Lattice Boltzmann, calcul d'erreur

Nguyen Phuc-Thien Thomas

30 avril 2013

Résumé

Ce document a été rédigé dans le cadre de mon stage à l'Université de Genève. Plus précisément au centre universitaire d'informatique, où on travaille notamment sur la simulation numérique de divers phénomènes physiques ou biologiques. C'est la suite du chapitre sur les automates cellulaires.

Ici sera présenté la méthode de Boltzmann sur réseau, ou lattice Boltzmann, qui n'est qu'autre qu'un automate cellulaire, avec simplement des règles plus élaborées qu'un simple jeu de la vie. C'est un automate qui permet de simuler de manière tout à fait honorable l'écoulement de fluides dans diverses situations. Une implémentation a été faite durant ce stage. Elle n'est pas parfaite, ce qui est constaté et expliqué dans le calcul d'erreurs.

1 Introduction

La méthode de *lattice Boltzmann* [2], francisable en *Boltzmann sur réseau*[1], est un modèle permettant de simuler l'écoulement de fluides. Il peut être implémenté de façon similaire à un automate cellulaire[2] sur ordinateur, et utilise un algorithme de collision nommé *BGK*[3].

Cette méthode considère non pas chaque particule individuellement (présence ou absence d'une particule à chaque point un temps donné), mais comme des populations de particules (densités et vitesses à chaque point du domaine un temps donné), contrairement à HPP ou FHP[4] qui considèrent chaque particule individuellement. Ces deux derniers simulent donc un fluide au niveau microscopique, tandis que lattice Boltzmann permet une simulation au niveau mésoscopique, c'est-à-dire à un niveau intermédiaire entre microscopique et macroscopique (de l'ordre du micromètre selon Wikipédia). Ceci permet d'avoir une bonne précision par rapport à une échelle macroscopique, tout en supprimant les défauts inhérents à une simulation à l'échelle microscopique (défauts qui ne seront pas traités dans ce rapport). Elle a aussi l'avantage d'être parallélisable.

Nous allons survoler cette méthode, l'implémenter, avant de faire des calculs d'erreurs.

1. Toutefois, la francisation n'est à peu près jamais utilisée. On dit également souvent *lattice Boltzmann* tout court.
2. En soi, la méthode de lattice Boltzmann en est un.
3. Initiales du nom des auteurs : Bhatnagar, Gross et Krook
4. Respectivement Hardy, Pommeau, Pazzis et Frisch, Hasslacher, Pommeau.

2 Algorithme

Il existe plusieurs possibilités pour cette méthode : utiliser un réseau hexagonal ou carré, en 1, 2 ou 3 dimensions, etc. Nous allons utiliser les caractéristiques suivantes : réseau carré, de vecteurs D2Q9, en deux dimensions. D2Q9 signifie que c'est en 2D, et qu'il y a neuf vecteurs différents. Ces vecteurs (2) représentent les neuf directions possibles que peuvent emprunter les particules du plan : une population de particules à un point donné pourra se déplacer à droite, en haut, en bas à gauche, etc, ou bien rester sur place. Ces vecteurs sont le plus souvent notés $\vec{c_i}$, $\vec{c}$ contenant toutes les neuf combinaisons, et i étant l'indice [5] (par exemple $i = 0$, $\vec{c_i} = \vec{c_0} =$ arbirairement un vecteur nul $(0, 0)$; $i = 1$, $\vec{c_i} = \vec{c_1} =$ arbirairement $(1, 0)$, etc.). Arbitrairement, car on peut choisir comme on le souhaite la correspondance entre un vecteur et son indice.

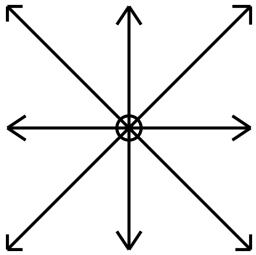

FIGURE 1 – Les neuf vecteurs vitesse possibles dans un réseau D2Q9.

