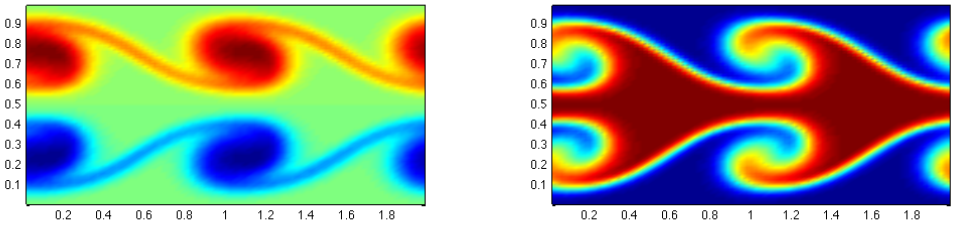
Mini projet de programmation

(Matmeca 2ème année, 2014-2015)

Luc Mieussens

Équipe Calcul Scientifique et Modélisation
Institut de Mathématiques de Bordeaux,
Université de Bordeaux,
351, cours de la Libération, 33405 Talence cedex,
France
(Luc.Mieussens@math.u-bordeaux1.fr)



Le but de ce mini projet est de programmer une méthode simple de résolution des équations de Navier-Stokes incompressibles pour simuler le phénomène des instabilités de Kelvin-Helmoltz. Celles-ci se produisent par exemple lorsque dans un fluide au repos, une partie de fluide est animée d'une vitesse non nulle (comme avec un jet) : en raison des contraintes de cisaillement, l'interface entre le fluide au repos et le jet est instable : elle se déforme, et des tourbillons se développent. La méthode numérique utilisée est la méthode des différences finies, basée sur une formulation des équation de Navier-Stokes proposée récemment par Johnson et Liu (Journal of Computational Physics 199 (2004) 221–259) qui élimine astucieusement la contrainte de divergence nulle.

1 Formulation PPE des équations de Navier-Stokes

D'après l'article mentionné plus haut, les équations de Navier-Stokes

$$\begin{aligned}
&\partial_t u + (u\cdot\nabla)u + \nabla p = \nu\Delta u,\\
&\nabla\cdot u = 0,\\
&u|_{\partial\Omega} = 0,
\end{aligned}$$

sont équivalentes à la la formulation suivante, dans laquelle la contrainte d'incompressibilité est éliminée et remplacée par une équation de Poisson pour la pression avec conditions aux limites

de Neumann :

$$\begin{aligned}
&\partial_t u + (u \cdot \nabla)u + \nabla p = \nu \Delta u, \\
&u|_{\partial\Omega} = 0, \\
&\Delta p = -\nabla \cdot ((u \cdot \nabla)u), \\
&\partial_n p|_{\partial\Omega} = -\nu n \cdot (\nabla \wedge \nabla \wedge u)|_{\partial\Omega},
\end{aligned}$$

où n est la normale extérieur à Ω. Cette équivalence se montre sans difficulté en utilisant l'identité $\Delta u = -\nabla \wedge \nabla \wedge u + \nabla(\nabla \cdot u)$.

Dans ce mini-projet, il n'y a pas de bord solide dans le problème considéré, et nous imposerons des CL périodiques pour l'équation sur u et pour l'équation de Poisson :

$$\begin{aligned}
&\partial_t u + (u \cdot \nabla)u + \nabla p = \nu \Delta u, \\
&\Delta p = -\nabla \cdot ((u \cdot \nabla)u), \\
&u \text{ et } p \text{ périodiques sur } \Omega
\end{aligned}$$

L'équivalence entre les deux formulations est encore plus simple à montrer.

L'avantage de cette formulation est la simplicité de la méthode numérique à utiliser : schéma collocalisé, différences finies centrées en espace, schéma RK3 en temps, résolution de l'équation de Poisson par transformée de Fourier discrète. Le but de ce mini-projet est de programmer cette méthode numérique en Matlab, et de la tester sur le problème des instabilités de Kelvin-Helmoltz.

2 Prise en main de matlab

2.1 Préliminaires

Pour commencer, nous allons résoudre une simple équation de convection-diffusion scalaire, linéaire, en une dimension d'espace :

$$\partial_t u + a\partial_x u = \nu \partial_{xx} u,$$

dans le domaine $[0, 1]$ avec la donnée initiale $u(0, x) = 1/\sqrt{0.4\pi}\exp(-(x - 0.5)^2/0.1)$ et des conditions aux limites périodiques, la vitesse de convection $a = 1$, et le coefficient de diffusion $\nu = 0.02$.

