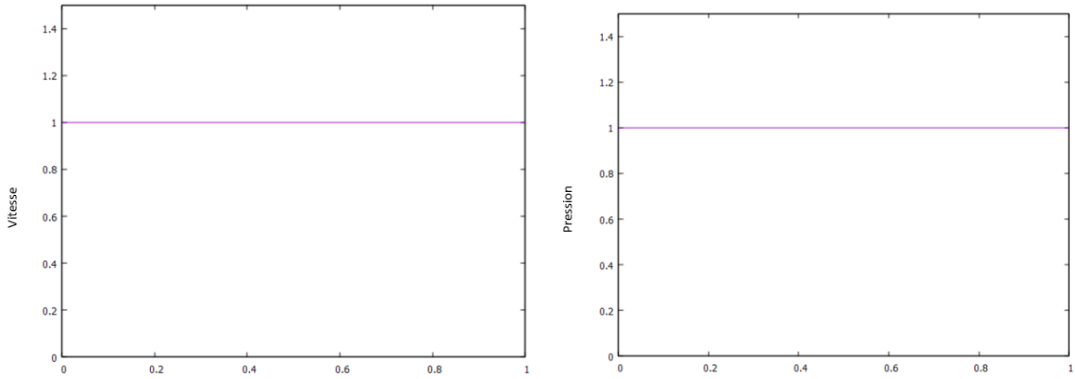


FIGURE 9 – Diagrammes d'évolution spatiale de la Vitesse et de la Pression

À noter ici que la pression et la vitesse ne varient pas selon t pour ce deuxième cas-test.

3.3.3 Tube de Shu-Osher

Pour ce troisième cas-test, on modélise le déplacement d'une onde de choc dans un écoulement non visqueux avec fluctuation de la masse volumique. Les conditions initiales sont comme suit : Pour $X \in [0; 0.1]$, $P = 10.33333$, $U = 2.629369$, $\rho = 3.857143$.
Et pour $X \in [0.1; 1]$, $P = 1$, $U = 0$, $\rho = 1 + 0.2 \sin(50(x - 0.5))$. On obtient donc les résultats suivants : (voir Figures 10, 11)

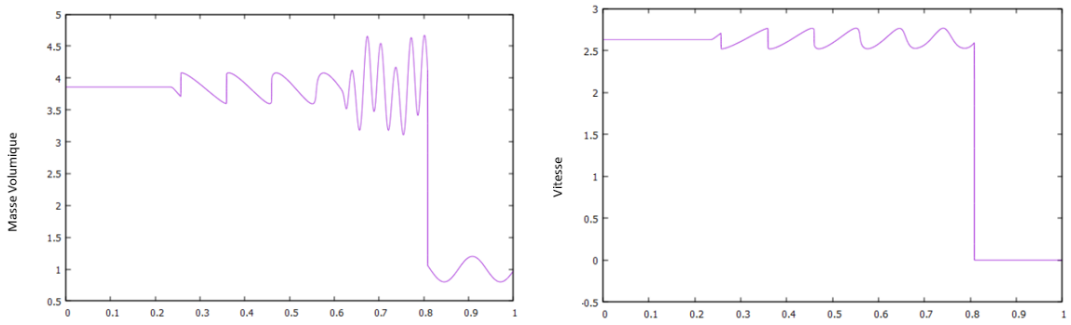


FIGURE 10 – Diagrammes d'évolution spatiale de la masse volumique ρ et de la vitesse pour le cas-test Tube de Shu-Osher

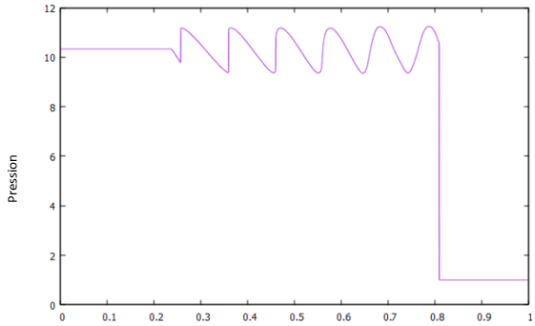


FIGURE 11 – Diagramme d'évolution spatiale de la pression cas-test Tube de Shu-Osher