Par conséquent, chaque point d'un domaine considéré possède neuf populations f_i [6] qui se déplaceront au temps $t + 1$ selon l'un des neuf vecteurs respectif. La somme de ces populations donne la densité ρ en ce point [7] :

$$\rho = \sum_i f_i. \tag{1}$$

Il y a une autre formule, qui permet de calculer la vitesse moyenne $\vec{u}$ de chaque point du plan. C'est la somme des produits scalaires entre les populations f_i et les vecteurs $\vec{c_i}$, divisé par la densité :

$$\vec{u} = \frac{1}{\rho} \sum_i \vec{c_i} f_i \tag{2}$$

Passons maintenant aux règles de collision. Bien qu'étant un automate cellulaire plus ou moins comme les autres, les règles de collision sont définies par des formules plus compliquées, dont leur démonstrations [2] sont assez longues et ne seront pas exposées ici. Une première formule donne ce qui est appelé la *distribution d'équilibre* $f_i^{(0)}$ pour chaque population :

$$f_i^{(0)} = \rho t_i (1 + \frac{\vec{u} \cdot \vec{c_i}}{c_s^2} + \frac{(\vec{u} \cdot \vec{c_i})^2}{2c_s^4} - \frac{\vec{u}^2}{2c_s^2}) \tag{3}$$

5. Ceci ne s'applique pas seulement ici, cette notion étant très utilisée en sciences dures. Notez que $i = \{0; 1; 2; ...; 8\}$, ou $\{1; 2; 3; ...; 9\}$ ici.

6. i représente toujours un indice. Les populations f_i sont respectives à leurs vecteurs $\vec{c_i}$, par exemple, si $\vec{c_2} = (0, -1)$, les populations de f_2 se déplaceront selon un vecteur $(0, -1)$, c'est-à-dire vers le haut.

7. Pour être plus rigoureux, il faudrait écrire $\rho(\vec{x}, t) = \sum_i f_i(\vec{x}, t)$ afin de montrer que ces grandeurs sont fonctions du temps et de la position $\vec{x} = (x, y)$. Cependant, pour éviter d'alourdir l'écriture, le $(\vec{x}, t)$ sera omis par la suite.

c_s réprésente la *vitesse du son* de l'automate, et vaut dans notre cas $\frac{1}{\sqrt{3}}$. t_i est une constante qui a trois valeurs possibles en fonction de i : $\frac{4}{9}$ si $\vec{c_i}$ est le vecteur nul, $\frac{1}{9}$ si c'est un vecteur cardinal, et $\frac{1}{36}$ si c'est un vecteur diagonal. La règle de collision de la méthode de lattice Boltzmann est ensuite donnée par :

$$f_i(\vec{x} + \vec{c_i}, t+1) = (1 - \frac{1}{\tau}) f_i(\vec{x}, t) + \frac{1}{\tau} f_i^{(0)}(\vec{x}, t) \tag{4}$$

Cela nous permet donc, comme les fonctions de f_i l'indiquent, de trouver l'état au temps $t+1$ de toutes les populations de tous les points du fluide, tout en prenant en compte leur déplacement selon leur vecteur $\vec{c}$. τ est le temps de relaxation, une valeur que vous pouvez choisir arbitrairement, et qui a un lien avec la viscosité ν du liquide [2] :

$$\nu = c_s^2(\tau - \frac{1}{2}) \tag{5}$$

En revanche, comme on peut le constater dans cette relation, les formules ne sont plus applicables si sa valeur est en-dessous de 0.5, car cela nous donnerait une viscosité négative [8].

3 Implémentation

3.1 Procédure

Maintenant, nous avons en main les informations nécessaires à la compréhension de la méthode de lattice Boltzmann. Nous pouvons alors en faire une implémentation, qui n'est pas très difficile à programmer si on a bien compris.

Pour commencer, il faut créer une matrice arbitraire de taille $9 \times m \times n$ qui contiendra toutes les populations f_i au temps initial [9]. Ensuite, il faudra calculer les vitesses et les densités en tout points (ce qui nécessitera deux matrices supplémentaires de dimensions $m \times n$), avant de pouvoir utiliser la formule permettant de trouver les distributions d'équilibre $f_i^{(0)}$ (dans encore une autre matrice de taille $9 \times m \times n$). Après cela, il s'agira de calculer la matrice finale, de taille égale à la matrice initiale, qui contiendra les états au temps $t+1$. Cela se fera en appliquant la formule 4. Bien entendu, il faudra veiller à gérer toutes les *propagations*, c'est-à-dire les déplacements de populations selon l'un des vecteurs $\vec{c}$. Ceci se fait par exemple à l'aide de la fonction *circshift* en langage MatLab.