Le maillage a un pas $\Delta x = c\frac{2\nu}{a}$, avec c une constante fixée pour le moment à $c = 0.9$. Les noeuds du maillage sont les centres des mailles, définis et stockés dans le tableau `x` avec l'instruction

```
x=0+dx/2:dx:L-dx/2;
```

où `dx` est la variable stockant Δx. La taille de ce tableau est

```
imax=length(x);
```

La solution $u(t, x)$ est stockée dans un tableau de même taille que `x`, avec pour valeur initiale `u= 1/sqrt(2*pi*0.05)*exp(-(x-0.5).^2/(2*0.05));` En matlab, toutes les fonctions mathématiques s'appliquent aux tableaux composantes par composantes, mais pour les opérations produit, division, et exposant, il faut faire précéder l'opérateur d'un point (voir l'instruction ci-dessus).

Pour visualiser cette donnée initiale, on peut utiliser l'instruction suivante

```
plot(x,u,'-')
```

pour une courbe en trait plein, ou

```
plot(x,u,'o')
```

pour ne tracer que les points (x_i, u_i) avec le symbole o.

À présent, ouvrir un fichier vierge avec l'éditeur de matlab, et copier dedans toutes les instructions précédentes, puis sauver ce fichier sous le nom `cdiff.m`. Ce fichier est un programme appelé script, que l'on peut exécuter dans la fenêtre de commande en tapant simplement le nom du fichier, sans l'extension, soit

```
cdiff
```

2.2 Boucle en temps

Dans un premier temps, nous utilisons un schéma d'euler explicite en temps, et centré en espace :

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + a\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = \nu\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2},$$

avec $i = 1$ à i_{max}, et les conditions aux limites $u_0^n = u_{i_{max}}^n$ et $u_{i_{max+1}}^n = u_1^n$. Le pas de temps est donné par la condition CFL usuelle $\Delta t = cfl\frac{\Delta x^2}{2\nu}$.

Pour que les calculs effectués par matlab soient rapides, il vaut mieux effectuer les calculs sur un vecteur plutôt que faire une boucle sur les composantes de ce vecteur. Pour le schéma étudié ici, cela signifie qu'il faut appliquer le schéma sur le vecteur u tout entier au lieu de faire une boucle en i. Pour cela, on va utiliser les tableaux d'entiers suivants, de même taille que `u` :

```
i=[1:imax]; ip=[2:imax,1]; im=[imax,1:imax-1];
```

de sorte que le schéma précédent et les conditions aux limites s'écrivent en une seule instruction

```
u = u - dt*a/(2*dx)*(u(ip) - u(im)) + dt*mu/dx^2*(u(ip) - 2*u + u(im));
```

Vérifier sur le papier que cette instruction correspond bien au schéma.

Enfin, il n'y a plus qu'à rajouter la boucle en temps

```
for n=1:1000
  % mettre ici l'instruction codant le schema

  % affichage
  plot(x,u,'-')
  axis([0 1 -0.2 3])
  drawnow
end
```

où les instructions d'affichage produisent une courbe dont l'axe des abscisses est restreint au segment $[0, 1]$ et celui des ordonnées au segment $[-0.2, 1]$. L'instruction finale force l'affichage au lieu que celui-ci ne se fasse quà la fin de la boucle.

Rajoutez toutes ces instructions dans le script, puis testez-le.

Il est possible de déterminer le comportement en temps grand de la solution exacte, et de comparer avec ce que donne le schéma.

2.3 Un autre schéma

Relancez le script avec $c = 2$ au lieu de 0.9. Vous devez constater que la solution numérique oscille et que les oscillations s'amplifient rapidement (enlever la restriction des axes imposée

par `axis` pour voir ces oscillations). En effet, on peut facilement prouver que le schéma ne préserve pas la positivité si $c > 1$ (voir les méthodes vues en ANPI1) : attention, cela n'est pas une condition CFL sur la pas de temps, mais une contrainte sur le maillage en espace qui est nécessaire pour assurer la positivité. Ces oscillations peuvent être atténuées en diminuant largement le pas de temps, par exemple avec $cfl = 0.2$, car le schéma est alors L^2 stable. En fait, une analyse soigneuse donne la stabilité quelle que soit c si $\Delta t \leq \min(\Delta x^2/(2\nu), 2\nu/a^2)$.