3.3.4 Comparaison avec et sans entropyfix

Nous avons réalisé la simulation sur le tube a choc simple cette fois en désactivant la subroutine `entropyFix`, pour faire apparaître son rôle on remarque que globalement le résultat est le même sauf au niveau du choc ou une instabilité numérique se crée. La subroutine `entropyFix` permet donc de résoudre ce problème et de rendre le résultat plus cohérent. (voir Figure 12)

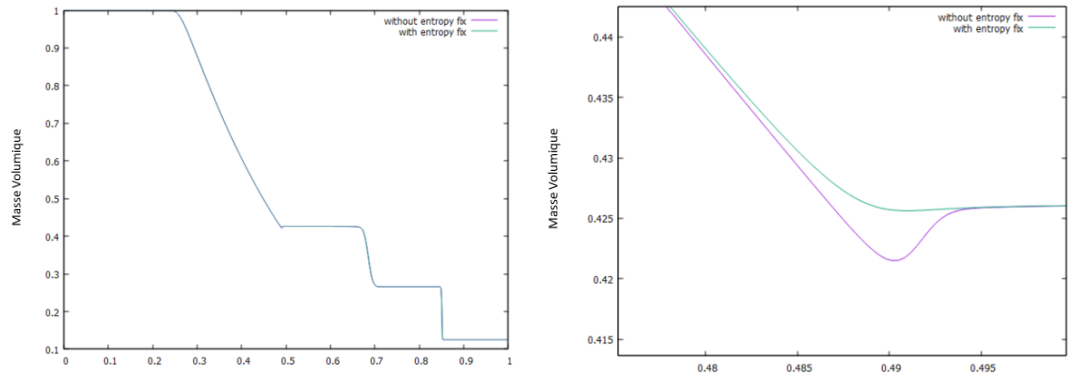


FIGURE 12 – Comparaison diagrammes d'évolution spatiale de la masse volumique avec/sans `entropyFix`, section $[0.475; 0.5]$ zoomée à droite

3.3.5 Comparaison Shu-Osher à une référence

Cette fois, on compare le résultat obtenu avec notre code avec Nelem = 10000 à l'ordre 1 avec les résultats fourni par le code de l'encadrant avec Nelem = 2000 à l'ordre 5. Les deux codes nécessitent une puissance de calcul similaire. Les résultats sont dans la figure 13 :

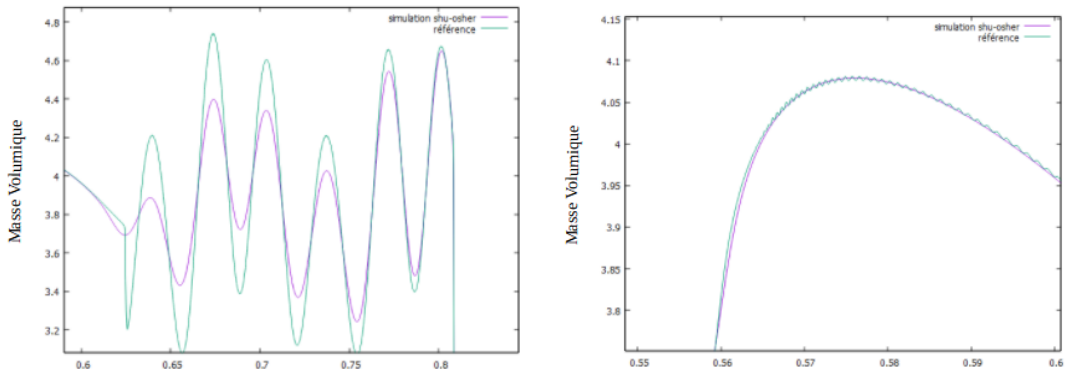


FIGURE 13 – Comparaison diagrammes d'évolution spatiale de la masse volumique avec la référence, section $[0.55; 0.6]$ zoomée à droite

On voit une différence notable dans la zone de turbulence due à la dissipation numérique présente dans notre code. On voit aussi que dans la simulation d'ordre plus élevé cette dissipation a beaucoup moins d'impact, en revanche des oscillations apparaissent tout le long de la courbe.