3.2 Aperçu de cette implémentation

Voici quelques images de ce que fait une telle application, écrite en MatLab. Il s'agit d'un fluide, dont les vitesses en tout point du domaine sont nulles, et les densités égales à 1, sauf le point central qui a un $\rho = 1.1$ et qui fait office de perturbation dans ce fluide. Le domaine est une matrice de 101 par 101 cases [10].

8. La viscosité mesurant une perte d'énergie dans un fluide lors de son déplacement, cela signifirait que ce fluide fabriquerait de l'énergie en se déplaçant, ce qui est hors de question d'envisager en sciences...

9. Elle peut aussi être créée à partir de deux matrices contenant les densités et les vitesses, en appliquant la formule pour calculer les $f_i^{(0)}$.

10. Afin que le point perturbateur soit bien au centre ; il y aura bien 50 cases à ses quatre points cardinaux. Si un côté mesurait 100 cases, le point le plus centré aurait 49 cases d'un côté et 50 de l'autre. Souci de perfectionnisme...

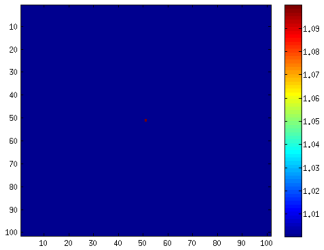

FIGURE 2 – Situation initiale (les couleurs représentent la densité)

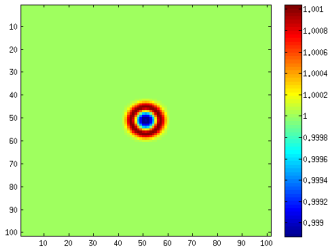

FIGURE 3 – Après 10 itérations

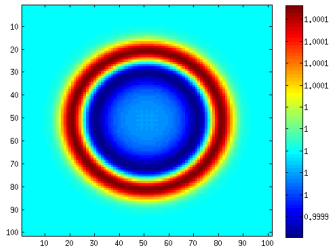

FIGURE 4 – Après 50 itérations

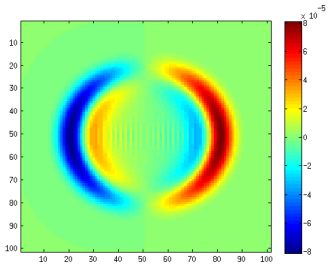

FIGURE 5 – Idem, mais en représentant les vitesses horizontales u_x

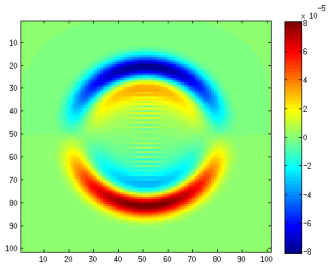

FIGURE 6 – Idem, mais en représentant les vitesses verticales u_y

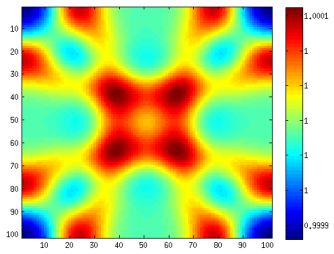

FIGURE 7 – Après 150 itérations, densité

4 Ajout de conditions aux bords

Jusque là, nous avons créé un programme qui fonctionne, mais qui possède un domaine périodique. Ce qui n'est pas très juste physiquement, à moins qu'un jour, une théorie démontre que l'Univers ne soit ni fini, ni infini, mais périodique... Par conséquent, nous allons mettre des *murs* autour de notre fluide, afin de simuler l'écoulement dans une cavité. En plus de cela, nous allons faire en sorte que le bord du haut ait une vitesse horizontale u_x de 0.1 ($\vec{u_{haut}} = (0.1, 0)$), donc allant de gauche à droite. On dit alors qu'on a imposé des *conditions aux bords*.

Il existe plusieurs manières de mettre en place cela. Nous allons mettre en place un système pas très compliqué : il suffira de faire en sorte qu'à chaque itération, les distributions d'équilibre $f_i^{(0)}$ aux bords soient définies avec $\rho = 1$, u_{haut} au bord du haut, et $\vec{u} = (0, 0)$ au trois autres bords. Elles doivent alors remplacer à chaque fois les valeurs respectives de la matrice initiale de l'itération.

Résultats en images

Matrice de 100 par 100.