En quelques mots, on peut interprêter cette contrainte avec la méthode des lignes. On discrétise l'équation de convection diffusion en espace seulement, avec le schéma centré. On obtient un système différentiel dont les valeur propres ont une partie réelle négative, due à la dérivée seconde, et une partie imaginaire pure due à la convection. Le schéma précédent est une approximation par Euler explicite du système différentiel. On sait qu'Euler explicite est instable, sans condition, pour un système ayant des valeurs propres imaginaires pures. Ici, comme le coefficient de diffusion est petit, les valeurs propres sont très proches de l'axe imaginaire, et le pas de temps doit être très petit pour stabiliser le schéma (d'où la condition $\Delta t \leq 2\nu/a^2$ obtenue plus haut). Cette contrainte est donc particulièrement restrictive quand ν est petit. On pourra vérifier ces assertions en modifiant le calcul de Δt dans le script.

Pour éliminer cette contrainte, on peut utiliser un schéma qui reste stable pour un système différentiel avec valeurs propres imaginaires pures : il suffit de trouver un schéma dont le domaine de stabilité comprend une partie de l'axe imaginaire. C'est le cas des schémas d'ordre 3, par exemple RK3, défini par le tableau

0			
1/2	1/2		
1	-1	2	
	1/6	2/3	1/6

Une analyse de stabilité montre que ce schéma est stable à la limite $\nu = 0$ pour $\Delta t \leq \sqrt{3}\Delta x/a$. On peut donc se donner comme pas de temps $\Delta t \leq \min(\Delta x^2/(2\nu), \sqrt{3}\Delta x/a)$, et l'on n'a plus de contrainte quand ν est trop petit.

Pour utiliser ce schéma dans le script, il convient de copier le calcul du second membre du schéma d'Euler explicite dans un autre script. Ce fichier va s'appeler `SM.m` et contient une fonction définie comme suit :

```
function du=SM(u,dx,i,ip,im,mu,a)
  du = - a/(2*dx)*(u(ip) - u(im)) + mu/dx^2*(u(ip) - 2*u + u(im));
```

Avec cette fonction, une itération en temps du schéma d'Euler explicite s'écrit maintenant dans le script principal :

```
du = SM(u,dx,i,ip,im,mu,a) ;
u=u+dt*du ;
```

Pour utiliser RK3, il suffit de combiner trois appels à `SM` pour calculer ensuite u

```
du1=SM(u,dx,i,ip,im,mu,a);
du2=SM(u+dt/2*du1,dx,i,ip,im,mu,a);
du3=SM(u-dt*du1+2*dt*du2,dx,i,ip,im,mu,a);

u=u+dt/6*(du1+4*du2+du3);
```

Vous pouvez à present constater que ce schéma permet de prendre des pas de temps bien plus grands que le schéma d'Euler explicite.

3 Résolution de l'équation de Poisson en 2D

Avant de passer à Navier-Stokes, il est nécessaire de savoir résoudre une équation de Poisson, afin de pouvoir calculer la pression. Cette équation est posée dans le rectangle $[0, L_x] \times [O, L_y]$ et s'écrit

$$\Delta p = f,$$

où $\Delta = \partial_{xx} + \partial_{yy}$ et f est un second membre. Les conditions aux limites sont périodiques. Si f est connue analytiquement, on peut résoudre cette EDP de façon exacte par séries de Fourier. Cependant, dans la suite, f sera issue d'une discrétisation de $-\nabla \cdot ((u \cdot \nabla)u)$ et ne sera donc connue qu'aux points (x_i, y_j) : l'outil qui remplace les séries de Fourier pour une tableau de points (ici $f = (f_{i,j})$) s'appelle la transformée de Fourier discrète (ou TFD) :

$$\hat{f}_{k,l} = \mathcal{F}(f)_{k,l} = \sum_{k=1,l=1}^{i_{max},j_{max}} f_{i,j} \exp\Big(-\mathbf{i}\frac{2\pi}{i_{max}}(k-1)(i-1)\Big) \exp\Big(-\mathbf{i}\frac{2\pi}{j_{max}}(l-1)(j-1)\Big),$$

qui s'inverse par la formule :

