Simulation numérique : sillage d'un obstacle confiné

Quelles sont les questions scientifiques ou techniques ?
Instabilité du sillage derrière un obstacle : seuil d'instabilité, fréquence et amplitude des oscillations au-delà du seuil.

Par quelles expériences y répondre ?
Expérience modèle avec un obstacle rectangulaire confiné dans un canal (comme pour l'expérience de vélocimétrie laser).

Quelles techniques expérimentales ?
Simulation numérique par éléments finis.
Comparaison avec les expériences réalisées par vélocimétrie laser.

Quels sont les résultats ?
À vous de les montrer à travers des graphes clairs.

Comment les interpréter ?
Ingrédients physiques, lois d'échelle, ajustement de courbes expérimentales : à vous de jouer !

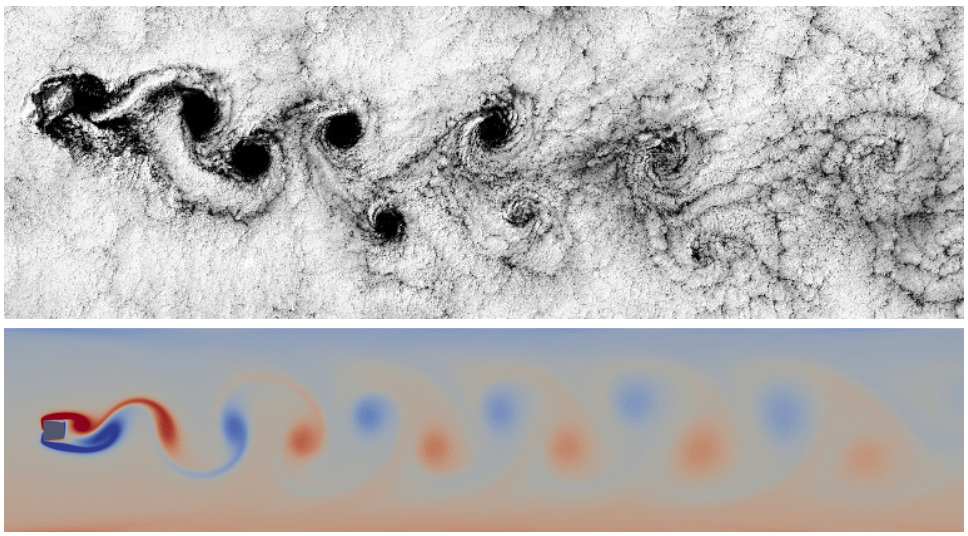

FIGURE 1 – Allée de vortex atmosphérique dans le sillage de l'île de Robinson Crusoé (archipel Juan Fernandez, Chili). Simulation numérique sous FreeFem (à un nombre de Reynolds modéré).

Le sillage d'un objet non profilé, de section circulaire ou carrée, présente une instabilité qui apparaît à des nombres de Reynolds de l'ordre de quelques dizaines. Cette instabilité, dite de Bénard-Von Karman, se manifeste par une allée de tourbillons alternés. Cette structure de tourbillons alternés persiste jusqu'à des nombres de Reynolds extrêmement grands puisqu'elle est observée dans le sillage atmosphérique d'iles élevées, sur des échelles de plusieurs dizaines de kilomètres (photo ci-dessus). Le but de cette étude numérique est essentiellement de caractériser cette instabilité près de son seuil et, accessoirement, de voir comment elle est influencée par le confinement entre deux parois, cette géométrie étant utilisée pour réaliser des débitmètres sans partie mobile.

La manière la plus simple d'étudier cette instabilité est de faire varier le paramètre de contrôle (en l'occurrence le nombre de Reynolds) et d'observer la structure d'écoulement correspondante. On peut ainsi définir (mais de manière peu précise) le nombre de Reynolds critique Re_c correspondant au seuil d'instabilité et les caractéristiques des modes instables (amplitude, fréquence temporelle et spatiale). Il est possible d'analyser plus finement la nature de l'instabilité en soumettant l'écoulement à des perturbations bien contrôlées (impulsion localisée dans le temps, excitation sinusoïdale, créneau dans la vitesse moyenne) et en suivant la réponse du sillage à ces perturbations. C'est ce qui a été fait dans une étude expérimentale qui fait référence sur ce problème [3] et que nous allons reproduire numériquement.

On simule ici un problème bidimensionnel, c'est à dire qu'on suppose qu'il n'y a que deux composantes non nulles du champ de vitesse et que le problème est invariant dans la troisième dimension de l'espace. Le schéma du domaine de calcul est représenté sur la fig. 2. Le diamètre de l'obstacle est pris égal à 1. La largeur du canal *bloc* définit le rapport de blocage. Le centre de l'obstacle est placé à deux largeurs de canal en aval de l'entrée et la longueur du canal est au moins égale à 10 fois la largeur.

