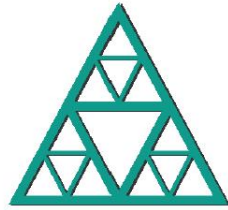


ECOLE NATIONALE DES PONTS ET CHAUSSEES



PROJET D'INITIATION A LA RECHERCHE 2018

**Réduction de modèles appliquée à la modélisation du
transport de polluants**

Léo Baty, Clément Lasuen, Chiheb Eddine Najjar,
Nathan Godey, Régis Santet, Song Phuc Duong
sous la direction de
Damiano Lombardi, Sébastien Boyaval
Laboratoire INRIA

Table des matières

1	Introduction	5
2	Matériels et méthodes	5
2.1	Méthode POD : <i>Proper Orthogonal Decomposition</i>	5
2.2	Paramétrage	6
2.3	Approche Eulérienne	7
2.3.1	Description	7
2.4	Méthode des volumes finis	8
2.5	Approche Lagrangienne	9
2.5.1	Description	9
2.5.2	Résolution numérique	10
2.5.3	Interpolation	10
2.5.4	Champ de vitesse constant	11
2.5.5	Ecoulement cellulaire	11
3	Résultats	11
3.1	Approche Eulerienne	11
3.1.1	Cas vitesse constante	11
3.1.2	Cas des écoulements cellulaires	12
3.2	Approche Lagrangienne	14
3.2.1	Vitesse constante	14
3.2.2	Écoulement cellulaire	16
3.2.3	Champs de Lamb-Oseen	16
4	Discussion	17
4.1	Approche Lagrangienne	17
4.1.1	Vitesse constante	17
4.1.2	Écoulement cellulaire	17
4.1.3	Champ de Lamb-Oseen	17
5	Conclusions	18
6	Appendice	18
6.1	Méthode des volumes finis [2]	18
6.1.1	Maillage	18
6.1.2	Schéma numérique	19
7	Annexes	20
7.1	Création du maillage	20
7.2	Approche Eulerienne	25
7.2.1	Calcul du flux : solveur de Lax-Friedrichs Riemann	25
7.2.2	Volumes finis - cas constant	25
7.2.3	Decomposition SVD	29

8	Remerciements	30
9	Exemples de figures, tableau, équations...	30

ABSTRACT : Predicting the behavior of a pollutant in a fluid is one of the major problems today, it is very complex to do, as there are too many variables to describe the phenomenon and achieve good precision overtime. We have to use reduction models such as POD (Proper Orthogonal Direct) in order to compute the solution in a reasonable time. It involves keeping the major dynamic of the solution with a few degrees of liberties and accepting an error between the approximation and the real solution. We use the advection equation to simulate a pollution peak represented by a Gaussian function in a fluid where the fluid speed varies in a closed region. After computing a couple of solutions using our initial integration schemes for some parameters like the finite volumes scheme and explicit Euler scheme to simulate the fundamental dynamic, we use the POD algorithm to compute approximate solutions for a new set of parameters, this time in a much shorter period. We decided to approach the problem with two different methods. The Eulerian method, in which we consider the velocity as a function of space and time, and try to compute the concentration of the pollutant as a function of the same variables. Whereas in the Lagrangian method, we follow some particles arraigned initially in a random way according to a probability law adapted to the initial concentration and compute their trajectory as a function of time and the initial location. Thus, we will try to compare both methods to see which one suits the problem better.

KEYWORDS : Mathematical Models, Numerical Simulations, Computational Complexity, Model Order Reduction, Proper Orthogonal Decomposition, Singular Value Decomposition

1 Introduction

Le but de ce projet d'initiation à la recherche est d'observer numériquement l'évolution d'une concentration de polluants (fluide que l'on considère incompressible) dans un liquide (rejets dans une rivière) ou dans un gaz (nuage radioactif). Ce projet est associé au cours intitulé "Méthodes numériques pour les problèmes en grandes dimensions" [1] tenu en première année à l'École nationale des ponts et chaussées par Damiano Lombardi (INRIA) et Sébastien Boyaval (INRIA). Nous allons donc chercher à utiliser des outils de réductions de modèles afin de pouvoir prédire, étant donné une concentration initiale de polluant en un lieu donné, l'évolution de la concentration de façon fiable. La dynamique de la solution réelle dépend d'un nombre de paramètres beaucoup trop grand si l'on souhaite la simuler tout le temps avec des paramètres différents à chaque fois (position initiale, champs de vitesse,...). La discrétisation du problème pose aussi problème, avec un phénomène de diffusion souvent inévitable. Le but de la méthode de réduction de modèles est de pouvoir se ramener à une base de solutions qui contiennent en elle-même la majeure partie de la dynamique de la solution réelle pour un certain nombre de paramètres où on aura la solution fine, afin de pouvoir simuler de façon approchée et surtout plus rapide des nouveaux schémas avec des paramètres différents. Il faut donc d'abord penser à l'approche que l'on va avoir sur la discrétisation du problème, puis savoir quel outil de la réduction de modèle utiliser afin d'avoir une erreur relativement faible pour un temps relativement court : un compromis est donc fait entre temps de calcul et précision.

2 Matériels et méthodes

Tout notre projet repose sur l'équation dite d'*advection*, ou de *transport de matière* :

$$\partial_t c + u \nabla c = 0. \quad (1)$$

où c est la concentration de polluant et u est le champs de vitesse présent dans le liquide ou le gaz. Pour une simulation numérique, nous devons nous munir d'un code donnant un maillage de la zone que nous voulons étudier, d'un schéma de discrétisation en temps et en espace de l'équation 1, et d'une condition initiale.

Une fois ce code mis en place, nous utilisons l'algorithme POD (Proper Orthogonal Decomposition) pour réduire notre solution. Nous nous posons alors trois questions :

- L'algorithme POD est-il adapté à l'équation d'advection ?
- Quelle approche, lagrangienne ou eulérienne, convient-il de prendre ?
- Peut-on trouver un moyen d'augmenter la compression de données ?

Nous allons explorer ces questions à travers deux exemples : un écoulement à vitesse constant (où juste l'angle d'incidence change) et un écoulement cellulaire (des tourbillons).

2.1 Méthode POD : *Proper Orthogonal Decomposition*

L'inconnue du problème est notée $X(t) \in \mathbb{R}^N$ pour $t \in [0, T]$. La première étape de cette méthode consiste à calculer des valeurs de la solution $X(t_0), X(t_1), \dots, X(t_n)$ à des instants t_0, t_1, \dots, t_n , appelées snapshots. Ces vecteurs sont ensuite classés dans une matrice M :

$$M = \begin{pmatrix} X(t_0) & | & X(t_1) & | & \cdots & | & X(t_n) \end{pmatrix}$$

La décomposition SVD (Singular Value Decomposition) de la matrice M s'écrit : $M = USV^T$. Où U et V sont des matrices orthogonales respectivement de taille N et n . La matrice S est diagonale, ses coefficients sont les valeurs singulières de M (les valeurs propres de $M^T M$). Cette décomposition est unique lorsque les coefficients sur la diagonale de S , notés $\sigma_1, \dots, \sigma_p$ avec $p = \text{rang}(S) = \text{rang}(M)$, sont ordonnés par ordre décroissant.

Les colonnes de U , notées $U^{(1)}, \dots, U^{(N)}$ sont les modes propres de la matrice (ce sont les vecteurs propres de $M^T M$). Soit r un entier compris entre 1 et le rang de S . Les vecteurs $U^{(1)}, \dots, U^{(r)}$ forment alors une base orthonormée d'un sous-espace vectoriel de \mathbb{R}^N dans lequel on projette la dynamique du système.

Cette méthode s'applique lorsque la dynamique du système reste proche d'un sous-espace vectoriel de dimension faible. Cela correspond à une décroissance rapide des valeurs singulières de M . L'erreur de représentation est donnée par : $\sum_{i=r+1}^p \sigma_i^2$.