Il existe néanmoins des méthodes pour éliminer les oscillations[6], il est ainsi possible d'obtenir un résultat beaucoup plus beau tout en évitant le risque de créer certaines instabilités fatale. (Voir figure 14)

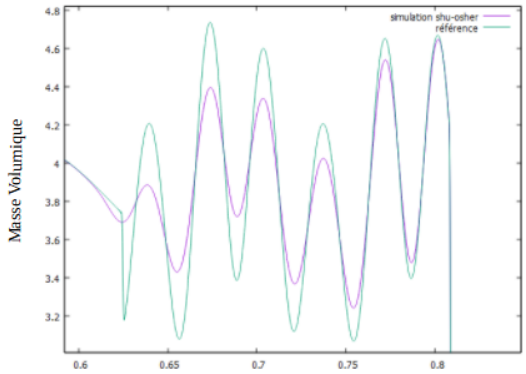


FIGURE 14 – Comparaison diagrammes d'évolution spatiale de la masse volumique avec la référence, après correction

4 Conclusion et perspectives

En utilisant des principes de base de la physique et de la dynamique des fluides, ainsi que des méthodes de résolution numérique, nous avons réussi à atteindre l'objectif que l'on s'était fixé, soit simuler la dynamique d'un gaz en une dimension, comme nous l'avons montré dans l'étude de différents cas-test. Notre code est une base qui peut être développée d'avantage pour avoir de réelles applications. D'abord, il pourrait être étendu aux trois dimensions de l'espace. L'ajout d'un facteur qui prendrait en compte une section variable permettrait de simuler le comportement d'un gaz traversant une tuyère par exemple. Notre programme, couplé avec celui réalisé par le groupe sur la diffusion de la chaleur (groupe de projet 28), permettrait de modéliser les équations de Navier-Stokes en ajoutant le facteur de dissipation thermique à notre code. Ainsi ce que nous avons réussi à accomplir au travers de ce projet constitue le premier bloc sur lequel se construisent des simulations beaucoup plus complexes.

Références

[1] G. Lodato, "*High Order DFE Methodes*"

[2] Richard L. Burden, Douglas J. Faires, Anette M. Burden, "*Numerical Analysis*"

[3] Sébastien TORDEUX, Victor PÉRON, "*Analyse numérique : La méthode des différences finies*", 2020/2021.

[4] Roe, Philip L., "*Approximate Riemann solvers, parameter vectors, and difference schemes*", Journal of computational physics 43.2 (1981) : 357-372.

[5] GOTTLIEB S., SHU C., "*Total variation diminishing Runge-Kutta schemes*", Mathematics of Computation 67(221) : 73-85, 1998..

[6] LODATO ET AL, "*Mitigation of post-shock oscillations induced by artificial viscosity in discontinuous finite element methods*", Computers and Fluids 241 (2022) 105491

[7] https://fr.wikipedia.org/wiki/%C3%89quations_de_Navier-Stokes (Valide à la date du 14/06/2023)

[8] https://fr.wikipedia.org/wiki/M%C3%A9thodes_de_quadrature_de_Gauss#Calcul_des_points_et_poids_de_quadrature (Valide à la date du 14/06/2023)

[9] https://en.wikipedia.org/wiki/Finite_volume_method (Valide à la date du 14/06/2023)