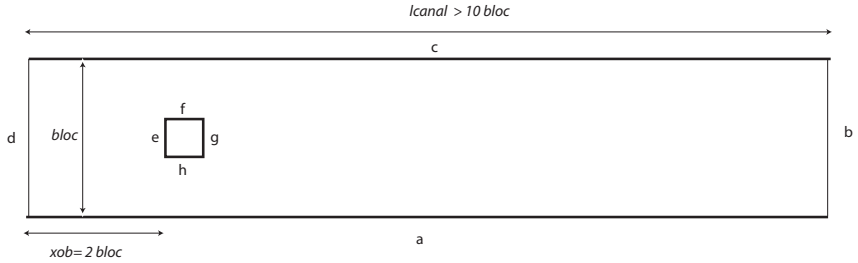

FIGURE 2 – Schéma du domaine de calcul pour un obstacle de section carrée, avec identification des différentes frontières (a-h).

L'origine des coordonnées (0,0) est le point en bas à gauche du domaine de calcul.

Le logiciel utilisé pour le calcul est FreeFem++, un logiciel domaine public développé au laboratoire d'analyse numérique de l'UPMC. Les sources ainsi que les exécutables pour Unix, Linux, Windows et MacOsX sont disponibles gratuitement.

FreeFem++ permet de résoudre des systèmes d'équations différentielles par les méthodes des éléments finis en utilisant une formulation variationnelle. Il permet également de générer automatiquement des maillages triangulés du domaine de calcul.

1 Expérience numérique

On se propose ici d'étudier quelques aspects de l'instabilité de Bénard-von Karman, en fixant le rapport de confinement, c'est-à-dire le rapport entre la largeur de l'obstacle et la largeur du canal :

- Quel est le seuil d'instabilité en nombre de Reynolds ? Ce seuil est-il comparable à celui déterminé expérimentalement par vos camarades ?

- Au delà du seuil d'instabilité, quelle est la fréquence des tourbillons émis et comment varie-t-elle avec le nombre de Reynolds ? Comment varie la force exercée par l'écoulement sur l'obstacle ? Les simulations rendent-elles compte des résultats expérimentaux obtenus pas vélocimétrie laser ? Plus généralement, comment passer des résultats numériques adimensionnés à des "vraies" grandeurs physiques ?

- Comment croît l'instabilité dans le sillage en fonction du temps ? Quel est le taux de croissance en fonction de l'écart (positif ou négatif) au seul d'instabilité ? Comment évolue-t-il avec Re au voisinage de Re_c ?

Finit-on par obtenir un régime permanent ? Dans ce cas, comment varie l'amplitude finale des oscillations en fonction du nombre de Reynolds ? Une théorie un peu poussée sur les instabilités suggère une évolution de l'amplitude finale sous la forme $A \sim A_0(Re - Re_c)^{1/2}$. Est-ce compatible avec vos résultats ? Pour répondre à cette question, vous pourrez utiliser la fonctionnalité du programme qui permet d'imposer divers types de perturbation sur le débit en entrée du canal.

- Comment pourriez-vous réaliser un débitmètre, ne comportant aucune partie mobile, en exploitant cette instabilité ?

1.1 Utilisation de FreeFem sous MacOs

Le calcul est entièrement défini dans un fichier texte "BVK_perturb_2019.edp" détaillé ci-dessous. Ce fichier peut être modifié avec n'importe quel éditeur de texte. Sous MacOs, l'éditeur Smultron présente l'avantage de colorer le texte en fonction de la syntaxe du programme.

Pour démarre le calcul, il faut ouvrir une fenêtre terminal, aller dans le repertoire où se trouve le fichier BVK_perturb_2019.edp et exécuter la commande : `FreeFem++ BVK_perturb_2019.edp`. Les paramètres du calcul sont rentrés dans cette fenêtre terminal. Au cours du calcul, lorsque la commande `plot` est utilisée, FreeFem ouvre une autre fenêtre pour afficher le maillage, les champs de pression, de vitesse, ...

En fin de calcul, il faut appuyer sur la touche escape pour arrêter le programme

Les résultats du calcul pour post-traitement sont sauvegardés dans divers fichiers qui se trouvent dans le même répertoire que le fichier exécuté par FreeFem. Il est recommandé de stocker ensuite ces fichiers dans un répertoire séparé pour chaque cas de calcul.

A la fin du nombre d'itérations demandées, le maillage ainsi que les champs de vitesse et de pression sont enregistrés sur le disque. Nom des fichiers de la forme : `mesh_b+rapport blocage+_re+reynolds+_t+nb iterations+.msh` et `pu_b+rapport blocage+_re+reynolds+_t+nb iterations+.sol`. Pour redémarrer un calcul et effectuer un plus grand nombre d'itérations, ces fichiers peuvent être relus par la commande `FreeFem++ BVK_restart_2019.edp`.

1.2 Paramètres à rentrer pour le calcul

La largeur de l'obstacle est prise égale à 1, ce qui revient à adimensionner toutes les longueurs par la largeur de l'obstacle. La vitesse moyenne de l'écoulement en entrée est prise égale à 1, ce qui revient à adimensionner les vitesse par la vitesse moyenne.