2.2 Paramétrage

Dans tout ce qui suit, nous avons opté comme maillage un quadrillage de la zone $(0, 1)^2$, et le nombre de cases sur le maillage est égal à $(2 \times 10^8 + 1)^2$: c'est un cas où le maillage est plutôt fin et le temps de calcul, bien qu'un peu long, reste raisonnable.

On effectue un nombre $n_s = 16$ de simulations différentes dans chaque cas afin de pouvoir remplir la matrice qui va nous servir pour faire l'algorithme POD (matrice contenant quelques solutions fines du problème).

On prend un temps maximal de simulation égal $T_{max} = 1$.

Lors de l'algorithme POD, on se fixe un nombre nb_{modes} que l'on choisit tel que l'erreur sur la solution soit environ de 10^{-3} . Cette erreur a été choisie a posteriori, étant un compromis entre le temps de calcul (nombre de modes) et l'erreur que la précision de l'algorithme.

Cas d'une vitesse constante Dans le cas où la vitesse est constante, seule sa direction varie et est donnée par un angle $\theta \in [0, \frac{\pi}{2}]$ (on prend ces angles pour éviter que la gaussienne n'atteigne le bord). On a donc

$$\begin{cases} u_x = ||u|| \cos(\theta) \\ u_y = ||u|| \sin(\theta) \end{cases}$$

Cas d'un écoulement cellulaire Dans ce cas, le champ de vitesse dérive du potentiel suivant :

$$\psi(x, y) = \sin(2\pi x) \sin(2\pi y) + \theta_0 \cos(2\pi \theta_1 x) \cos(2\pi \theta_2 y)$$

où $\theta_0 \in [0, 0.75]$ et $(\theta_1, \theta_2) \in [0.5, 4]^2$. La vitesse s'exprime alors :

$$\begin{cases} u_x(x, y) = \frac{\partial \psi}{\partial y} = 2\pi \sin(2\pi x) \cos(2\pi y) - \theta_0 2\pi \theta_2 \cos(2\pi \theta_1 x) \sin(2\pi \theta_2 y) \\ u_y(x, y) = -\frac{\partial \psi}{\partial x} = -2\pi \sin(2\pi y) \cos(2\pi x) + \theta_0 2\pi \theta_1 \cos(2\pi \theta_2 y) \sin(2\pi \theta_1 x) \end{cases}$$

Champ de vitesse de Lamb-Oseen Nous prenons dans ce cas une vitesse analytique donnée par :

$$\mathbf{V}(r, \theta, t) = \frac{\Gamma}{2\pi r} \left(1 - \exp\left(\frac{-r^2}{4\nu t + r_c^2}\right) \right) \mathbf{u}_\theta$$

où ν désigne la viscosité cinématique, Γ est la circulation de l'écoulement et r_c le rayon moyen.

2.3 Approche Eulérienne

2.3.1 Description

L'approche Eulérienne consiste à déterminer en chaque point de notre maillage les valeurs de la concentration en polluant. On va donc, à chaque itération de l'algorithme, mettre à jour l'ensemble du maillage.

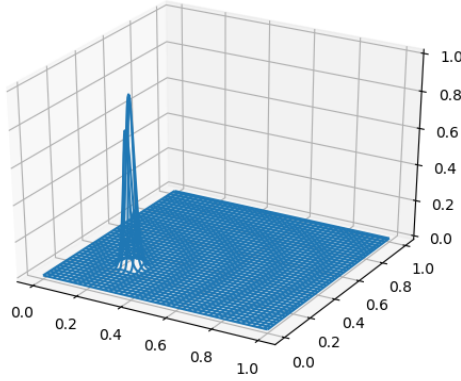
Il se trouve qu'une simple approche par la méthode des différences finies n'est pas un schéma stable pour notre étude : le schéma est un peu trop instable (concentration pouvant devenir négative). On va lui préférer la méthode des volumes finis avec un schéma de Lax-Friedrichs, qui est un schéma qui amène un peu plus de phénomène de diffusion, mais qui assure la positivité de la concentration. Pour plus de détails, voir la section 2.4.

Nous avons à notre disposition un code d'éléments finis 2D sous Scilab faisant tourner le schéma de Lax-Friedrichs sur des cas très compliqués et un maillage uniforme sur $(0, 1)^2$. En déchiffrant une partie de ce code, et en s'y inspirant, nous avons codé ce schéma en Python, de façon simplifiée et en retenant seulement ce qui allait nous servir pour notre étude (maillage, fonction permettant l'actualisation, affichage de la solution). La zone étudiée est toujours $(0, 1)^2$, et notre condition initiale est représentée par une gaussienne étroite (équivalente à un pic de polluant). De plus, les conditions aux bords sont périodiques : nous nous plaçons sur une sphère afin de rendre le code plus simple, même si nous ferons en sorte de ne jamais atteindre les bords. Ce code Python est donné en annexe.

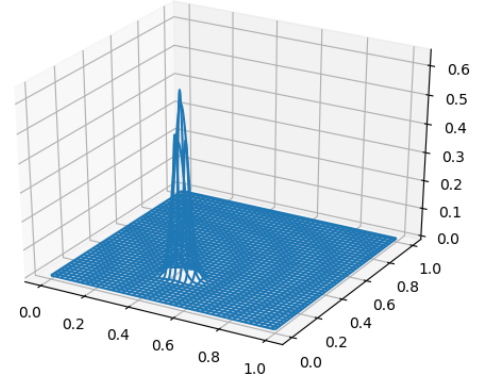
Notre condition initiale est donnée par une gaussienne :

$$c_0(x, y) = \exp\left(-\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right)$$

avec $(x_0, y_0) = (0.25, 0.25)$, $\sigma = \frac{1}{50}$. L'approche eulérienne consiste donc à transporter cette gaussienne le long du maillage.



Condition initiale...



...et après un certain temps

On stocke les valeurs de la solution au cours du temps sur tout le maillage dans une matrice à laquelle on va appliquer l'algorithme POD. Après un choix adapté du nombre de modes, on va pouvoir simuler la dynamique de notre système avec un nombre de degré de libertés bien plus faible.

2.4 Méthode des volumes finis

Le but de cette méthode est d'implémenter un schéma qui permet d'approcher la solution faible de l'équation :

$$\frac{\partial c}{\partial t} + \text{div}(c u) = 0 \quad (2)$$

de manière à assurer la convergence et la stabilité (voir le paragraphe Critères de convergence et de stabilité).

Nous regardons cette équation dans un compact en temps et en espace $[0, T] \times [0, L]^2$ Pour cela nous considérerons un maillage en espace rectangulaire uniforme $K_{i,j}$ de pas Δx , Δy , et un maillage en temps de pas Δt_n qui dépend *a priori* de l'instant t_n . La méthode des volumes finis cherche à exprimer la valeur moyenne de la concentration c sur chaque cellule $K_{i,j}$ en fonction de la valeur moyenne cellulaire la vitesse est des différents paramètres du maillage à savoir le pas et le volumes des cellules en d'autres termes on exprime :

$$c_{i,j}(t_n) = \frac{1}{|K_{i,j}|} \int_{K_{i,j}} c(t_n, X) dX$$

en fonction de $u_{i,j}(t_n)$, Δx , Δy et Δt_n pour cela nous intégrons l'équation (2) sur la cellule $K_{i,j}$ à l'instant t_n nous obtenons :

$$\frac{\partial c_{i,j}}{\partial t}(t_n) + \frac{1}{|K_{i,j}|} \int_{\partial K_{i,j}} c u \cdot n dL = 0$$

nous approchons la dérivé temporelle par un schéma d'Euler explicite et le flux par un schéma de Lax-Friedrichs : Pour toute cellule $K_{i,j}$

$$c_{i,j}(t_n) = c_{i,j}(t_{n-1}) - \frac{t_n - t_{n-1}}{|K_{i,j}|} \sum_{k,l} |\Gamma_{i,j}^{k,l}| g(c_{i,j}(t_{n-1}), c_{k,l}(t_{n-1})) \quad \forall n \quad (3)$$

où $\Gamma_{i,j}^{k,l} = K_{i,j} \cap K_{k,l}$ et :

$$\begin{cases} \lambda_{i,j \rightarrow k,l} = \max\{u_{i,j} \cdot n_{i,j \rightarrow k,l}, u_{k,l} \cdot n_{i,j \rightarrow k,l}\} \\ g(c_{i,j}, c_{k,l}) = \frac{1}{2}(c_{i,j}u_{i,j} + c_{k,l}u_{k,l}) \cdot n_{i,j \rightarrow k,l} + \frac{1}{2\lambda_{i,j \rightarrow k,l}}(c_{i,j} - c_{k,l}) \end{cases} \quad (4)$$

à l'instant t_n . Pour plus de détails voir la section (6.1).