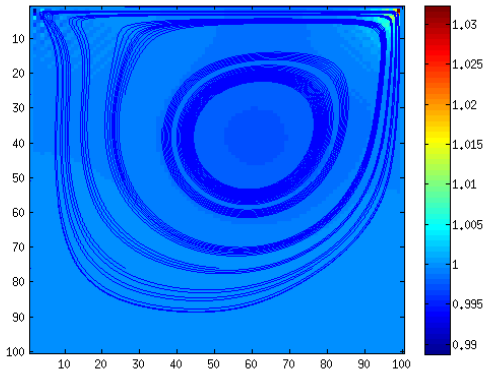

FIGURE 8 – Après 10 000 itérations (densité et lignes de courant)

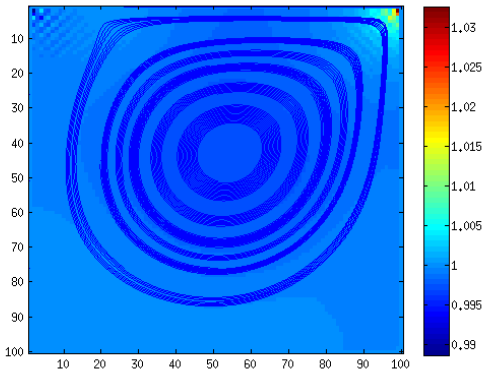

FIGURE 9 – Après 100 000 itérations

5 Calcul d'erreur

Bien que les lignes de courant semblent correspondre à une certaine réalité, le résultat est loin d'être parfait. Nous pouvons nous en convaincre en calculant les erreurs.

5.1 Protocole

Nous allons pousser un peu plus loin cette étude en calculant l'erreur dans plusieurs cas de figure. Pour commencer, des valeurs de référence seront utilisées à titre de comparaison. Ensuite, l'erreur sera calculée avec des matrices carrées de tailles diverses (25, 50, 100, 125, 150, 175 et 200). Elle sera également déterminée en fonction du nombre d'itérations, entre 10 000 et 100 000. Puis nous en ferons une analyse.

5.2 Valeurs de référence

Les valeurs obtenues avec cette implémentation seront comparées avec celles de Ghia [1], pour un nombre de Reynolds égal à 1000 [11]. On comparera les vitesses horizontales $u_x(pos)$ sur une ligne coupant verticalement et au centre le plan.

Liste des valeurs de référence :

pos_y	-1	-0.8906	-0.875	-0.8594	-0.7968	-0.6562	-0.4374	-0.0938	-
u_x	0	-0.18109	-0.20196	-0.2222	-0.2973	-0.38289	-0.27805	-0.10648	-
pos_y	0	0.2344	0.4688	0.7032	0.9062	0.9218	0.9376	0.9532	1
u_x	-0.0608	0.05702	0.18719	0.33304	0.46604	0.51117	0.57492	0.65928	1

Le nombre de Reynolds Re dépendant de la vitesse u_o qu'on a choisie pour la condition au bord en haut (pour rappel $u_x = 0.1$), de la longueur L_0 de la matrice, et de la viscosité ν, ce dernier dépendant du temps de relaxation τ, nous pouvons déterminer cette dernière

11. Les grandeurs et valeurs traitées sont caractéristiques de la situation, et n'ont donc pas d'unités.

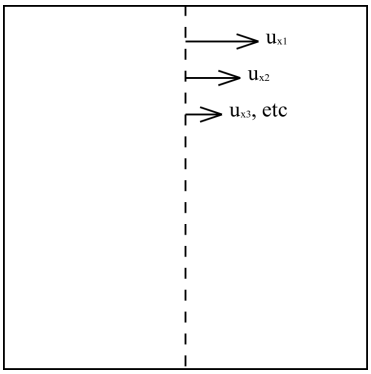

FIGURE 10 – Illustration

en fonction de la taille de la matrice, puisqu'on connaît la vitesse. En effet, on a :

$$Re = \frac{L_0 u_0}{\nu} \tag{6}$$

et comme vu précédemment :

$$\nu = c_s^2(\tau - \frac{1}{2})$$

d'où :

$$\tau = \frac{u_0 L_0}{Re c_s^2} + \frac{1}{2} \tag{7}$$

Par exemple, pour une matrice de 200 par 200 :