Les paramètre suivants sont à rentrer pour le calcul :

1. Définition du maillage
   - Rapport de blocage : largeur du canal/largeur de l'obstacle
   - Longueur du canal
   - Paramètre de définition du maillage : nb de mailles par unité de longueur sur les bords du canal
   - Raffinement du maillage sur l'obstacle : facteur de multiplication du nb de mailles sur l'obstacle
   - Raffinement automatique du maillage au cours du calcul (oui ou non. Si oui, le maillage est adapté à chaque pas de temps en fonction du gradient de vitesse local. Donne une meilleur résolution du champ de vitesse, mais ralentit le calcul).
2. Paramètres physiques
   - Nombre de Reynolds
   - Perturbation de l'écoulement (oui ou non)
   - Pas de temps de calcul (en temps adimensionné)
   - Nombre d'itérations à effectuer
3. Sortie des données pour post-traitement
   - Position x et y du point de mesure de la vitesse en fonction du temps. Ces coordonnées sont en position relative par rapport au centre de l'obstacle. Les deux composantes de vitesse à chaque pas de temps sont enregistrées dans un fichier

— Calcul des forces sur l'obstacle (oui ou non. Si oui, les deux composantes de force sur l'obstacle sont calculées à chaque pas de temps et enregistrées dans un fichier).
— Enregistrement des champs de pression (oui ou non. Si oui, tous les 10 pas de temps, le champ de pression est affiché à l'écran et enregistré dans un fichier lisible avec Paraview)
— Enregistrement des lignes de courant (oui ou non. Si oui, tous les 10 pas de temps, le lignes de courant sont affichées à l'écran et le champ de vitesse est enregistré dans un fichier lisible avec Paraview)
— Enregistrement du champ de vorticité (oui ou non. Si oui, tous les 10 pas de temps, le champ de vorticité est affiché à l'écran et enregistré dans un fichier lisible avec Paraview)
— Sortie graphiques sur une partie du domaine de calcul (oui ou non. Si oui, rentrer les limites (point inférieur gauche et point supérieur droit) du rectangle qu'on va afficher à l'écran. La totalité du domaine de calcul reste sauvegardée dans les fichiers.)

1.3 Post-traitement des données

A la fin du calcul les fichiers enregistrés sur le disque sont :
— un fichier dont le nom est de la forme :
`"u_vs_t_b"+bloc+"_re"+reynolds+".txt"`
où bloc est la valeur du rapport de blocage. 3 colonnes : numéro d'itération, deux composantes de vitesse u_x et u_y au point de mesure défini au cours du calcul.
— si on calcule les forces sur l'obstacle, un fichier dont le nom est de la forme :
`"f_vs_t_b"+bloc+"_re"+reynolds+".txt"`
. 3 colonnes : numéro d'itération, deux composantes de force f_x et f_y
— si on a décidé d'enregistrer les champs de pression, de vitesse ou de vorticité, des séries de fichiers dont les noms sont de la forme
`"p_b"+bloc+"_re"+reynolds+"_t"+i+".vtk"`
ou
`"vit_b"+bloc+"_re"+reynolds+"_t"+i+".vtk"`
ou
`"vor_b"+bloc+"_re"+reynolds+"_t"+i+".vtk"`
— le maillage (fichier .msh) et la solution complète, vitesse et pression (fichier .sol) à la dernière itération.

Les données vitesse locale et force en fonction du temps peuvent être importées directement dans Matlab.

Les fichiers .vtk peuvent être lus avec le logiciel Paraview pour examiner l'évolution temporelle des solutions et réaliser éventuellement des animations.

2 Formulation variationnelle

Le logiciel FreeFem est conçu pour résoudre n'importe quel type d'équation différentielle pourvu qu'elle puisse être écrite sous forme variationnelle.

Pour décrire ce qu'est la formulation variationnelle [1], considérons le problème d'une poutre flexible, de longueur unité soumise à une charge répartie f. Le moment fléchissant local $u(x)$ est décrite par l'équation différentielle de Poisson qui s'écrit, une fois adimensionnée :

$$-u"(x) = f(x) \tag{1}$$

avec les conditions aux limites $u(0) = u(1) = 0$. En multipliant cette équation par une fonction continue et différentiable $v(x)$, on obtient :

$$-\int_0^1 u"(x)v(x)dx = \int_0^1 f(x)v(x) \tag{2}$$

En intégrant par parties le membre de gauche, on a :

$$\int_0^1 u'(x)v'(x)dx - [u"v]_0^1 = \int_0^1 f(x)v(x) \tag{3}$$

Si v est choisi telle que $v(0) = v(1) = 0$:

$$\int_0^1 u'(x)v'(x)dx = \int_0^1 f(x)v(x) \tag{4}$$

Le problème variationnel consiste à trouver la fonction u qui satisfait l'équation 4 quelle que soit la fonction v.

La méthode des éléments finis [2] consiste à trouver une solution sous la forme : $u(x) = \sum_{i=1}^{N} u_i\phi_i(x)$ où les fonctions ϕ_i sont N fonctions linéairement indépendantes et u_i sont N nombres réels à déterminer. En prenant $v = \phi_j$, le problème 4 se ramène à trouver les nombres u_i tels que :

$$\sum_{i=1}^{N} u_i \left(\int_0^1 \phi_i'(x)\phi_j'(x)dx \right) = \int_0^1 f(x)\phi_j'(x)dx \tag{5}$$

pour tout j compris entre 1 et N. On définit la matrice de rigidité $N \times N$ A telle que $A_{ji} = \int_0^1 \phi_i'(x)\phi_j'(x)dx$. Si $\mathbf{u}$ est le vecteur de composantes u_i et $\mathbf{f}$ le vecteur de composantes $f_j = \int_0^1 f(x)\phi_j'(x)dx$, le problème se ramène à résoudre le système linéaire :

$$A\mathbf{u} = \mathbf{f} \tag{6}$$

et donc à inverser la matrice A.