Critères de convergence et de stabilité Lorsque on part d'une condition initiale bornée l'équation dans le sens faible (2) admet une unique solution entropique, c'est à dire la solution vérifie : $\forall S \in \mathcal{C}^2$ convexe, on a

$$\frac{\partial S(c)}{\partial t} + \text{div}(G(c)) \leq 0$$

où G est un flux d'entropie qui vérifie $G' = S' u$.

Le schéma numérique (3) converge vers cette solution (lorsque $\Delta x, \Delta y, \Delta t_n \rightarrow 0$) dès qu'il vérifie l'inégalité entropique discrétisée : $\forall S \in \mathcal{C}^2$ convexe, $\exists \Phi(c_1, c_2)$ telle que

$$\begin{cases} \Phi(c, c) = G(c) \\ \frac{S(c_{i,j}(t_n)) - S(c_{i,j}(t_{n-1}))}{t_n - t_{n-1}} + \sum_{k,l} \frac{|\Gamma_{i,j}^{k,l}|}{|K_{i,j}|} \Phi(c_{i,j}(t_{n-1}), c_{k,l}(t_{n-1})) \leq 0 \quad \forall i, j, n \end{cases} \quad (5)$$

Ainsi l'entropie doit être dissipée ou conservée, sa augmentation est une source d'instabilité du schéma.

Pour que le schéma des volumes finis (3) soit stable il faut satisfaire la condition CFL (Courant, Friedrichs, Lewy) qui s'écrit :

$$t_n - t_{n-1} < \frac{|K_{i,j}|}{2|\Gamma_{i,j}^{k,l}|\lambda_{i,j \rightarrow k,l}(t_{n-1})} \quad \forall i, j, k, l$$

Cette condition nous permet de choisir le pas de temps adapté à la vitesse et au maillage à chaque instant pour assurer la stabilité.

2.5 Approche Lagrangienne

2.5.1 Description

L'approche lagrangienne consiste à suivre les particules le long de leur trajectoires. Cela revient à résoudre :

$$\begin{cases} \partial_t X = v(X(\xi, t), t) \\ X(\xi, 0) = \xi \end{cases}$$

Où $X(\xi, t) \in \mathbb{R}^{n \times 2}$, n étant le nombre de particules, $t \in [0, T]$ et ξ étant les positions initiales des particules.

2.5.2 Résolution numérique

Nous avons utilisé le schéma de Crank-Nicholson :

$$X^{(k+1)} = X^{(k)} + \frac{\Delta t}{2}(v(X^{(k)}, t^k) + v(X^{(k+1)}, t^{k+1}))$$

Nous avons utilisé une méthode du point fixe pour évaluer $X^{(k+1)}$ en connaissant $X^{(k)}$. Initialisation ($r=0$) :

$$\begin{aligned} X_0^{(k+1)} &= X^{(k)} \\ X_1^{(k+1)} &= X^{(k)} + \Delta t v(X^{(k)}, t^k) \\ X_{r+1}^{(k+1)} &= X^{(k)} + \frac{\Delta t}{2}(v(X^{(k)}, t^k) + v(X_r^{(k+1)}, t^{k+1})) \end{aligned}$$

Et on arrête lorsque :

$$\|X_{r+1}^{(k+1)} - X_r^{(k+1)}\| < \epsilon$$

où ϵ est une erreur donnant un critère d'arrêt : on suppose qu'on est "suffisamment proche de la solution" quand l'algorithme s'arrête.

2.5.3 Interpolation

La résolution numérique du problème nécessite de connaître le champ de vitesse des particules, noté v . Nous supposons que l'ensemble $\Omega = [-1, 1]^2$ soit muni d'un maillage carré régulier de pas noté Δx en abscisse et Δy en ordonnée. Nous supposons également que le champ de vitesse est connu en tout point du maillage. Nous présentons ici la méthode d'interpolation que nous avons utilisé pour approximer le champ de vitesse dans l'ensemble du domaine Ω .

Considérons une maille dont les coordonnées du coin inférieur gauche sont notées (x^{LL}, y^{LL}) . Soit un point (x, y) appartenant à cette maille. On pose :

$$\begin{cases} \tilde{x} = \frac{x - x^{LL}}{\Delta x} \\ \tilde{y} = \frac{y - y^{LL}}{\Delta y} \end{cases}$$

On définit :

$$\begin{cases} \varphi_1(\tilde{x}, \tilde{y}) = (1 - \tilde{x})(1 - \tilde{y}) \\ \varphi_2(\tilde{x}, \tilde{y}) = \tilde{x}(1 - \tilde{y}) \\ \varphi_3(\tilde{x}, \tilde{y}) = \tilde{x}\tilde{y} \\ \varphi_4(\tilde{x}, \tilde{y}) = (1 - \tilde{x})\tilde{y} \end{cases}$$

La vitesse au point (x, y) à l'instant t est alors donnée par :

$$\begin{aligned} v(x, y, t) &= v(x^{LL}, y^{LL}, t)\varphi_1(\tilde{x}, \tilde{y}) + v(x^{LL} + \Delta x, y^{LL}, t)\varphi_2(\tilde{x}, \tilde{y}) \\ &+ v(x^{LL} + \Delta x, y^{LL} + \Delta y, t)\varphi_3(\tilde{x}, \tilde{y}) \\ &+ v(x^{LL}, y^{LL} + \Delta y, t)\varphi_4(\tilde{x}, \tilde{y}) \end{aligned}$$

2.5.4 Champ de vitesse constant

Cette fois, les particules sont distribuées selon une gaussienne centrée en $(0.25; 0.25)$ de variance $1/50$. Leurs vitesses sont identiques, constante et orientées avec un angle θ compris entre 0 et $\frac{\pi}{2}$. Les positions des différentes particules sont ensuite stockées dans une matrice M de taille $2nb_{particules} \times nb_{steps}$. Nous avons ensuite effectué la décomposition SVD de la matrice M : $M = USV^T$

2.5.5 Ecoulement cellulaire

Dans cette simulation : $\theta_0 = 0.2, \theta_1 = 3.12, \theta_2 = 2.69$.

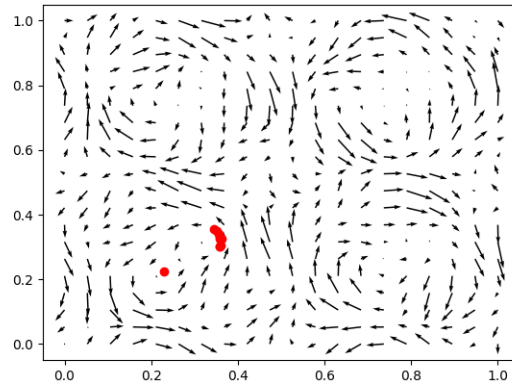


FIGURE 1 –

La décroissance des valeurs singulières est assez lente, comme le montre la figure suivante. Cette méthode n'est pas efficace dans cette situation.