Annexes

Code

— main :

```
module globals
  implicit none


  integer :: N = 1 !nombre de points solutions
  integer :: Nelem = 1000 !Nombres d'element dans le domaine(doit etre pair)
  integer :: Niter = 999999 !nombre d'iterations
  integer :: test = 2 !selection du cas test
                      ! 1 : tube a choc
                      ! 2 : tube torique de grande taille
                      ! 3 : tube a choc de Shu-Osher
  double precision :: domaine = 1.0d0 !taille du domaine
  double precision :: t_final = 2.0d0 ! duree de la simulation
  double precision :: cft = 0.8d0 !condition de stabilite(courant-Friedrich-Lewy)
  double precision :: gamma = 1.4d0 !constante gaz monoatomique
  double precision :: time = 0.0d0 ! temps initial
  double precision :: h ! taille d'un element
  double precision :: dt ! pas temporelle
  double precision, parameter :: pi = dacos(-1.D0)


  double precision,allocatable :: X(:) ! discretisation du domaine
  double precision,allocatable :: Xs(:) ! valeurs quadrature de Gauss
  double precision,allocatable :: Xf(:) ! valeurs quadrature de Gauss
  double precision,allocatable :: Jac(:) ! la jacobienne
  double precision,allocatable :: invJ(:) ! l'inverse de la jacobienne
  double precision,allocatable :: Mmat(:,:) ! interpolation de Lagrange !!!!!
  double precision,allocatable :: Lmat(:,:) ! interpolation de Lagrange !!!!!


end module globals

program main
  use globals

  implicit none


  integer :: i,j,check,is,ic
  double precision, allocatable :: Us(:,:,:)! 3 equations; N solution par elem;
    Nelem / solution physique
  double precision, allocatable :: Un(:,:,:)! 3 equations; N solution par elem;
    Nelem / solution numerique
  double precision, allocatable :: P(:,:),U(:,:),Ro(:,:)! element; Nbr iterations
  double precision :: Pt,Ut


  write(*,*) '------- START -------'


  !allocation
  allocate(X(Nelem),Xs(N),Xf(N+1),Jac(Nelem),invJ(Nelem),Mmat(N,N+1),Lmat(N+1,N)
    ,&
           Us(3,N,Nelem),Un(3,N,Nelem),Ro(N,Nelem),P(N,Nelem),U(N,Nelem))

  !maillage
  h = Domaine/real(Nelem) ! calcul du domaine


  do ic=1,Nelem
    X(ic) = real(ic - 1) * h ! creation du maillage
  end do
```

```
call jacobian


call getXsXf


call get_lmat_Mmat
!initialisation

! 1 : tube a choc
if (test.eq.1) then
  do j=1,N
    do i=1,(Nelem/2)
      P(j,i) = 1.0d0
      U(j,i) = 0.0d0
      Ro(j,i) = 1.0d0


      P(j,Nelem+1-i) = 0.10d0
      U(j,Nelem+1-i) = 0.0d0
      Ro(j,Nelem+1-i) = 0.125d0
    end do
  end do
end if


! 2 : tube torique de grande taille
if (test .eq. 2) then
  do j=1,N
    do i=1,(Nelem)
      P(j,i) = 1.0d0
      U(j,i) = 1.0d0
      Ro(j,i) = 1.0d0 + 0.2d0*sin (2*pi * x(i))
    end do
  end do
end if


! 3 : tube a choc de Shu-Osher
if (test.eq.3) then
  do j=1,N
    do i=1,(Nelem/10)
      P(j,i) = 10.333333d0
      U(j,i) = 2.629369d0
      Ro(j,i) = 3.857143d0
    end do
    do i=(Nelem/10)+1,(Nelem)
      P(j,i) = 1.0d0
      U(j,i) = 0.0d0
      Ro(j,i) = 1.0d0 + 0.2d0 * sin(50*( x(i) - 0.5d0))
    end do
  end do
end if


!creation du vecteur solution
Us(1,:,:) = Ro(:,:)
Us(2,:,:) = Ro(:,:)*U(:,:)
Us(3,:,:) = P(:,:)/(gamma-1.0d0) + 0.5d0 * Ro(:,:) * U(:,:)**2


!numerisation
do i=1,Nelem
  Un(:,:,i) = Jac(i)*Us(:,:,i)
end do


!integration en temps
```

```
i = 1
check = 0
do while (check.eq.0)
  if (time.ge.t_final .or. i.gt.Niter-1) check = 1
  if (modulo(i,100) == 1) then
    do ic=1,Nelem
      do is=1,N
        Ut = Us(2,is,ic)/Us(1,is,ic)
        Pt = (gamma-1.0d0)*(Us(3,is,ic)- 0.5d0 * Us(1,is,ic)*(Ut**2))
        write(i+10,*) X(ic) + Xs(is) * h, Us(1,is,ic),Ut,Pt
      end do
    end do
  end if
  u(:,:) = Us(2,:,:)/Us(1,:,:)
  call rk3(Un,Us)
  i = i + 1
end do

do ic=1,Nelem
  do is=1,N
    Ut = Us(2,is,ic)/Us(1,is,ic)
    Pt = (gamma-1.0d0)*(Us(3,is,ic)- 0.5d0 * Us(1,is,ic)*(Ut**2))
    write(i+10,*) X(ic) + Xs(is) * h, Us(1,is,ic),Ut,Pt
  end do
end do


deallocate(X,Xs,Xf,Jac,invJ,Mmat,Lmat,Us,Un,P,U,Ro)
write(*,*)"----------END----------"


contains
include 'compresid.f90'
include 'flux_interface.f90'
include 'Flux_interieur.f90'
include 'getLmat_Mmat.f90'
include 'getXsXf.f90'
include 'jacobian.f90'
include 'function_rk3.f90'


end program
```