Dans la méthodes des éléments finis de degré 1, on divise l'intervalle $[0, 1]$ en $N + 1$ parties de largeur $h = 1/(N + 1)$ et commençant respectivement en $x_i = ih$. On définit les fonctions $\phi_i(x)$ "triangulaires"telles que :

$$\phi_i(x) = \begin{cases} \frac{x-x_{i-1}}{h} & \text{si } x_{i-1} < x < x_i \\ \frac{x-x_{i+1}}{h} & \text{si } x_i < x < x_{i+1} \\ 0 & \text{si } x <= x_{i-1} \text{ ou } x >= x_{i+1} \end{cases} \tag{7}$$

Ceci revient à faire une approximation de la solution par un ensemble de N fonctions affines par morceaux.

De la même manière, on peut définir des éléments de degré 2 où les fonctions ϕ_i sont des fonctions quadratiques de x, la solution approchée étant une succession d'arcs de parabole. Ces résultats à une dimension peuvent être étendus aux cas bidimensionnel et tridimensionnel.

3 Programme utilisé pour le calcul

3.1 Définition du domaine de calcul

Cette première partie permet de définir le domaine de calcul. Les différentes frontières sont définies par des équations paramétriques. Le domaine de calcul se trouve à gauche de la frontière décrite dans le sens trigonométrique. Notez que les sens de parcours permettent d'exclure l'intérieur de l'obstacle du domaine de calcul.

```
load "iovtk" ;
load "UMFPACK64";
real bloc, lcanal , lcrel;
cout << " Entrer le rapport de blocage (largeur du canal/largeur de l'obstacle >1) :"; cin >> blo
assert (bloc>1);
// la longueur du canal est au moins 10 fois la largeur de l'obstacle
```

```
cout << " Entrer la longueur du canal (>10 et < 150) :"; cin >> lcanal;
assert (lcanal>10);
assert (lcanal<150);
// definition du nombre minimal de mailles en fonction de lcanal et bloc
int nbloc,ncanal ;
nbloc=floor(bloc);
ncanal=floor(lcanal);
// limites du canal
// fronti?eres d?ecrites dans le sens trigo direct
border a(t=0,lcanal) {x=t;y=0;label=1;}; //bas
border b(t=0,bloc) {x=lcanal;y=t;label=2;}; // sortie
border c(t=0,lcanal) {x=lcanal-t;y=bloc;label=3;}; // haut
border d(t=0,bloc) {x=0;y=bloc-t;label=4;}; // entree
// obstacle
// frontie?re d?ecrite dans le sens trigo inverse
// le centre de l'obstacle est place? ? 2 largeurs en aval de l'entr?ee du canal
int C1=99,top=97,bottom=96 ;
real xob,yob;
xob=2.*bloc;
yob=0.5*bloc;
// section trapèze
border e(t=-0.5, 0.5) {x=xob;y=yob+t;label=5;};
border f(t=0, 1) {x=xob+t;y=yob+0.5-0.1*t;label=5;};
border g(t=0.4, -0.4) {x=xob+1;y=yob+t;label=5;};
border h(t=0, 1) {x=xob+1-t;y=yob-0.4-0.1*t;label=5;};
```

3.2 Définition du maillage

Cette seconde partie permet de définir le maillage et les espaces d'éléments finis. Le maillage triangulaire est défini à partir du nombre de noeuds sur chacune des frontières du domaine de calcul. Ici un paramètre défini par l'utilisateur (n) permet de raffiner plus ou moins le maillage.

On définit les champs de vitesse ainsi que le champ de vorticité et les composantes du tenseur des contraintes sur l'espace Xh et le champ de pression sur l'espace Mh.