3 Résultats

3.1 Approche Eulerienne

3.1.1 Cas vitesse constante

Voici d'abord le tracé du log des valeurs singulières en fonction de leur rang dans la décomposition SVD.

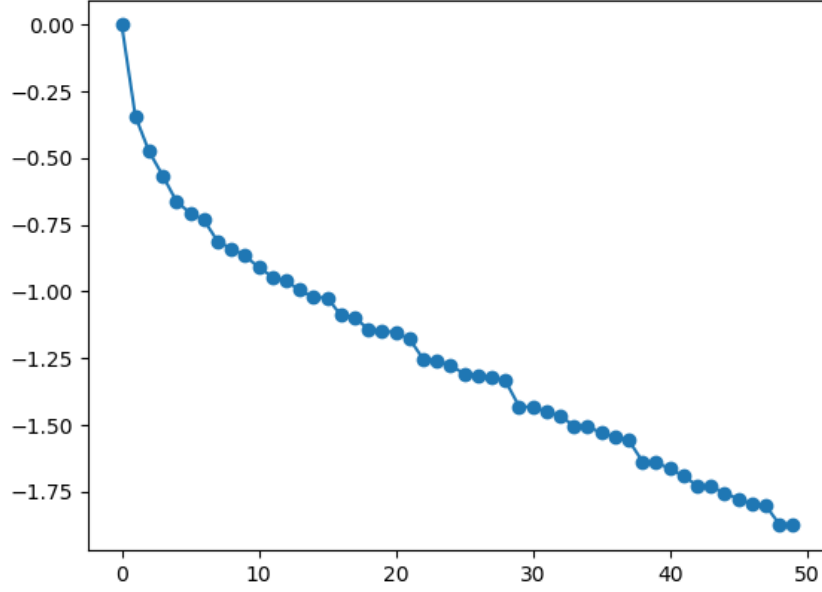


FIGURE 2 – Tracé du log des valeurs singulières en fonction du rang

Ceci nous permet de choisir le nombre de modes qu'on va utiliser, l'erreur étant fixée à 10^{-3} : on choisit $nb_{modes} = 13$.

On récupère donc l'approximation de la matrice de *snapshots* avec 13 vecteurs propres, et on va pouvoir simuler la dynamique de notre système assez fidèlement.

3.1.2 Cas des écoulements cellulaires

On considère le champ de vitesse défini par la fonction potentiel suivante :

$$\psi(x, y) = \sin(2\pi x) \sin(2\pi y) + \nu_0 \cos(2\pi\nu_1 x) \cos(2\pi\nu_2 y)$$

Avec $\nu_0 \in [0, 0.75]$, $(\nu_1, \nu_2) \in [0.5, 4]^2$

D'où :

$$\begin{cases} u_x = \frac{\partial \psi}{\partial y} = 2\pi \cos(2\pi y) \sin(2\pi x) - 2\pi \nu_y \nu_0 \cos(2\pi \nu_1 y) \sin(2\pi \nu_1 x) \\ u_y = -\frac{\partial \psi}{\partial x} = -2\pi \cos(2\pi x) \sin(2\pi y) + 2\pi \nu_x \nu_0 \cos(2\pi \nu_1 x) \sin(2\pi \nu_1 y) \end{cases}$$

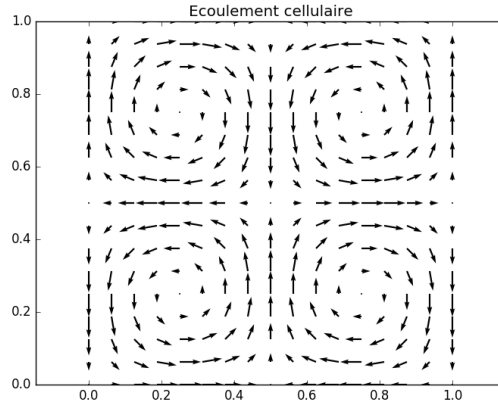


FIGURE 3 –

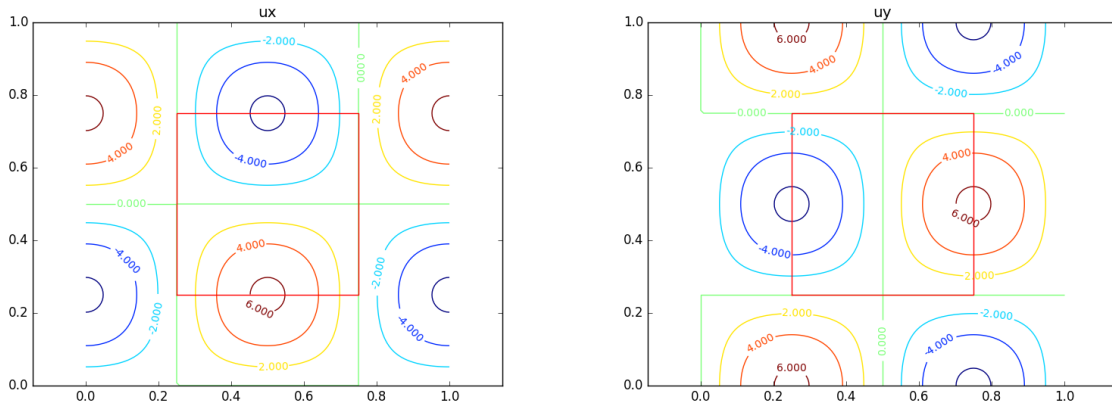


FIGURE 4 –

De plus, on utilise la condition initiale suivante :

$$\begin{cases} c(\underline{x}, 0) = \exp\left(-\frac{1}{k^2 - |\underline{x} - \underline{x}^{(0)}|}\right) & \text{si } |\underline{x} - \underline{x}^{(0)}| \leq k \\ c(\underline{x}, 0) = 0 & \text{si } |\underline{x} - \underline{x}^{(0)}| > k \end{cases}$$

On résouds l'équation de transport pour différents ν_0 , ν_x et ν_y , ainsi que pour différents k et $\underline{x}^{(0)}$ (24 différents quintuplets $(\nu_0, \nu_x, \nu_y, k, \underline{x}^{(0)})$ au total), pendant $T = 0.1$. On a pris un pas de maillage de $\frac{1}{27}$, i.e. deux fois moins fin que pour un champ de vitesse constant, sans quoi le temps de calcul aurait été trop important. Après deux heures de simulation, on obtient une matrice M des snapshots de taille 27^2 , et qui pèse 1.38 Go.

On a effectué la SVD (décomposition en valeurs singulières) de M , puis on a tracé les 50 premières valeurs singulières :

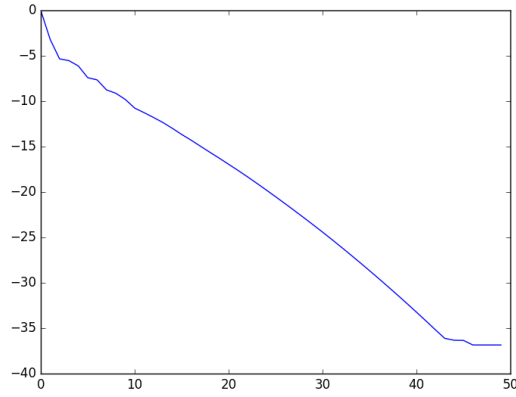


FIGURE 5 – Tracé du log des 50 premières valeurs singulières

On observe une décroissance exponentielle des valeurs singulières, ce qui indique que la POD sera efficace dans le cas des écoulements cellulaires. On peut donc sélectionner les 50 premiers vecteurs de la base afin de résoudre l'équation pour d'autres jeux de paramètres avec une durée de simulation beaucoup plus rapide, et un espace de stockage beaucoup moins important (6.7 Mo).