— jacobian :

```
subroutine jacobian
use globals
implicit none


integer :: i


do i=1,Nelem
    Jac(i) = h
end do


invJ(:) = 1.0d0 / Jac(:)
end subroutine
```

— getXsXf :

```
subroutine getXsXf
use globals
implicit none
```

```
double precision :: A2,A3,A4,B4,A5,B5 !stockage des valeur des points de gauss


A2 = 1.0d0 / sqrt(3.0d0)
A3 = sqrt(0.6d0)
A4 = sqrt(3.0d0 / 7.0d0 - (2.0d0 / 7.0d0)*sqrt(1.2d0))
B4 = sqrt(3.0d0 / 7.0d0 + (2.0d0 / 7.0d0)*sqrt(1.2d0))
A5 = (1.0d0 / 3.0d0)*sqrt(5.0d0 - 2.0d0*sqrt(10.0d0 / 7.0d0))
B5 = (1.0d0 / 3.0d0)*sqrt(5.0d0 + 2.0d0*sqrt(10.0d0 / 7.0d0))


select case (N)
case(1)
    Xs = (/0.0d0/)
    Xf = (/-1.0d0,1.0d0/)
case(2)
    Xs = (/-A2,A2/)
    Xf = (/-1.0d0,0.0d0,1.0d0/)
case(3)
    Xs = (/-A3,0.0d0,A3/)
    Xf = (/-1.0d0,-A2,A2,1.0d0/)
case(4)
    Xs = (/-B4,-A4,A4,B4/)
    Xf = (/-1.0d0,-A3,0.0d0,A3,1.0d0/)
case(5)
    Xs = (/-B5,-A5,0.0d0,A5,B5/)
    Xf = (/-1.0d0,-B4,-A4,A4,B4,1.0d0/)
end select


Xs = 0.5d0 * (Xs + 1.0d0)
Xf = 0.5d0 * (Xf + 1.0d0)


end subroutine
```