On utilise des éléments finis d'ordre 2 pour les champs de vitesse et de vorticité et des éléments finis d'ordre 1 pour le champ de pression. Le champ de vitesse est donc mieux résolu que le champ de pression. Ceci est lié au fait que la pression intervient sous forme d'un gradient dans l'équation de Navier-Stokes, alors que le terme de viscosité fait intervenir le laplacien de la vitesse.

```
int i=0;
int n=1;
int nr=1;
cout << " Entrer la resolution du maillage (>1) :"; cin >> n;
cout << " Entrer le raffinement sur l'obstacle (>1) :"; cin >> nr;
mesh th = buildmesh(a(ncanal*n)+b(nbloc*n)+c(ncanal*n)+d(nbloc*n)+e(nr*n)+f(nr*n)+g(nr*n)+h(nr*n)
//plot (th,wait=1,ps="maillage.ps") ;
plot (th,wait=1) ;
// determination de la surface min,max et moyenne des mailles
int nbtriangles=th.nt;
real minarea=100000., maxarea=0., avarea=0.;
real minlength=100000., maxlength=0., avlength=0.;
for (i=0;i<nbtriangles;i++){
avarea=avarea+th[i].area;
```

```
minarea=min(minarea,th[i].area);
maxarea=max(maxarea,th[i].area);
}
avarea=avarea/nbtriangles;
cout << "surface min : " << minarea << " surface max : " << maxarea << " surface moyenne : " << a
minlength=sqrt(minarea);
maxlength=sqrt(maxarea);
avlength=sqrt(avarea);


// d?efinition des espaces d'?el?ments finis
// P2 ?ele?ments quadratiques continus par morceaux

fespace Xh(th,P2); // velocity space
fespace Mh(th,P1); // pressure space
// u1,u2,p sont les deux composantes de vitesse et la pression calcul?ees
Xh u2,v2;
Xh u1,v1;
Xh vor, sigma11, sigma22, sigma12 ;
Mh p,q;
```

3.3 Problème de Stokes

On définit le profil de vitesse en entrée du canal. On prend un profil parabolique, solution du problème à petit nombre de Reynolds. La vitesse moyenne est prise égale à 1, de manière à normaliser correctement les quantités calculées ultérieurement. On définit la forme variationnelle de l'équation de Stokes ($\eta\Delta\mathbf{u} = \nabla p$), le champ de vitesse étant à divergence nulle pour respecter la condition d'incompressibilité (cf § 9.6.1 de la documentation de FreeFem). La commande `solver=UMFPACK` définit l'algorithme utilisé pour la résolution du système linéaire. On ajoute les conditions aux limites : profil parabolique définit par la fonction uin en entrée et en sortie (frontières 2 et 4), vitesse nulle sur les parois du canal et sur l'obstacle (frontières 1,3 et 5).

```
// de?finition de la condition aux limites en entr?e du canal
// profil de vitesse parabolique
// u1(y=0) = 0 ; u1(y=bloc) = 0 : vitesse moyenne <u1> = 1
func uin=6*y*(bloc-y)/(bloc*bloc) ;
//func uin=1 ;



// calcul de la solution de Stokes pour initialiser le champ de vitesse

solve Stokes ([u1,u2,p],[v1,v2,q],solver=UMFPACK) =
    int2d(th)( ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
            +  dx(u2)*dx(v2) + dy(u2)*dy(v2) )
            - p*q*(0.000001)
            - p*dx(v1)- p*dy(v2)
            - dx(u1)*q- dy(u2)*q
           )
+ on(2,4,u1=uin,u2=0)
+ on(1,3,5,u1=0,u2=0);
```

3.4 Calcul de la fonction de courant

Pour visualiser le champ de vitesse, il est commode de tracer les lignes de courant plutôt que d'afficher les vecteurs vitesse sur chaque élément du maillage. Dans un écoulement bidimensionnel incompressible, on peut définir une fonction de courant ψ telle que : $u_x = \partial\psi/\partial y$ et $u_y = -\partial\psi/\partial x$ ou encore : $-\Delta\psi = \text{rot}\mathbf{u}$. Les lignes de courant sont les lignes d'égale valeur de ψ. La commande `plot` permet d'afficher ces lignes de courant.

```
// calcul et affichage de la fonction de courant
Xh psi,phi;
problem streamlines(psi,phi) =
int2d(th)( dx(psi)*dx(phi) + dy(psi)*dy(phi))
+ int2d(th)( -phi*(dy(u1)-dx(u2)))
+ on(a,psi=0);

streamlines ;
plot(psi,nbiso=50,wait=1);
```

3.5 Paramètres physiques du calcul

Définition du nombre de Reynolds et des éventuelles perturbations imposées sur la vitesse moyenne en entrée du canal.

Si on perturbe l'écoulement moyen, il y a trois possibilités :

— une impulsion brève (de forme gaussienne). Ce type de perturbation permet essentiellement de mesurer le temps d'amortissement des fluctuations lorsqu'on se trouve en dessous du seuil d'instabilité. Il faut définir l'amplitude de l'impulsion comprise entre 0 et 1, la vitesse moyenne étant égale à une unité. La position en temps de l'impulsion doit être définie après un régime transitoire que l'on pourra observer sur le calcul en l'absence de perturbation. Il faut que l'écoulement soit bien établi et stable avant l'application de l'impulsion. L'unité de temps correspond au temps nécessaire pour parcourir la largeur de l'obstacle. Enfin on définit la largeur en temps de l'impulsion. Cette largeur doit être typiquement inférieure à l'unité de temps.

— une variation brusque de la vitesse moyenne. Ce type de perturbation permet de mesurer le temps nécessaire à l'écoulement pour se réorganiser à la nouvelle valeur du paramètre de contrôle (en l'occurrence le nombre de Reynolds). Il faut entrer la nouvelle valeur du nombre de Reynolds (différente de celle utilisée pour lancer le calcul), le temps adimensionné auquel le changement de vitesse moyenne se produit (ce temps doit être assez grand pour que l'écoulement soit bien établi) et la durée sur laquelle le changement se produit (ce dernier temps doit être assez court, inféroeir ou égal à l'unité)..