POD

3.2 Approche Lagrangienne

3.2.1 Vitesse constante

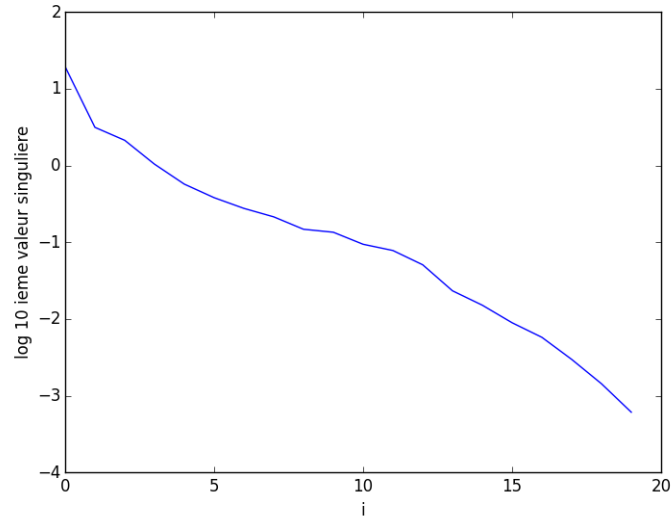


FIGURE 6 – Valeurs singulières pour 10 particules

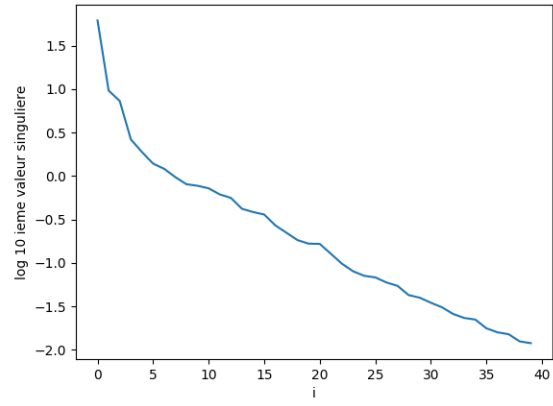


FIGURE 7 – Valeurs singulières pour 100 particules

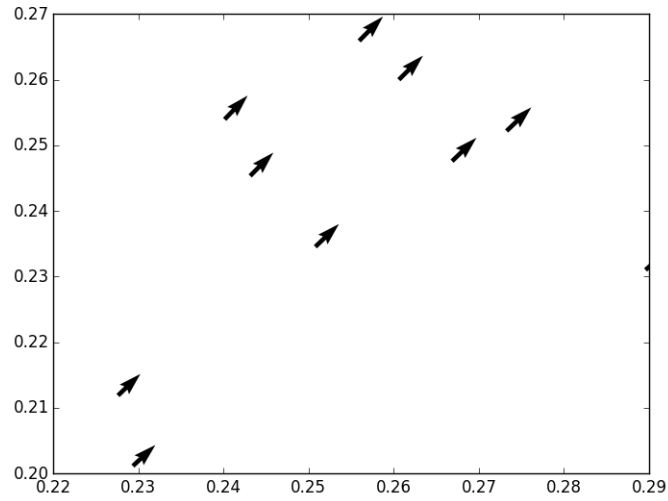


FIGURE 8 – Projection suivant le premier mode

3.2.2 Écoulement cellulaire

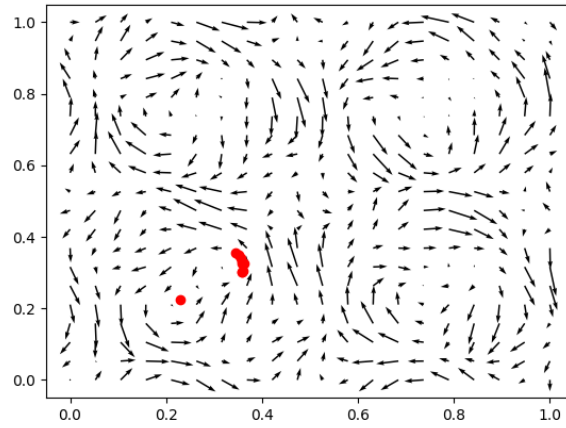


FIGURE 9 – 10 particules dans un écoulement cellulaire

3.2.3 Champs de Lamb-Oseen

Dans cette simulation : $\Gamma = 10$, $\nu = 0.5$ et $r_c = 0.7$.

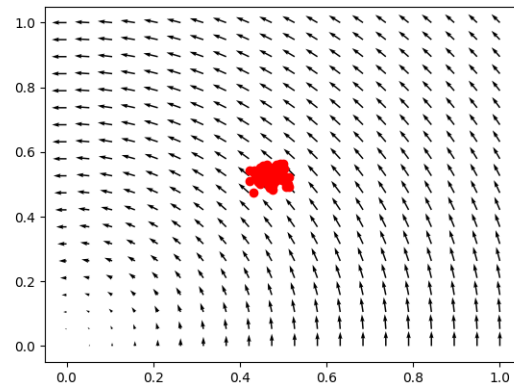


FIGURE 10 – 100 particules dans un champs de Lamb-Oseen

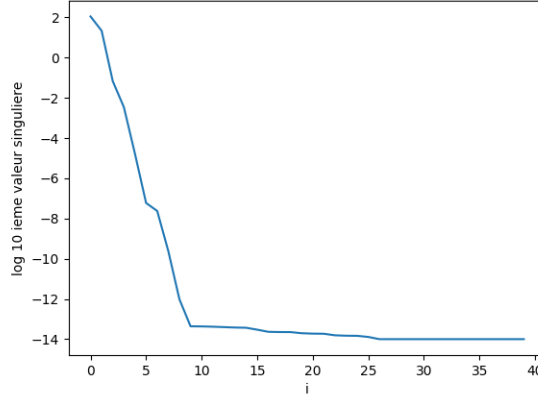


FIGURE 11 – Valeurs singulières pour 100 particules

4 Discussion

4.1 Approche Lagrangienne

4.1.1 Vitesse constante

Dans ce cas particulier, la projection suivant le premier mode propre (première colonne de U) permet d'obtenir une approximation satisfaisante. Comme on peut le voir sur les figures, la décroissance des valeurs singulières est très rapide. Cela semble normal puisque les particules possèdent une trajectoire rectiligne uniforme. Cependant, ce mode propre dépend de l'angle θ qui caractérise la vitesse des particules. Si ce paramètre est modifié alors le mode propre est également modifié. Il n'est donc pas possible de réduire efficacement ce modèle.

4.1.2 Écoulement cellulaire

La décroissance des valeurs singulières est assez lente. Un grand nombre de modes propres est donc nécessaire pour donner une bonne approximation de la solution.

4.1.3 Champ de Lamb-Oseen

La décroissance des valeurs singulières est rapide. Un faible nombre de modes propres est par conséquent nécessaire pour obtenir une approximation convenable.

5 Conclusions

6 Appendice

6.1 Méthode des volumes finis [2]

Dans le cas de l'approche eulérienne, on utilise un schéma qui est différent de ceux utilisés jusqu'à présent dans nos cours. Détaillons un peu comment cette méthode fonctionne.

Considérons l'équation de transport dans le cas général en dimension $d > 0$ sur un compact en temps et en espace $[0, T] \times [a, b]^d$ pour $T \in \mathbb{R}_+^*$ et $a < b$ dans \mathbb{R}

$$\frac{\partial c}{\partial t} + u \cdot \nabla c = f \quad (6)$$

où $c : [0, T] \times [a, b]^d \rightarrow [0, 1]$ la concentration du polluant, $u : [0, T] \times [a, b]^d \rightarrow \mathbb{R}^d$ le champs des vitesses imposé dans le milieu et $f : [0, T] \times [a, b]^d \rightarrow \mathbb{R}$ la source de pollution.

Dans notre étude nous prenons $f = 0$, nous considérons qu'il n'y a pas de source.