$$\tau = \frac{200 \cdot 0.1}{1000(\frac{1}{\sqrt{3}})^2} + \frac{1}{2} = 0.56$$

5.3 Valeurs obtenues

À titre d'exemple, voici les valeurs des u_x obtenues pour une matrice de 100 par 100, après les 100 000 itérations, avec les valeurs de références pour comparer :

pos_y	-1	-0.8906	-0.875	-0.8594	-0.7968	-0.6562	-0.4374	-0.0938	-
u_{xObt}	0	-0.0854	-0.1045	-0.1241	-0.1866	-0.3187	-0.2879	-0.1139	-
u_{xRef}	0	-0.18109	-0.20196	-0.2222	-0.2973	-0.38289	-0.27805	-0.10648	-
pos_y	0	0.2344	0.4688	0.7032	0.9062	0.9218	0.9376	0.9532	1
u_{xObt}	-0.0759	0.0423	0.1738	0.3033	0.4482	0.5209	0.6265	0.7628	1
u_{xRef}	-0.0608	0.05702	0.18719	0.33304	0.46604	0.51117	0.57492	0.65928	1

5.3.1 Formule pour calculer l'erreur

Cette formule est utilisée pour calculer l'erreur de ce que nous avons calculé par rapport aux valeurs de référence :

$$E = \sqrt{\frac{1}{N}\sum_i (x_i - y_i)^2} \tag{8}$$

Où N est le nombre de valeurs relevées (ici 17), x_i est la valeur obtenue et y_i la valeur de référence.

Le programme a été écrit de façon à montrer l'erreur toutes les 10000 itérations. Toujours en prenant ce même exemple, il a été calculé que $E = 0.0595$ après 100000 itérations. Ceci ce confirme par un calcul manuel où la valeur 0.05946 a été trouvée sur une calculatrice de poche, le calcul d'erreur a donc été correctement implémenté.

5.3.2 Tableau des erreurs

Voici maintenant les erreurs en fonction du nombre d'itérations et de la taille de la matrice (toujours carrée pour rappel), données en pourcentages. Une case vide indique que la dernière valeur a convergé, et donc que l'erreur ne varie plus. Vous pouvez voir graphiquement un exemple d'erreur en fonction du temps, et mieux mettre en évidence la convergence (page suivante).

Nb.d'it.	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
25	13.38	13.36								
50	10.40	8.37	8.30							
75	11.83	7.12	6.42	6.36	6.35					
100	14.13	8.65	6.45	6.03	5.96	5.95				
125	15.54	10.10	6.49	4.90	4.83	4.81				
150	16.59	11.92	8.23	6.30	5.61	5.39	5.33	5.31	5.31	5.30
175	16.89	12.97	9.35	6.80	5.58	5.10	4.93	4.87	4.85	4.84
200	17.60	14.10	10.77	7.97	6.24	5.40	5.03	4.88	4.82	4.80

5.4 Analyse

Nous pouvons faire un certain nombre de remarques à propos des valeurs de ce tableau. Pour commencer, nous constatons que plus la matrice est grande, plus de temps l'erreur met à converger (et plus grande elle est au début). Ceci est assez intuitif et évident, une matrice qui a moins de points, mais les même vitesses qu'une plus grande formera plus rapidement une situation stable, où les lignes de courant ne changent plus. Il en va donc de même pour l'erreur.

Naturellement, plus il y a d'itérations, plus la situation se rapproche de la stabilité, donc plus l'erreur diminue (jusqu'à convergence). Cette figure montre l'évolution des $u_x(pos)$ pour une matrice de 100 cases de côté (rouge après 10 000 itérations, verte après 100 000, bleues pour les intermédiaires), et les valeurs de références en noir, et on voit que les valeurs mesurées se rapprochent de plus en plus des références jusqu'à un moment où elles se mettent à se chevaucher :