— une perturbation sinusoïdale. En imposant une perturbation à une fréquence différente de la fréquence naturelle d'émission des tourbillons, on peut étudier la courbe de résonance du sillage. Il faut définir l'amplitude de la perturbation de vitesse (la vitesse moyenne est toujours égale à une unité) et la fréquence de la perturbation (la fréquence unité correspond à la période unité qui est le temps nécessaire pour parcourir la largeur de l'obstacle). Lorsque ce type de perturbation est utilisé, le programme calcule l'amplitude quadratique moyenne des fluctuations de la composante verticale de vitesse.

```
// nu est l'inverse du nb de Reynolds
// attention ?a la d?finition de Reynolds.
real reynolds ;
cout << " Entrer le nombre de Reynolds :"; cin >> reynolds;
real nu=1./reynolds;

// d?efinition des perturbations de vitesse
```

```
real amp1=0.,amp2=0.,ref=reynolds,ampm,tp=0.,dtp =1.,tpm=0.,dtpm=1.,omega=1.;
bool pert=false, pulse=false, sinus = false , marche =false ;
string yes ;
cout << "Perturbation de l'ecoulement ? (o, n)"; cin >> yes;
pert = (yes == "o");
if (pert) {
cout << " Definition des parametres des perturbations de l'ecoulement";
cout << " Impulsion ? (o, n) :"; cin >> yes;
pulse = (yes == "o");
if (pulse) {
cout << " Entrer l'amplitude de l'impulsion de vitesse (<1) :"; cin >> amp1;
cout << " Entrer la position en temps de l'impulsion  :"; cin >> tp;
cout << " Entrer la largeur en temps de l'impulsion :"; cin >> dtp;
}
dtp=1/(dtp*dtp) ;
cout << " Variation de vitesse moyenne ? (o, n) :"; cin >> yes;
marche = (yes == "o");
if (marche){
cout << " Entrer le nb de Reynolds final :"; cin >> ref;
cout << " Entrer la position en temps de la variation  :"; cin >> tpm;
cout << " Entrer la largeur en temps de la variation  :"; cin >> dtpm;
}
ampm=ref/reynolds-1 ;
cout << " Perturbation sinusoidale ? (o, n) :"; cin >> yes;
sinus = (yes == "o");
if (sinus){
cout << " Entrer l'amplitude de la perturbation (<1) :"; cin >> amp2;
cout << " Entrer la frequence de la perturbation  :"; cin >> omega;
}
omega=omega*2*pi ;
}
```

3.6 Problème de Navier-Stokes instationnaire

Dans cette partie, on définit le problème de Navier-Stokes instationnaire $\rho(\partial_t u + u.\nabla u) = -\nabla p + \eta\Delta u$. La taille de l'obstacle et la vitesse moyenne ayant été prises égales à 1, le nombre de Reynolds du problème se réduit à : $Re = 1/\nu$. L'équation de Navier-Stokes, s'écrit donc, en variables adimensionnées : $\partial_t u + u.\nabla u = -\nabla p + \nu\Delta u$, la pression étant normalisée par la pression dynamique ρU^2. De la même façon que pour l'équation de Stokes, FreeFem résout une forme variationnelle de l'équation (cf § 9.6.1 de la documentation de FreeFem). Pour calculer le terme instationnaire, il faut définir un pas de temps `dt` (en temps adimensionné). On vérifie que le pas de temps est assez petit pour que le déplacement sur un pas de temps soit plus petit que la taille d'une maille. Les conditions aux limites en entrée prennent en compte une éventuelle modulation temporelle de la la vitesse moyenne.

```
// d?efinition du pas de temps
real dt=0.1;
bool courantnotok=true;
while (courantnotok){
cout << " Entrer le pas de temps : (<1) "; cin >> dt;
cout << "nombres de Courant sur long. min : " << dt/minlength << " sur long. max : " << dt/maxlen
cout << " pas de temps de correct ? (o,n) "; cin >> yes;
if (yes == "o") {
```

```
courantnotok=false;
}
}
real alpha=1/dt;
// d?efinition du proble?me de Navier-Stokes
// la condition aux limites sur la face d'entr?ee dans le canal (d) prend en compte la perturbati
// d?ependant du temps (=dt*i)
Xh up1,up2;
problem NS (u1,u2,p,v1,v2,q,solver=UMFPACK,init=i) =
int2d(th)(
alpha*( u1*v1 + u2*v2)
+ nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
+ dx(u2)*dx(v2) + dy(u2)*dy(v2) )
- p*q*(0.000001)
- p*dx(v1) - p*dy(v2)
- dx(u1)*q - dy(u2)*q
)
+ int2d(th) ( -alpha*
convect([up1,up2],-dt,up1)*v1 -alpha*convect([up1,up2],-dt,up2)*v2 )
+ on(4,u1=uin*(1+ampm*0.5*(1+tanh((dt*i-tpm)/dtpm)))+amp1*exp(-(dt*i-tp)*(dt*i-tp)*dtp)+amp2*sin(
+ on(2,u2=0)
+ on(1,3,5,u1=0,u2=0);
```

3.7 Paramètres des sorties de données

Dans cette partie, on définit les paramètres pour les sorties graphiques. Par défaut, on affichera le champ de vorticité $\omega_z = \partial_y u_x - \partial_x u_y$ de manière à visualiser les tourbillons dans le sillage de l'obstacle. On définit le fichier qui permettra d'enregistrer l'évolution temporelle de la vitesse en un point du domaine de calcul. Ensuite, on spécifie si on calcule les forces sur l'obstacle, si on enregistre le champ de pression, le champ de vitesse ou le champ de vorticité.