$$f_{i,j} = \mathcal{F}^{-1}(\hat{f})_{i,j} = \frac{1}{i_{max}j_{max}} \sum_{i=1,j=1}^{i_{max},j_{max}} \hat{f}_{k,l} \exp\Big(\mathbf{i}\frac{2\pi}{i_{max}}(k-1)(i-1)\Big) \exp\Big(\mathbf{i}\frac{2\pi}{j_{max}}(l-1)(j-1)\Big),$$

où l'on note $\mathbf{i}$ la racine de -1. Il est facile de voir que cet opérateur diagonalise l'équation de Poisson discrétisée par différence finies centrées (de même que la transformée de Fourier continue diagonalise l'équation de Poisson continue). Autrement dit, si $p = (p_{i,j})$ est solution de

$$\frac{1}{\Delta x^2}(p_{i+1,j} - 2p_{i,j} + p_{i-1,j}) + \frac{1}{\Delta y^2}(p_{i,j+1} - 2p_{i,j} + p_{i,j-1}) = f_{i,j},$$

avec les conditions périodiques $p_{0,j} = p_{imax,j}$, $p_{imax+1,j} = p_{1,j}$, $p_{i,0} = p_{i,jmax}$, $p_{i,jmax+1} = p_{i,1}$, alors $\hat{p} = \mathcal{F}(p)$ vérifie

$$\hat{p}_{k,l} = \frac{1}{c_{x,k} + c_{y,l}} \hat{f}_{k,l},$$

pour $k = 1$ à i_{max} et $l = 1$ à j_{max}, où

$$c_{x,k} = \frac{2}{\Delta x^2}\left(\cos(\frac{2\pi}{i_{max}}(k-1)) - 1\right) \text{ et } c_{y,l} = \frac{2}{\Delta y^2}\left(\cos(\frac{2\pi}{j_{max}}(l-1)) - 1\right).$$

Attention, cette formule n'a pas de sens pour $(k, l) = (1, 1)$, car $c_{x,1} = c_{y,l} = 0$: cela traduit le fait que l'équation de Poisson considérée n'a pas de solution unique (toute constante est solution), ce qui est cohérent avec le fait que dans les équations de Navier-Stokes, la pression n'est définie qu'à une constante près. On définit donc arbitrairement la valeur $\hat{p}_{1,1} = 0$ (ce qui revient à imposer une valeur moyenne nulle de la pression). Il ne reste plus qu'à effectuer la transformation inverse pour obtenir p.

Avec matlab, tout ceci va se faire très simplement en utilisant la fonction `fft2` qui calcule la transformée de Fourier discrète de tout tableau à deux dimensions. L'algorithme utilisé est le célèbre FFT (Fast Fourier transform) ou transformée de Fourier rapide, dont le coût est proportionnel au nombre d'éléments dans le tableau.

Vous devez donc à présent écrire un script pour coder la résolution expliquée ci-dessus, et la tester sur une solution exacte, par exemple avec $p(x, y) = \sin(ax)\cos(by)$ qui est solution de l'équation de Poisson avec $f(x, y) = -(a^2 + b^2)\sin(ax)\cos(by)$. Voici le script en question :

```
%-- parametres
Lx=1; Ly=1;
imax=80; jmax=80;
dx=Lx/(imax); dy=Ly/(jmax);
x=0+dx/2:dx:Lx-dx/2; y=0+dy/2:dy:Ly-dy/2;

%-- indices
i=1:imax;
j=1:jmax;

%-- 2nd membre
a=2*pi/Lx; b=2*pi/Ly;
f=-(a^2+b^2)*sin(a*x)'*cos(b*y);

%--- solution exacte
pex=sin(a*x)'*cos(b*y);

%--- sol. Fourier
p=poisson(f,i,j,dx,dy,imax,jmax);

%-- calcul de l'erreur
norm(p-pex)/norm(pex)
```

Il vous reste à écrire la fonction `poisson` qui résoud l'équation de Poisson. Pour cela, il faut là encore ne travailler qu'avec des tableaux (2D ou 1D), et ne pas faire de boucle sur leurs éléments. Voici deux astuces, la première utilisée dans le script ci-dessus, l'autre à utiliser dans votre fonction.