— get$_l mat_M mat$:

```
subroutine get_lmat_Mmat
use globals
implicit none


integer :: i,j,k,l
doubleprecision :: num,den,sum


!calcul Lmat
do k=1,N+1
  do j=1,N
    num = 1.0d0
    den = 1.0d0
    do i=1,N
      if(i.ne.j) then
        num = num*(Xf(k)-Xs(i))
        den = den*(Xs(j)-Xs(i))
      end if
    end do
    Lmat(k,j) = num/den
  end do
end do


!calcul Mmat
do i=1,N
  do j =1,N+1
    sum = 0.0d0
```

```
      do l=1,N+1
        if (l.ne.j) then
          den = 1.0d0
          num = 1.0d0
          do k=1,N+1
            if ((k.ne.j).and.(k.ne.l)) then
              num = num *(Xs(i)-Xf(k))
            end if
            if (k.ne.j) then
              den = den*(Xf(j) - Xf(k))
            end if
          end do
          sum = sum + num/den
        end if

      end do
      Mmat(i,j)=sum
    end do
  end do
  end subroutine
```

— FluxInterieur :

```
  subroutine FluxInterieur(Uf,F)
  use globals
  implicit none


  integer :: j


  double precision, intent(in) :: Uf(3,N+1)
  double precision, intent(out) :: F(3,N-1)
  double precision :: u


  !calcul du flux
  do j=1,N-1
      u = Uf(2,j+1)/Uf(1,j+1)
      F(1,j) = Uf(2,j+1)
      F(2,j) = Uf(2,j+1) * u + (gamma-1) * (Uf(3,j+1) - 0.5d0 * Uf(2,j+1) * u )
      F(3,j) = u * (Uf(3,j+1) + (gamma-1) * (Uf(3,j+1) - 0.5d0 * Uf(2,j+1) * u ))
  end do


  end subroutine
```

— fluxinterface :

```
  subroutine fluxinterface(Ql,Qr,Fint)


  implicit none


  double precision, intent(in) :: Ql(3),Qr(3)
  double precision, intent(out) ::Fint(3)
  double precision :: F(3),Vect1(3),Vect2(3),alpha(3),Xi(3),deltaQ(3),AFJ(3)&
      &,rrho,rrhoL,rrhoR,ul,ur,um,pl,pr,hl,hr,hm,sq_rho,usq,am_sq,am &
      ,eta_pos,eta_neg,a1,a2,a3,a4,lambda(-1:1),uk,ak,aL(-1:1),aR(-1:1)


  double precision, parameter :: gamma = 1.4d0


  rrhoL = 1.0d0 / Ql(1) !inverse de Ro gauche puis droite
```

```
rrhoR = 1.0d0 / Qr(1)
uL = rrhoL * Ql(2) !vitesse gauche puis droite
uR = rrhor * Qr(2)
pL = (Ql(3) - 0.5d0 * uL * Ql(2)) * (gamma - 1.0d0) !pression gauche puis droite
pR =(Qr(3) - 0.5d0 * uR * QR(2)) * (gamma -1.0d0 )
hL = rrhoL * (Ql(3) + pL) !enthalpie gauche puis droite
hR = rrhor * (Qr(3) + pR)
sq_rho = sqrt(Qr(1) * rrhoL)
rrho = 1.0d0 / (sq_rho + 1.0d0)
hm = rrho * (hL + sq_rho * hR) !enthalpie moyenne
um = rrho * (uR + sq_rho * uR) !vitesse moyenne
usq = 0.5d0 * um * um !vitesse au carre
am_sq = (gamma - 1.0d0) * (hm - usq) !carre vitesse du son
am = sqrt(am_sq) !vitesse du son


!calcul du flux
F(1) = Ql(2) + Qr(2)
F(2) = Ql(2) * uL + Qr(2) * uR + pL + pR
F(3) = Ql(2) * hL + Qr(2) * hR


!Entropy fix
if (.true.) then
  lambda(-1) = um-am
  lambda( 0) = um
  lambda( 1) = um+am
  uk = Ql(2)*rrhoL
  ak = sqrt(gamma*pL*rrhoL)
  aL(-1) = uk-ak
  aL( 0) = uk
  aL( 1) = uk+ak
  uk = Qr(2)*rrhoR
  ak = sqrt(gamma*pR*rrhoR)
  aR(-1) = uk-ak
  aR( 0) = uk
  aR( 1) = uk+ak
  call EntropyFix(lambda,aL,aR)
else
  lambda(-1) = abs(um-am)
  lambda( 0) = abs(um)
  lambda( 1) = abs(um+am)
end if


!calcul
Xi(:) = (/-um,1.0d0,0.0d0/)
alpha(:) = (/usq,-um,1.0d0/)


Vect1(:) = (/1.0d0,um,Hm/) !pour calcul AFJ
Vect2(:) = (/0.0d0,1.0d0,um/) !pour calcul AFJ


deltaQ(:) = Qr(:) - Ql(:)


eta_pos = (lambda(1) + lambda(-1))*0.5d0-lambda(0)
eta_neg = (lambda(1) - lambda(-1))*0.5d0


a1 = alpha(1)*deltaQ(1) + alpha(2)*deltaQ(2) + alpha(3)*deltaQ(3)
a2 = Xi(1)*deltaQ(1) + Xi(2)*deltaQ(2) + Xi(3)*deltaQ(3)


a3 = eta_pos * (gamma-1.0d0) * a1 / am_sq + eta_neg * a2 / am
a4 = eta_pos * a2 + eta_neg * (gamma-1.0d0) * a1 / am


AFJ(:) = lambda(0)*deltaQ(:) + a3 * Vect1(:) + a4 * Vect2(:)
```

```
    Fint(:) = 0.5d0*(F(:) - Afj(:))



    end subroutine
```