Par défaut les sorties graphiques à l'écran affichent tout le domaine de calcul, mais on peut spécifier une zone plus petite.

```
// d?efinition des valeurs pour la visualisation de la vorticite? de -vmax ? vmax
real[int] vorval(51);
real vmax=10;
for (i=0;i<51;i++){
vorval[i]=vmax*(-1.+0.04*i);
}
// definition des couleurs pour l'affichage de la vorticite
// les couleurs sont definies par leur coordonnees hsv (hue, saturation, value ; teinte, saturati
real[int] colors(153) ;
for (i=0;i<26;i++){
colors[3*i]=0.008*i;
//colors[3*i+1]=1-0.04*i;
colors[3*i+1]=1;
colors[3*i+2]=1;
}
for (i=26;i<51;i++){
colors[3*i]=0.5+0.008*(i-26);
//colors[3*i+1]=0.04*(i-25);
colors[3*i+1]=1;
colors[3*i+2]=1;
```

```
}

// fichier de sortie pour l'?evolution temporelle de la vitesse derri?re l'obstacle
string uvstfile="u_vs_t_b"+bloc+"_re"+reynolds+".txt" ;
ofstream uvst(uvstfile,append);
// fichier de sortie pour le'?volution temporelle de la force sur l'obstacle
string fvstfile="f_vs_t_b"+bloc+"_re"+reynolds+".txt" ;
ofstream fvst(fvstfile,append);
// nombre d'ite?rations en temps
int nbiter ;
cout << " Entrer le nombre d iterations :"; cin >> nbiter;
// d?efinition de la position du point de mesure de la vitesse
 real xmes,ymes,xrel,yrel ;
xmes = -lcanal ;
ymes = 2*bloc ;
cout << " Position x du point de mesure de vitesse par rapport a l obstacle :"; cin >> xrel;
xmes=xob+xrel;
cout << " Position y du point de mesure de vitesse par rapport a l obstacle :"; cin >> yrel;
ymes=yob+yrel;
// de?finition des param?tres pour les sorties graphiques
//string store_dir,base_dir ;
bool calcf ;
real fx,fy ;
cout << "calcul des forces sur l'obstacle ? (o, n)"; cin >> yes;
calcf = (yes == "o");
bool sortiep, sortiev, plotv, zoom, sortier;
cout << "enregistrement des champs de pression ? (o, n)"; cin >> yes;
sortiep = (yes == "o");
cout << "enregistrement des lignes de courant ? (o, n)"; cin >> yes;
sortiev = (yes == "o");
cout << "enregistrement du champ de vorticite ? (o, n)"; cin >> yes;
sortier = (yes == "o");
cout << "affichage du champ de vitesse ? (o, n)"; cin >> yes;
plotv = (yes == "o");
cout << " sorties graphiques sur une partie seulement du domaine de calcul ? (o, n)"; cin >> yes;
zoom = (yes == "o");
real llx,lly,urx,ury;
llx=0. ;
lly=0. ;
urx=lcanal ;
ury=bloc ;
if (zoom) {
cout << "coordonnee x du point inferieur gauche :"; cin >> llx;
cout << "coordonnee y du point inferieur gauche :"; cin >> lly;
cout << "coordonnee x du point superieur droit :"; cin >> urx;
cout << "coordonnee y du point superieur droit :"; cin >> ury;
}
// sauvegarde des parametres de calcul dans un fichier
string paramfile="parametres_b"+bloc+"_rei"+reynolds+"_ref"+ref+".txt" ;
ofstream param(paramfile,append);
param << "blocage " << "," << bloc << "\n";
param << "longueur relative " << "," << lcrel << "\n";
param << "longueur totale " << "," << lcanal << "\n";
param << "reynolds initial" << "," << reynolds << "\n";
```

```
param << "reynolds final" << "," << ref << "\n";
param << "temps de chgt de re" << "," << tp << "," << "largeur de marche" << "," << dtp << "\n" ;
param << "pas de temps " << "," << dt << ", nb iterations" << "," << nbiter <<"\n";
param << "bounding box " << "," << llx << "," << lly << "," << urx << "," << ury <<"\n";
param << "point de mesure ," << xmes << "," << ymes <<"\n" ;
// d?efinition des vecteurs pour l'affichage force et vitesse en fonction du temps
real[int] temps(nbiter),vitx(nbiter),vity(nbiter),fox(nbiter),foy(nbiter) ;
for (i=0;i<nbiter;i++){
temps[i]=dt*i;
vitx[i]=0 ;
vity[i]=0 ;
fox[i]=0 ;
foy[i]=0 ;
}
```

3.8 Itération en temps

Boucle de calcul proprement dite. Le temps total de calcul est `nbiter*dt`.