6.1.1 Maillage

Nous posons $[a, b] = [0, L]$ afin de simplifier. Maintenant, pour discrétiser l'équation (6) en espace, nous utilisons en utilisant deux suites arithmétiques $(x_i)_{i=1}^{n_x}$ sur l'axe des abscisses et $(y_i)_{i=1}^{n_y}$ sur l'axe des ordonnées, vérifiant :

$$\begin{cases} x_i = i\Delta x = i \frac{L}{n_x} \\ y_i = i\Delta y = i \frac{L}{n_y} \end{cases}$$

On obtient une grille de $(n_x - 1)(n_y - 1)$ cellules, on appelle donc $K_{i,j} = [x_i, x_{i+1}] \times [y_j, y_{j+1}]$, et $\Gamma_{i,j}^{k,l} = K_{i,j} \cap K_{k,l}$.

On discrétise $[0, T]$ en utilisant la suite $(t_i)_{i=1}^{n_t}$ qui n'est pas nécessairement arithmétique, vérifiant la condition CFL et :

$$\begin{cases} t_1 = 0 \\ t_{n_t} = T \\ t_i < t_{i+1} \quad \forall i < n_t \end{cases}$$

Ainsi, nous posons pour $1 \leq i \leq n_x$, $1 \leq j \leq n_y$ et $1 \leq n \leq n_t$:

$$\begin{aligned} X &= (x, y), \quad dX = dx dy \\ |K_{i,j}| &= (x_{i+1} - x_i)(y_{j+1} - y_j) = \Delta x \Delta y = \frac{L^2}{n_x n_y} \\ c_{i,j}(t_n) &= \frac{1}{|K_{i,j}|} \int_{K_{i,j}} c(t_n, X) dX, \\ u_{i,j}(t_n) &= \frac{1}{|K_{i,j}|} \int_{K_{i,j}} u(t_n, X) dX. \end{aligned}$$

6.1.2 Schéma numérique

Nous considérons l'équation (6) dans sa forme faible, c'est à dire au sens des distributions, en utilisons le fait que $\operatorname{div}(c u) = u \nabla c + c \operatorname{div}(u)$, nous pouvons la réécrire sous la forme suivante :

$$\frac{\partial c}{\partial t} + \operatorname{div}(c u) - c \operatorname{div}(u) = 0$$

Dans notre projet on considère des vitesse des fluides incompressibles, $\operatorname{div} u = 0$, l'équation devient :

$$\frac{\partial c}{\partial t} + \operatorname{div}(c u) = 0 \quad (7)$$

Maintenant en fixant $t_n \in [0, T]$ nous intégrons sur chaque cellule $K_{i,j}$,

$$\frac{1}{|K_{i,j}|} \int_{K_{i,j}} \left(\frac{\partial c}{\partial t} + \operatorname{div}(c u) \right) dX = 0$$

On peut permuter l'intégrale et la dérivation par rapport au temps puisque les cellules $K_{i,j}$ sont indépendantes de temps, et on utilise le théorème de Green pour des fonctions suffisamment régulières $\int_{\Omega} \operatorname{div}(a) d\Omega = \int_{\partial\Omega} a \cdot n dL$, où n est le vecteur normal sortant, ainsi :

$$\frac{\partial c_{i,j}}{\partial t}(t_n) + \frac{1}{|K_{i,j}|} \int_{\partial K_{i,j}} c u \cdot n dL = 0$$

Chaque cellule est un rectangle ces bords sont des segments, nous posons, dans chaque segment Γ la quantité $\frac{1}{|\Gamma|} \int_{\Gamma} c u \cdot n dL = F_{\Gamma}(c)$:

$$\int_{\partial K_{i,j}} c u \cdot n dL = \sum_{k,l} |\Gamma_{i,j}^{k,l}| F_{\Gamma_{i,j}^{k,l}}(c) \cdot n_{i,j \rightarrow k,l}$$

Cette somme contient bien sûr quatre termes puisque les cellules sont des rectangles.

$F_{\Gamma} \cdot n$ est le flux sortant de la cellule $K_{i,j}$ à partir de la face Γ , on cherche maintenant à l'approcher numériquement de manière stable, on choisit pour cela un schéma de flux en 2-points, c'est à dire,

$$F_{\Gamma_{i,j}^{k,l}}(c) \cdot n_{i,j \rightarrow k,l} \simeq g(c_{i,j}, c_{k,l})$$

Le flux sortant de Γ dépend de la valeur de c dans les deux cellules voisines qui contiennent Γ .

La méthode de Godunov permet de donner une fonction g qui assure la stabilité du schéma en effet, il propose pour des scalaires a et b

$$g(a, b) = F_{\Gamma_{i,j}^{k,l}}(c_R(t_n, 0)) \cdot n_{i,j \rightarrow k,l}$$

où c_R est la solution du problème du Riemann défini par :

$$\begin{cases} \text{trouver } c(t, x) \text{ telle que} \\ \frac{\partial c}{\partial t} + \text{div}(c u) = 0 \\ c(0, x) = a \mathbb{1}_{x < 0} + b \mathbb{1}_{x > 0} \end{cases} \quad (8)$$

mais vu la difficulté de la résolution du problème exact (8) nous utilisons le schéma de Lax-Friedrichs qui peut être considéré comme une méthode approchée de la méthode de Godunov, plus simple à implémenter et rapide :

$$\begin{cases} \lambda_{i,j \rightarrow k,l} = \max\{u_{i,j} \cdot n_{i,j \rightarrow k,l}, u_{k,l} \cdot n_{i,j \rightarrow k,l}\} \\ g(c_{i,j}, c_{k,l}) = \frac{1}{2}(c_{i,j}u_{i,j} + c_{k,l}u_{k,l}) \cdot n_{i,j \rightarrow k,l} + \frac{1}{2\lambda_{i,j \rightarrow k,l}}(c_{i,j} - c_{k,l}) \end{cases} \quad (9)$$

Il faut noter que le flux que nous avons utilisé g est un flux conservatif c'est à dire

$$g(c_{i,j}, c_{k,l}) + g(c_{k,l}, c_{i,j}) = 0,$$

ce qui est rassurant vu la nature conservative de l'équation (7).

On utilise pour discrétiser le terme en ∂_t un schéma d'Euler explicite :

$$\frac{\partial c}{\partial t}(t_n) \simeq \frac{c(t_n) - c(t_{n-1})}{t_n - t_{n-1}}$$

Enfin le schéma numérique à implémenter est : Pour toute cellule $K_{i,j}$

$$c_{i,j}(t_n) = c_{i,j}(t_{n-1}) - \frac{t_n - t_{n-1}}{|K_{i,j}|} \sum_{k,l} |\Gamma_{i,j}^{k,l}| g(c_{i,j}(t_{n-1}), c_{k,l}(t_{n-1})) \quad \forall n \quad (10)$$

7 Annexes

7.1 Création du maillage

————— Creation du maillage carre en 2 dimensions

```
import math
import numpy as np
import matplotlib.pyplot as plt
```

```

x0 = 0; y0 = 0 # On definit l'origine du repere 2D (coordonnees de la
    premiere face)
# Domaine de resolution
Lx = 1 # intervalle [x0,Lx] selon x
Ly = 1 # intervalle [y0,Ly] selon y

LL = min(Lx,Ly)
mylevel = 8 # Parametre entier a modifier pour modifier le pas du
    maillage
NN = 2**mylevel

Nx = NN * int(Lx/LL) # nombre de faces selon la direction x
Ny = NN * int(Ly/LL) # nombre de faces selon la direction y

hx = Lx/Nx # pas d'espace selon x
hy = Ly/Ny # pas d'espace selon y
volume = hx*hy # volume d'une cellule

neighbours = 4 # nombre de voisins par cellule

## Initialisation
Ncell = 0 # nombre de cellules
codim0 = np.array([[0,0]]) # chaque ligne stocke les coordonnees (x,y)
    du centre de chaque cellule
Nface = 0 # nombre de frontieres
codim1 = np.array([[0,0,0,0]]) # chaque ligne stocke les coordonnees (x,
    y)
    # de deux points delimitant une face
    # (de la gauche vers la droite pour les faces horizontales
    # du bas vers le haut pour les faces verticales)

codim0to1E = [] # liste des longueurs des frontieres (correspond avec
    codim1)
codim0to1NX = [] # composantes x des normales des faces (correspond avec
    codim1)
codim0to1NY = [] # composantes y des normales des faces (correspond avec
    codim1)
Nghost = 0 # nombre de cellules fantome88 (au bord)
codim0to1A = [] # liste des numeros des cellules en amont des frontieres
    correspondant a codim1
codim0to1B = [] # liste des numeros des cellules en aval des frontieres
    correspondant a codim1

## Creation du maillage