— EntropyFix :

```
    subroutine EntropyFix(ak,aL,aR)
    ! entropy fix as per Harten, Hyman, J. Comput. Phys., 50 (1983).


          implicit none

          double precision, intent(in)    :: aL(3),aR(3)
          double precision, intent(inout) :: ak(3)

          integer :: i
          double precision :: dlt(3),akL(3),akR(3),coef

          dlt(:) = max(0.0d0,ak(:)-aL(:),aR(:)-ak(:))
          akL(:) = ak(:)-dlt(:)
          akR(:) = ak(:)+dlt(:)
          coef = -2.0d0


          do i=1,3
            if(akL(i).lt.0.0d0 .and. akR(i).gt.0.0d0) then
              ak(i) = ((akR(i)+akL(i))*ak(i)+coef*akR(i)*akL(i)) &
                    / (akR(i)-akL(i))
            else
              ak(i) = abs(ak(i))
            end if
          end do


          end subroutine EntropyFix
```

— compresid :

```
    subroutine compresid(Us,dF)
    use globals
    implicit none

    integer :: i,j
    double precision, intent(in) :: Us(3,N,Nelem)
    double precision, intent(out) :: dF(3,N,Nelem)


    double precision,allocatable, save :: Fnmil(:,:,:),Fnint(:,:),Ftn(:,:,:),Uf
        (:,:,:)
    double precision :: Qr(3),Ql(3)


    if(.not.allocated(Fnint)) then

      allocate(Fnmil(3,N-1,Nelem),Fnint(3,Nelem+1),Ftn(3,N+1,Nelem),Uf(3,N+1,Nelem))


    end if


    dF = 0.0d0
```

```
do i=1,Nelem
  !interpolation au point flux
  Uf(1,:,i) = matmul(Lmat(:,:),Us(1,:,i))
  Uf(2,:,i) = matmul(Lmat(:,:),Us(2,:,i))
  Uf(3,:,i) = matmul(Lmat(:,:),Us(3,:,i))


  !calcul du flux au milieu de l'element
  if (N>1) then
    call fluxINterieur(Uf,Fnmil(:,:,i))
  end if
end do


if (test.eq.1) then
  !flux a l'interface au bord avec conditions symetrique
  Qr(1) = Uf(1,N+1,Nelem)
  Qr(2) = -Uf(2,N+1,Nelem)
  Qr(3) = Uf(3,N+1,Nelem)
  Ql(1) = Uf(1,1,1)
  Ql(2) = -Uf(2,1,1)
  Ql(3) = Uf(3,1,1)
end if


if (test.eq.2) then
  !flux a l'interface au bord avec conditions periodiques
  Qr(:) = Uf(:, 1, 1)
  Ql(:) = Uf(:, N+1, Nelem)
end if


if (test.eq.3) then
  !flux a l'interface au bord avec conditions entree a gauche / sortie a droite
  Qr(1) = Uf(1,N+1,Nelem)
  Qr(2) = Uf(2,N+1,Nelem)
  Qr(3) = 1.0d0 / (gamma - 1.0d0) + 0.5d0 * (Qr(2)**2) / Qr(1)