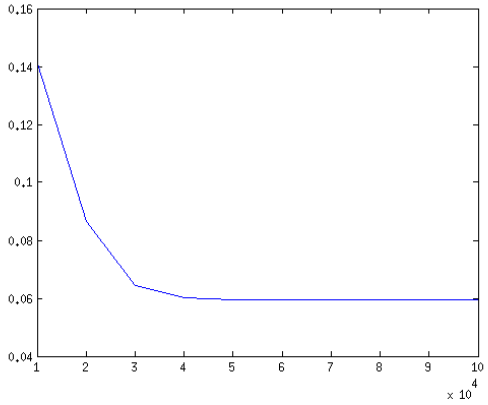

FIGURE 11 – Évolution des erreurs en fonction du temps pour cette matrice

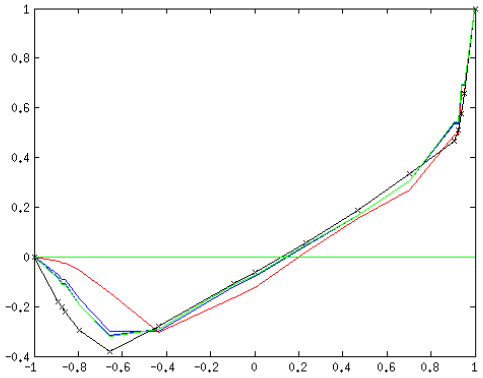

FIGURE 12 – On voit l'évolution des $u_x(pos)$ et des erreurs au cours du temps

Ensuite, si nous prenons les valeurs convergées, et que nous dessinons un graphique de ces dernières en fonction de la taille de la matrice :

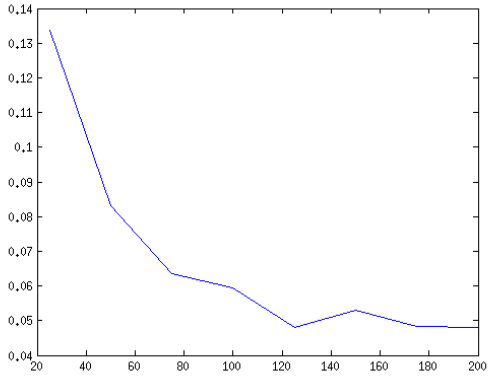

FIGURE 13 – Évolution des erreurs en fonction de la taille de la matrice

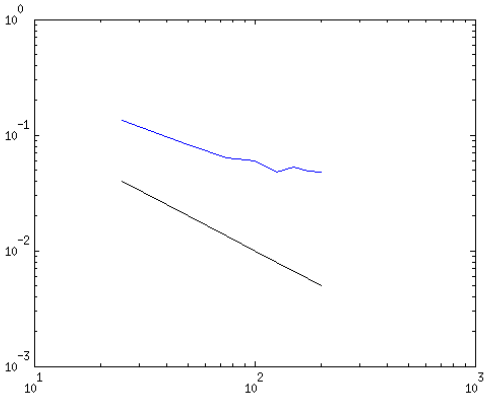

FIGURE 14 – Idem avec une échelle logarithmique (en bleu). Le graphe noir est une pente de -1, celle qu'aurait en théorie la bleue si l'erreur est divisée par deux lorsque la taille de la matrice double.

Nous remarquons que jusqu'à $L_0 = 75$, le graphique avec l'échelle logarithmique fait quasiment une droite. C'est ce qui est attendu : si on double la longueur de la matrice, on double la précision [12]. En revanche, à partir de $L_0 = 100$, la fonction commence à se comporter de manière curieuse. En y regardant de plus près, il semblerait même que l'erreur diminue de moins en moins vite en général.

Cela s'explique par le fait que notre modèle de conditions aux bords n'est pas parfait. En effet, nous avons utilisé un modèle très simple, qui est loin d'être le meilleur.

12. Ce n'est pas tout à fait le cas de cette droite, comme on le voit à la figure 14.

6 Conclusion

Dans ce rapport, nous avons pu introduire lattice Boltzmann, puis survoler une version de son algorithme. Nous l'avons ensuite implémenté. Puis nous avons ajouté des conditions aux bords, ce qui nous a permis de simuler un écoulement dans une cavité. Enfin, nous nous sommes rendus compte que notre modèle n'est pas parfait, et qu'il est possible de faire mieux. Ce document nous a surtout apporté une bonne mise en pratique des connaissances sur les automates cellulaires et du calcul d'erreurs, une bonne introduction à la méthode de lattice Boltzmann, et un petit premier pas dans le monde de la mécanique des fluides.

Références

[1] Cours contenant les solutions de Ghia. http://158.110.32.35/download/CERN07/exe1.pdf/, consulté le 30 avril 2013.

[2] Erlend Magnus Viggen. The Lattice Boltzmann Method with Applicaions in Acoustics, 2009.