Noter qu'on réutilise le champ de vitesse déterminé à l'itération n (`u1,u2`) pour calculer la vitesse à l'itération $n+1$.

Pour calculer la force sur l'obstacle, on calcule les trois composantes indépendantes du tenseur des contraintes $\sigma = -pI + 2\eta(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$. En variables adimensionnées (contraintes normalisées par la pression dynamique ρU^2), on a :

$$\begin{aligned} \sigma_{xx} &= -p + \nu\frac{\partial u_x}{\partial x} \\ \sigma_{xy} = \sigma_{yx} &= \nu\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \\ \sigma_{yy} &= -p + \nu\frac{\partial u_y}{\partial y} \end{aligned} \tag{8}$$

Pour afficher les lignes de courant, on doit recalculer la fonction de courant, par la commande `streamlines` définie plus haut.

Si on veut adapter le maillage à chaque pas de temps, on utilise la commande `tadaptmesh` puis on interpole les champs calculés sur le nouveau maillage.

```
// off we go, iteration en temps
for (i=0;i<nbiter;i++){
up1=u1;
up2=u2;
NS;
x=xmes ; y=ymes ;
uvst << i << "," << u1 << "," << u2 << "\n";
vitx[i]=u1;
vity[i]=u2;
pp[i]=p;
// calcul des contraintes et de la force sur l'obstacle
if (calcf){
sigma11=2*nu*dx(u1)-p;
sigma22=2*nu*dy(u2)-p ;
sigma12=nu*(dy(u1)+dx(u2));
fx=-int1d(th,5) (sigma11*N.x+sigma12*N.y);
fy=-int1d(th,5) (sigma12*N.x+sigma22*N.y);
fox[i]=fx;
foy[i]=fy;
fvst << i << "," << fx << "," << fy << "\n";
}
```

```
// affichage des composantes x et y de force en fonction du temps
plot(cmm="Forces sur l'obstacle  iteration no "+i+" temps "+dt*i,[temps,fox],[temps,foy]);
if ( !(i % 10)) {
vor = dy(u1)-dx(u2);
if (sortiep) {
plot(cmm="iteration no "+i+" temps "+dt*i,p,nbiso=50,fill=1,bb=[[llx,lly],[urx,ury]]);
ptkfile="p_b"+bloc+"_re"+reynolds+"_t"+i+".vtk";
savevtk(ptkfile,th,p,dataname="pression");
}
if (sortiev) {
// calcul de la fonction de courant de l'ecoulement
streamlines ;
plot (cmm="iteration no "+i+" temps "+dt*i,psi,nbiso=50,bb=[[llx,lly],[urx,ury]]);
vtkfile="vit_b"+bloc+"_re"+reynolds+"_t"+i+".vtk";
savevtk(vtkfile,th,[u1,u2],dataname="vitesse");
}
if (sortier) {
plot(cmm="iteration no "+i+" temps "+dt*i,vor,viso=vorval,hsv=colors,fill=1,bb=[[llx,lly],[urx,ur
omegatkfile="vor_b"+bloc+"_re"+reynolds+"_t"+i+".vtk";
savevtk(omegatkfile,th,vor,dataname="vorticite");
}
// if (plotv) {
// affichage du champ de vitesse
// plot (cmm="iteration no "+i,[u1,u2],bb=[[llx,lly],[urx,ury]]);
// }
}
if (refinemesh) {
// adaptation du maillage et interpolation des fonctions sur le nouveau maillage
th=adaptmesh(th,u1,u2,hmin=0.01);
u1=u1;
u2=u2;
p=p;
}
}
```

3.9 Réponse à une perturbation périodique en temps

Enfin, dans le cas où on impose une perturbation sinusoïdale en entrée, on calcule la valeur quadratique moyenne de la fluctuation de vitesse transverse au point de mesure choisi dans le sillage.

```
// dans le cas d'une perturbation sinusoidale calcul de la valeur rms de la fluctuation transvers
// on ne prend pas en compte le régime transitoire au début du calcul
if (sinus) {
int nstart ;
nstart=floor(lcanal/dt) ;
real uyrms=0.,uyav=0. ;
for (i=nstart;i<nbiter;i++){
uyav=uyav+vity[i] ;
}
uyav=uyav/(nbiter-nstart) ;
for (i=nstart;i<nbiter;i++){
```

```
uyrms=uyrms+(vity[i]-uyav)*(vity[i]-uyav) ;
}
uyrms=sqrt(uyrms/(nbiter-nstart));
cout << "valeur quadratique moyenne de uy :" << uyrms ;
}
```

Références

[1] J. Rappaz, M. Picasso, Introduction à l'analyse numérique, Presses Polytechniques Romandes (1998)

[2] M.G. Larson & F. Bengzon, The Finite Element Method : Theory, Implementation and Applications, Spinger (2013)

[3] M. Provansal, C. Mathis & L. Boyer, Benard-von Karman instability : transient and forced regimes, J. Fluid Mech. 182, 1 (1987)