```

```

# On parcourt le maillage ligne par ligne , de la gauche vers la droite ,
  et de bas en haut
ny = 0
while ny<=Ny:
    nx = 0
    while nx<=Nx:
        #(numcell = nx + ny*Nx # numero de la cellule (on commence la
          numerotation a 0))

        # ----- On ajoute une cellule
        -----
        xx = x0 + nx*hx # abscisse du centre de la cellule
        yy = y0 + ny*hy # ordonnee du centre de la cellule

        codim0=np.concatenate((codim0,np.array([[xx,yy]])))

        # — On cree la face verticale a gauche de la cellule —
        # Cas du bord x=0
        if nx==0:
            Nface+=1
            codim1=np.concatenate((codim1,np.array([[xx-hx/2,yy-hy/2,xx-
              hx/2,yy+hy/2]])))

            ex = codim1[Nface,2]-codim1[Nface,0] # >0 (face horizontale)
              , ou 0
            ey = codim1[Nface,3]-codim1[Nface,1] # >0 (face verticale),
              ou 0
            assert(ex>=0 and ey>=0)
            E = math.sqrt(ex*ex + ey*ey) # norme de la face
            NX = ey/E; # composante x de la normale a la face (ne pas
              confondre avec Nx!!)
            NY = ex/E; # composante y de la normale a la face (ne pas
              confondre avec Ny!!)
            codim0to1E.append(E)
            codim0to1NX.append(NX)
            codim0to1NY.append(NY)

            # On ajoute les deux cotes de la face
            codim0to1A.append(Ncell+Nx) # car il n'y a pas de cellule en
              amont
            codim0to1B.append(Ncell) # la cellule que l'on vient de
              creer est en aval
            Nghost+=1

```

```

# — On cree la face verticale a droite de la cellule —
Nface+=1
codim1=np.concatenate((codim1,np.array([[xx+hx/2,yy-hy/2,xx+hx
/2,yy+hy/2]])))

ex = codim1[Nface,2]-codim1[Nface,0] # >0 (face horizontale), or
0
ey = codim1[Nface,3]-codim1[Nface,1] # >0 (face verticale), or 0
assert(ex>=0 and ey>=0)
E = math.sqrt(ex*ex + ey*ey) # norme de la face
NX = ey/E; # composante x de la normale a la face (ne pas
confondre avec Nx!!)
NY = ex/E; # composante y de la normale a la face (ne pas
confondre avec Ny!!)
codim0to1E.append(E)
codim0to1NX.append(NX)
codim0to1NY.append(NY)

# Cas du bord x=Nx
if nx==Nx:
    codim0to1A.append(Ncell) # la cellule que l'on vient de
    creer est en amont
    codim0to1B.append(-(Nghost+1)) # car il n'y a pas de cellule
    en aval
    Nghost+=1

else:
    codim0to1A.append(Ncell) # la cellule que l'on vient de
    creer est en amont
    codim0to1B.append(Ncell+1)

# — On cree la face horizontale en-dessous de la cellule —
# Cas ou y=0
if ny==0:
    Nface+=1
    codim1=np.concatenate((codim1,np.array([[xx-hx/2,yy-hy/2,xx+
hx/2,yy-hy/2]])))

    ex = codim1[Nface,2]-codim1[Nface,0] # >0 (face horizontale)
    , ou 0
    ey = codim1[Nface,3]-codim1[Nface,1] # >0 (face verticale),
    ou 0
    assert(ex>=0 and ey>=0)
    E = math.sqrt(ex*ex + ey*ey) # norme de la face

```

```

NX = ey/E; # composante x de la normale a la face (ne pas
            confondre avec Nx!!)
NY = ex/E; # composante y de la normale a la face (ne pas
            confondre avec Ny!!)
codim0to1E.append(E)
codim0to1NX.append(NX)
codim0to1NY.append(NY)
codim0to1A.append(-(Nghost+1)) # il n'y a rien en amont
codim0to1B.append(Ncell) # la cellule que l'on vient de
            creer est en aval

Nghost+=1

# — On cree la face horizontale au-dessus de la cellule —
Nface+=1
codim1=np.concatenate((codim1,np.array([[xx-hx/2,yy+hy/2,xx+hx
            /2,yy+hy/2]])))

ex = codim1[Nface,2]-codim1[Nface,0] # >0 (face horizontale), ou
0
ey = codim1[Nface,3]-codim1[Nface,1] # >0 (face verticale), ou 0
assert(ex>=0 and ey>=0)
E = math.sqrt(ex*ex + ey*ey) # norme de la face
NX = ey/E; # composante x de la normale a la face (ne pas
            confondre avec Nx!!)
NY = ex/E; # composante y de la normale a la face (ne pas
            confondre avec Ny!!)
codim0to1E.append(E)
codim0to1NX.append(NX)
codim0to1NY.append(NY)

# Cas ou y=Ny-1
if ny==Ny:
    codim0to1A.append(Ncell) # la cellule que l'on vient de
        creer est en amont
    codim0to1B.append(Ncell-Ny*(Nx+1)) # il n'y a rien en aval

    Nghost+=1
else:
    codim0to1A.append(Ncell)
    codim0to1B.append(Ncell+Nx+1)

# On passe a la cellule suivante
Ncell+=1
nx+=1

```



```

ny+=1

codim0=codim0[1:,:]
codim1=codim1[1:,:]

'''
## Visualisation et verification du maillage:
#### En noir le maillage, en bleu le passage des face
for i in range(len(codim1[:,1])):
    a=[codim1[i,0],codim1[i,2]]
    b=[codim1[i,1],codim1[i,3]]
    plt.plot(a,b,"black")
    if codim0to1A[i]>=0 and codim0to1B[i]>=0:
        X=[codim0[codim0to1A[i],0],codim0[codim0to1B[i],0]]
        Y=[codim0[codim0to1A[i],1],codim0[codim0to1B[i],1]]
        plt.plot(X,Y,'b')
plt.axis('equal')
plt.show()
'''

## Sauvegarde du maillage
np.save('codim0', codim0)
np.save('codim1', codim1)
np.save('codim0to1A', codim0to1A)
np.save('codim0to1B', codim0to1B)
np.save('codim0to1E', codim0to1E)
np.save('codim0to1NX', codim0to1NX)
np.save('codim0to1NY', codim0to1NY)
np.save('mylevel', mylevel)

```

7.2 Approche Eulerienne

7.2.1 Calcul du flux : solveur de Lax-Friedrichs Riemann

```

def RIEMANN(lambda1,state1,lambda2,state2):
    lambda0=max(abs(lambda1),abs(lambda2)) ## maximal wave speed
    rightf=(lambda1*state1+lambda2*state2+lambda0*(state1-state2))/2.
    leftf=-rightf ## conservative, entropy flux=0
    return [leftf,rightf,lambda0] ## end of Lax-Friedrichs Riemann
    solver