  Ql(1) = 3.857143d0
  Ql(2) = 2.629369d0 * Ql(1)
  Ql(3) = Uf(3,1,1) - 0.5d0 * (Uf(2,1,1)**2) / Uf(1,1,1) + 0.5d0 * (Ql(2)**2) /
    Ql(1)
end if


call fluxinterface(Ql,Uf(:,1,1),Fnint(:,1))


call fluxinterface(Uf(:,N+1,Nelem),Qr,Fnint(:,Nelem+1))


!flux aux interfaces centrales
do i=2,Nelem
  call fluxinterface(Uf(:,N+1,i-1),Uf(:,1,i),Fnint(:,i))
end do


!assemblage des flux
do i=1, Nelem
  Ftn(:,1,i) = Fnint(:,i)
  do j=2, N
    Ftn(:,j,i) = Fnmil(:,j,i)
  end do
  Ftn(:,N+1,i) = Fnint(:,i+1)
end do


do i=1,Nelem
  dF(1,:,i) = -matmul(Mmat(:,:),Ftn(1,:,i))
```

```
    dF(2,:,i) = -matmul(Mmat(:,:),Ftn(2,:,i))
    dF(3,:,i) = -matmul(Mmat(:,:),Ftn(3,:,i))
  end do


  end subroutine
```

— rk3 :

```
  subroutine rk3(Un,Us)
      use globals


      implicit none


      double precision, intent(inout) :: Un(3,N,Nelem),Us(3,N,Nelem)
      integer :: i,j,k
      double precision :: a(3),b(3),c(3),ini_time,P,Vit_son,ulim


      parameter(a=(/ 1.0d0, 1.0d0/4.0d0, 2.0d0/3.0d0 /))
      parameter(b=(/ 0.0d0, 3.0d0/4.0d0, 1.0d0/3.0d0 /))
      parameter(c=(/ 1.0d0,       0.5d0,       1.0d0 /))


      double precision, allocatable, save :: Unini(:,:,:),resid(:,:,:)


      if(.not.allocated(Unini)) then
        ! on fait l'allocation de 'Qini' et 'resid'
        allocate(Unini(3,N,Nelem),resid(3,N,Nelem))
      end if


      Unini = Un
      ini_time = time


      do i=1,3
  ! ici on calcule le residu a chaque point solution dans tous les elements.
        call compresid(Us,resid)
        !calcul du pas de temps dt
        dt = 1.0d300
        do j=1,Nelem
          do k=1,N
            P = (gamma-1) * (Us(3,k,j) - 0.5d0 * (Us(2,k,j)**2) / Us(1,k,j))
            vit_son = sqrt(gamma * P / Us(1,k,j))
            ulim = abs(Us(2,k,j) / Us(1,k,j)) + vit_son
            dt = min(dt, cft*h/ulim)
          end do
        end do
        dt = min(dt, t_final - time)


        !iteration sur le temps
        Un = a(i) * (Un + dt * resid) + b(i) * Unini
        time = ini_time + c(i) * dt


        !passage a une solution physique
        do j=1,Nelem
          Us(:,:,j) = InvJ(j)*Un(:,:,j)
        end do
      end do


      end subroutine
```