Astuce 1. Calcul du tableau 2D `pex`. Pour matlab, `x` et `y` sont deux tableaux de tailles $1 \times i_{max}$ et $1 \times i_{max}$ (comme l'indique la commande `size(x)`). Comme on veut que p_{ex} soit de taille $i_{max} \times j_{max}$, on écrit l'instruction

```
pex=sin(a*x)'*cos(b*y)
```

qui fait le produit *matriciel* (symbole `*`) de la transposée de `sin(a*x)` (symbole `'`) et de `cos(b*y)`. On a donc un tableau de taille $(i_{max} \times 1) * (1 \times j_{max}) = i_{max} \times j_{max}$.

Astuce 2. Pour coder l'instruction $\hat{p}_{k,l} = \frac{1}{c_{x,k}+c_{y,l}} \hat{f}_{k,l}$, on va définir `fc=fft2(f)` (c'est $\hat{f}$), qui est un tableau de même taille que f, c'est-à-dire $i_{max} \times j_{max}$. Par contre, `cx`, qui va contenir les $c_{x,k}$, sera un tableau de taille $1 \times i_{max}$ (comme `x`) et `cy` de taille $1 \times j_{max}$. Le plus simple est donc de transformer `cx` en un tableau de i_{max} lignes et j_{max} colonnes en répliquant j_{max} fois le tableau `cx`, ce qui se fait ainsi :

```
Cx=repmat(cx',1,jmax);
```

et de même pour `cy` :

```
Cy=repmat(cy,imax,1);
```

Ainsi, on calcule $\hat{p}_{k,l}$ avec l'instruction

```
pc= fc./(Cx+Cy)
```

Pour vous convaincre que cela correspond bien à la formule donnée plus haut, vérifiez sur le papier que l'élément `pc(k,l)` est bien $\hat{p}_{k,l}$.

4 Résolution des équations de Navier-Stokes

Les outils étudiés aux sections précédentes vous permettent maintenant de résoudre les équations de Navier-Stokes en formulation PPE.

Données physiques et géométrie Le domaine de calcul est $[0, L_x = 2] \times [0, L_y = 1]$. Le maillage est cartésien à pas uniforme avec 85 mailles en x et 66 en y. Il faut créer les tableaux `x` et `y` qui contiennent les coordonnées des centres des mailles, à l'aide des commandes vues précédemment.

La vitesse initiale du fluide est donnée par

$$u_y(x, y) = 0 \quad \text{et} \quad u_x(x, y) = U_0 \left(1 + \tanh\left(\frac{P}{2} \left(1 - |y - L_y/2|/R \right) \right) \right) /2,$$

avec les paramètres

$$U_0 = 1, \qquad R = L_y/4, \qquad P = 20.$$

Pour définir le tableau u_x facilement, il faut construire un tableau bidimensionnel stockant les ordonnées de tous les points du maillage, ce qui se fait ainsi (on pourrait aussi utiliser l'instruction `repmat` vue précédemment) :

```
[xx,yy]=meshgrid(x,y);xx=xx';yy=yy';
```

Visualisez enfin le champ u_x à l'aide de la commande suivante :

```
pcolor(x,y,ux'); axis equal; axis tight; shading('interp')
```

qui montre les valeurs de u_x en tout point du domaine avec un code de couleur. On peut obtenir ce code avec la commande `colorbar`. Pour comprendre le rôle de chaque commande, tapez les les unes après les autres. Les deux commandes

```
axis
```

donnent la même échelle en x et y et réduisent le tracé au minimum. La commande `shading('interp')` interpole les valeurs de u_x pour une visualisation plus lisse.

La viscosité du fluide sera $\nu = 0.001$. La pression initiale sera prise égale à 0 dans tout le domaine.

Discrétisation des équation de Navier-Stokes Rappelons les équations :

$$\begin{aligned} &\partial_t u + (u \cdot \nabla)u + \nabla p = \nu \Delta u, \\ &\Delta p = -\nabla \cdot ((u \cdot \nabla)u), \\ &u \text{ et } p \text{ périodiques sur } \Omega \end{aligned}$$

Il suffit alors d'écrire une boucle en temps avec la structure

```
for n=1:50

% calcul du pas de temps dt

% calcul de ux et uy par RK3 au temps tn+1
% (avec résolution de l'équation de Poisson à chaque étage du RK3)
```

```
% calcul de p au temps tn+1

% visualisation de la solution

end
```

Le pas de temps sera calculé en utilisant une formule analogue à celle donnée par l'analyse de stabilité de l'équation de convection diffusion vue précédemment :

```
dt=0.9*min([1/(2*nu*(1/dx^2+1/dy^2)),sqrt(3)/(max(max(abs(ux)/dx+abs(uy)/dy)))]);
```

L'équation sur u est traitée séparément pour u_x et u_y comme pour l'équation de convection diffusion (à étendre à la dimension 2).