```

7.2.2 Volumes finis - cas constant

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

```

```

from riemann_scal import *

## Recuperation des parametres du maillage
codim0=np.load( 'codim0.npy ' )
codim1=np.load( 'codim1.npy ' )
codim0to1A=np.load( 'codim0to1A.npy ' )
codim0to1B=np.load( 'codim0to1B.npy ' )
codim0to1E=np.load( 'codim0to1E.npy ' )
codim0to1NX=np.load( 'codim0to1NX.npy ' )
codim0to1NY=np.load( 'codim0to1NY.npy ' )
x0 = 0.0; y0 = 0.0 # origine du repere 2D
Lx = 1.0; Ly = 1.0 # domaine de resolution [x0,x0+Lx]*[y0,y0+Ly]

LL = min(Lx,Ly)
mylevel = np.load( 'mylevel.npy ' )
NN = 2**mylevel

Nx = NN * int(Lx/LL) # nombre de faces selon la direction x
Ny = NN * int(Ly/LL) # nombre de faces selon la direction y

hx = Lx/Nx # pas d'espace selon x
hy = Ly/Ny # pas d'espace selon y
volume = hx*hy # volume d'une cellule

neighbours = 4 # nombre de voisins par cellule

## Parametres

t = 0
Tfinal = 1 # Temps final d'une simulation

ns = 16 # nombre de simulations
THETA=np.linspace(0,np.pi/2,ns) # differents angles de vitesse entre 0
    et pi/2

normev = 0.5 # cas constant : norme de la vitesse

x0,y0=0.25,0.25 # position initiale de la gaussienne
sigma = 1/50 # largeur de la gaussienne

# Definition de la vitesse
def ux(theta):
    return normev*np.cos(theta)

def uy(theta):

```

```

    return norme*np.sin(theta)

## Initialisation

def initialisation(sigma,x0,y0):
    qini=np.zeros(codim0.shape[0])
    for i in range(len(qini)):
        x,y=codim0[i]
        qini[i]=np.exp(-((x-x0)**2+(y-y0)**2)/(2*sigma**2))
    return qini

qini=initialisation(sigma,x0,y0) # Condition initiale


compteur=0
qhist=[]

## Boucle sur les differents angles de la vitesse
for theta in THETA : # Cas vitesse constante, pas uniforme sur les
    angles
    print(theta)
    qhist.append(qini.copy()) # Recuperation des vecteurs solutions a
        chaque instant de la simulation

    q0 = qini.copy()

    q1 = qini.copy()

    compteur+=1
    print (compteur)
    t=0

    uxtheta = ux(theta)
    uytheta = uy(theta)

## Boucle temporelle
while t<Tfinal:
    k = 1000
    flux=np.zeros(codim0.shape[0]) # Matrice des flux
    # On parcourt toutes les faces
    for iface in range(codim1.shape[0]):
        i=codim0to1A[iface] # Cellule en amont

```

```

j=codim0to1B[iface] # Cellule en aval
if i<0 or j<0: # Conditions de bord periodiques
    continue
# Vitesse du fluide en i et j projete selon la normale a la
    face
lambdai = uxtheta*codim0to1NX[iface]+uytheta*codim0to1NY[
    iface]
lambdaj = uxtheta*codim0to1NX[iface]+uytheta*codim0to1NY[
    iface]
statei = q0[i] # Concentration en i
statej = q0[j] # Concentration en j
# Schema de Lax-Friedrich: calcul du flux en i et en j a
    travers la face
[leftf ,rightf ,Lambda] = RIEMANN(lambdai ,statei ,lambdaj ,
    statej)
# Mise a jour des flux
flux[i]+=leftf*codim0to1E[iface]
flux[j]+=rightf*codim0to1E[iface]
# Calcul du pas de temps associez
k=min(k , volume / (2*(hx+hy)*Lambda))
# Mise a jour de la concentration
q1+=flux*(k/volume)
qhist.append(q1.copy())
q0 = q1.copy()
t+=k
# Remplissage d'une matrice compose des vecteurs solutions que l'on
    va rajouter a la matrice de snapshots

SNAPSHOTMATRIX=np.zeros((codim0.shape[0] , len(qhist)))
for ligne in range(codim0.shape[0]):
    for colonne in range(len(qhist)):
        SNAPSHOTMATRIX[ligne ,colonne]=qhist[colonne][ligne]

print (SNAPSHOTMATRIX.shape[0]) # Compte le nombre de snapshots
np.save( 'SNAPSHOTMATRIX' ,SNAPSHOTMATRIX) # Sauvegarde. Attention a
    changer le nom pour differentes simulations

## Animation 3D

for q in qhist:
    plt.clf()
    fig = plt.figure(1)
    ax = fig.gca(projection='3d')

```

```

X=np.array([codim0[i,0] for i in range(codim0.shape[0])])
Y=np.array([codim0[i,1] for i in range(codim0.shape[0])])
x=X.reshape(Nx+1,Ny+1)
y=Y.reshape(Nx+1,Ny+1)
ax.plot_wireframe(x,y,q.reshape(Nx+1,Ny+1))#,False)
plt.pause(0.1)

```

7.2.3 Decomposition SVD

```

import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

## Declaration des parametres et recuperation des donnees du maillage

nbmodes = 25 #nombre de modes pour la POD

# Domaine de resolution
Lx = 1 # intervalle [x0,Lx] selon x
Ly = 1 # intervalle [y0,Ly] selon y

LL = min(Lx,Ly)
mylevel = np.load('mylevel.npy')
NN = 2**mylevel

Nx = NN * int(Lx/LL) # nombre de faces selon la direction x
Ny = NN * int(Ly/LL) # nombre de faces selon la direction y

hx = Lx/Nx # pas d'espace selon x
hy = Ly/Ny # pas d'espace selon y
volume = hx*hy # volume d'une cellule

codim0=np.load('codim0.npy')
codim1=np.load('codim1.npy')
codim0to1A=np.load('codim0to1A.npy')
codim0to1B=np.load('codim0to1B.npy')
codim0to1E=np.load('codim0to1E.npy')
codim0to1NX=np.load('codim0to1NX.npy')
codim0to1NY=np.load('codim0to1NY.npy')

## Recuperation de la matrice de snapshots

M=np.load('SNAPSHOTMATRIX.npy')

## Decomposition SVD

```

```

u,s,v=np.linalg.svd(M,False) # Thin SVD

## Trace des valeurs singulieres
plt.plot([i for i in range(nbmodes)], [np.log(s[i]/s[0]) for i in range(
    nbmodes)])
plt.show()

## Trace du premier mode propre en 3D

fig=plt.figure(1)
ax=fig.gca(projection='3d')
X=np.array([codim0[i,0] for i in range(codim0.shape[0])])
Y=np.array([codim0[i,1] for i in range(codim0.shape[0])])
x=X.reshape(Nx+1,Ny+1)
y=Y.reshape(Nx+1,Ny+1)
ax.plot_wireframe(x,y,-u[:,0].reshape(Nx+1,Ny+1))
plt.show()

## Sauvegarde de la decomposition

np.save('u_eulerien_constant',u)
np.save('s_eulerien_constant',s)

## Recuperation et sauvegarde de la base de vecteurs propres

ur=u[:, [i for i in range(13)]]

np.save('ur_eulerien_constant',ur)

```

8 Remerciements

9 Exemples de figures, tableau, équations...

Ce court texte illustre la façon de citer les références [1, 2, ?, ?, ?, ?], de faire une note de bas de page¹, de faire référence à une partie du document (Sec. 4), à une figure (Figure ??), à un tableau (Tableau ??) ou à une équation (Equation 11).

$$H\psi = i\hbar \frac{\partial \psi}{\partial t}. \quad (11)$$

1. Exemple de note de bas de page.

Références

- [1] V. Ehrlacher, S.Boyaval *Méthodes numériques pour les problèmes en grande dimension*. Cours de l'École nationale des ponts et chaussées, 2018.
- [2] S.Boyaval *Numerical Hydrodynamics for the environment Illustrated by river flows* Cours de l'École nationale des ponts et chaussées, 2018.