L'équation de Poisson a déjà été vue à la section précédente. Il suffit de rajouter la discrétisation du second membre : avec les différences finies et l'astuce sur les tableaux d'indices, cela s'écrit en 4 lignes seulement. Notez qu'il faudra tout de même utiliser 2 tableaux d'indices supplémentaires en i

```
ip2=[3:imax,1,2]; im2=[imax-1,imax,1:imax-2];
```

et de même en j.

Pour visualiser la solution, une première possibilité est de tracer la vorticité $w = \partial_y u_x - \partial_x u_y$, discrétisée par différences finies. Cette fonction permet de localiser les tourbillons. La commande associée est alors

```
pcolor(x,y,w'); axis equal; axis tight; shading('interp')
```

Executez alors votre programme : vous devez constater qu'il ne se passe absolument rien, la solution ne bouge pas !

Génération de l'instabilité Pour voir les instabilités de Kelvin-Helmoltz, il faut perturber légèrement la donnée initiale. On va donc rajouter la ligne suivante à la suite de la définition du `ux` initial :

```
Ax=0.25; lamx=0.5*Lx; ux=ux+Ax*ux.*sin(2*pi/lamx*xx);
```

Ces instructions rajoutent une oscillation de la composante horizontale de la vitesse dans la direction x.

En relançant votre programme, vous devez voir que les oscillations initiales (invisibles au début) s'amplifient et donnent naissance à deux beaux tourbillons en haut et deux en bas.

Utilisation d'un traceur En pratique, ce que l'on verrait en regardant une telle instabilité, ce sont plutôt les trajectoires des particules fluides. Par exemple, si le jet est coloré en rouge et le reste du fluide en bleu, on verra facilement le développement des instabilités. On peut simuler ceci à l'aide d'un traceur numérique : il s'agit d'une fonction $\phi(t,x,y)$ définie à l'instant initial par $\phi(0,x,y) = 2 + \tanh(\frac{P}{2}(1 - |y - L_y/2|/R)))/2$, une fonction similaire à celle définissant $u_x(0,x,y)$, qui vaut près de 3 dans le jet, et près de 1 à l'extérieur. Ensuite, on souhaite que cette fonction soit transportée à la vitesse du fluide : ainsi, en tout temps, les valeurs proches de 3 de cette fonction représenteront les particules fluides issues du jet initial, ce qui permet de suivre l'évolution de ce jet. L'évolution de ϕ est donc régie par l'équation de convection

$$\partial_t \phi + u \cdot \nabla \phi = 0.$$

Vous pouvez résoudre cette EDP avec le même schéma que celui utilisé pour l'équation de la vitesse de Navier-Stokes (RK3 plus schéma centré). Cependant, la dispersion numérique liée au

schéma centré va rapidement produire de petites oscillations qui vont déteriorer la solution. Si l'on utilise un schéma décentré, tel que celui d'ordre un vu en cours, la dissipation numérique va trop diffuser l'interface entre le jet et le fluide environnant. L'idéal serait d'utiliser un schéma décentré d'ordre élevé, mais cela serait trop long dans le cadre de ce mini-projet. Une idée simple consiste donc à rajouter de la diffusion physique (c.-à-d. le second membre $\mu\Delta\phi$), ce qui revient à considérer que le colorant utilisé diffuse peu à peu dans le fluide. Je vous suggère de prendre pour μ la même valeur que la le coefficient de diffusion du fluide ν, ce qui a l'avantage de diffuser assez peu l'interface durant la durée de la simulation, tout en permettant d'utiliser la même CFL que pour les équations de Navier-Stokes.

Programmez alors le schéma, et testez le dans votre script, en rajoutant la commande

```
pcolor(x,y,phi'); axis equal; axis tight; shading('interp')
```

Pour éviter que les deux visualisations (de w et ϕ) se fassent dans la même fenêtre, faites précéder celle de w par la commande `figure(1)` et celle de ϕ par la commande `figure(2)`. Les images obtenues pour ϕ ressemblent beaucoup aux instabilités de Kelvin-Helmoltz observées dans la nature.