Université Pierre
et Marie Curie

M2 AN&EDP

Projet d'Informatique Scientifique

Résolution numérique des équations de Navier-Stokes en 2D

Auteurs :
Ethem Nayir
Olivier Tissot

Responsable du cours :
M. Frédéric Hecht

9 juin 2014

Table des matières

Remerciements

Nous tenons à remercier M.Frédéric Hecht pour sa disponibilité et sa pédagogie.

Introduction

La simulation numérique d'écoulements fluides [1] est une problématique qui intéresse fortement l'industrie [2]. Pour autant, la complexité des équations à résoudre rend leur approximation numérique parfois très difficile et c'est un domaine de recherche particulièrement fructueux.

Pour modéliser des écoulements monophasiques à une petite échelle [3], on utilise souvent les équations de Navier-Stokes. Même dans cette situation, qui simplifie parfois la réalité, on ne sait toujours pas à l'heure actuelle démontrer le caractère bien posé du problème général [4].

Une approche possible pour résoudre numériquement ces équations est d'utiliser une méthode Éléments Finis. L'idée est de réécrire la version faible du problème pour des espaces discrets bien choisis à la place des espaces de Sobolev de départ.

1. Aussi appelée CFD pour Computational Fluid Dynamic
2. Aéronautique, nucléaire...
3. De l'ordre du mètre.
4. Cela fait l'objet d'un des problèmes du Millénaire.

1 Etude préliminaire

1.1 Description du problème

Dans ce rapport, on considère un fluide incompressible de vitesse $u \in H^1(\Omega)^2$ et de pression $p \in L^2(\Omega)$, solution des équations de Navier-Stokes dans un domaine Ω de $\mathbb{R}^2$:

$$\left\{ \begin{array}{rcl} \frac{\partial u}{\partial t} + u.\nabla u - \nu\Delta u + \nabla p & = & 0 \\ \nabla.u & = & 0 \end{array} \right. \quad (1)$$

où ν est la viscosité qui sera ici constante et égale à $\frac{1}{100}$ ou $\frac{1}{400}$ (nous ferons des tests numériques avec ces 2 valeurs de ν pour faire apparaître des différences au niveau de l'écoulement du fluide et de la physique du phénomène).

Le domaine de calcul Ω sera un canal de hauteur 1 avec une marche descendante de $\frac{1}{2}$. On imposera une vitesse d'entrée parabolique de vitesse maximale unitaire, une vitesse nulle sur les parties haute et basse, et une condition de type Neumann en sortie c'est à dire : $(\nu\nabla u).n - pn = 0$ où n est le vecteur unitaire sortant. Nous avons donc imposé des conditions de type Dirichlet et Neumann au bord.

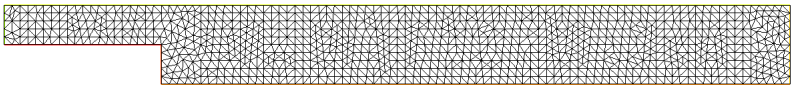

FIGURE 1 – Domaine Ω

Le but est de résoudre numériquement les équations de Navier-Stokes en utilisant les éléments finis $\mathbb{P}^1$ (pour la pression) et $\mathbb{P}^2$ (pour la vitesse) pour la discrétisation spatiale et la méthode des caractéristiques pour la discrétisation temporelle.

1.2 Formulation variationnelle associée

Soit $u \in H^1(\Omega)$ et $p \in L^2(\Omega)$ solutions du problème de Stokes :

$$\left\{ \begin{array}{rcll} -\nu\Delta u + \nabla p & = & 0 & (2.1) \\ \nabla.u & = & 0 & (2.2) \end{array} \right.$$

on ajoute à ce système d'équations des contions aux limites de type Dirichlet-Neumann précisées plus haut.

Établissons maintenant la formulation variationnelle pour le problème de Stokes. Pour l'équations (2.1), on prend $v \in H_0^1(\Omega)^2$ puis on intègre sur Ω,

on obtient grâce à la formule de Green et aux conditions aux limites :

$$\nu \int_\Omega \nabla u : \nabla v dx - \int_\Omega p \mathrm{div}(v) dx = 0$$

Pour (2.2) on prend $q \in L^2(\Omega)$ et on intègre sur Ω :

$$\int_\Omega \mathrm{div}(u) q dx = 0$$

Cependant avec de telles formulations variationnelles, nous ne pouvons pas trouver de solutions (la matrice du système ne sera pas inversible) car p est définie à une constante près alors que u est bien défini grâce aux conditions aux limites : si p est solution de Stokes alors $p+c$ est encore solution de stokes à cause du ∇p. Nous allons donc assurer l'inversibilité du système en ajoutant le terme $\int_\Omega \epsilon p v$ avec ϵ petit sur la première formulation variationnelle, cela revient à rajouter $-\epsilon$ sur la diagonale de la matrice du système. Nous obtenons au final le système de formulations variationnelles suivant :

$$\left\{ \begin{array}{rcll} \nu \int_\Omega \nabla u : \nabla v \, dx - \int_\Omega p \mathrm{div}(v) \, dx & = & 0 & \forall v \in H_0^1(\Omega)^2 \\ \int_\Omega \mathrm{div}(u) q \, dx - \epsilon \int_\Omega pq \, dx & = & 0 & \forall q \in L^2(\Omega) \end{array} \right. \quad (2)$$

Dans la prochaine partie nous aborderons la discrétisation du problème de Stokes en dimension 2. Il s'agira de trouver une solution u_h et p_h de ces formulations variationnelles dans des espaces bien adaptés.

1.3 Discrétisation du problème

Introduisons tout d'abord une triangulation régulière de Ω :

On pose $\Omega_h = \bigcup_1^{nt} K$ où nt est le nombre de triangles et on posera a_i pour $i \in \{1, ..., nbv\}$ les sommets de cette triangulation.

Dans notre rapport nous prendrons les espaces de discrétisation suivant :

$$X_h = \{v_h \in \mathcal{C}^0(\Omega_h) / \forall k \in \{1, ..., nt\} \ v_h \mid_K \in \mathbb{P}^2\}$$

$$Y_h = \{q_h \in \mathcal{C}^0(\Omega_h) / \forall k \in \{1, ..., nt\} \ p_h \mid_K \in \mathbb{P}^1\}$$

$\mathbb{P}^k$ est l'ensemble des polynômes de deux variables de degré total au plus k. Il est à noter que nous pouvons pas prendre les espaces $\mathbb{P}^1/\mathbb{P}^1$ à cause de la condition inf-sup : si on prend $\mathbb{P}^1$ en pression on doit au moins prendre $\mathbb{P}^2$ en vitesse.

Rappelons que pour l'espace $\mathbb{P}^1$ si $(\hat{x}, \hat{y}) \in \widehat{K}$ les fonctions de base sont les suivantes :

$$\begin{array}{rcl} \hat{\lambda}_0(\hat{x}, \hat{y}) & = & 1 - \hat{x} - \hat{y} \\ \hat{\lambda}_1(\hat{x}, \hat{y}) & = & \hat{x} \\ \hat{\lambda}_2(\hat{x}, \hat{y}) & = & \hat{y} \end{array}$$

Plus généralement, les $\lambda_i(x,y)$ sont les coordonnées barycentriques du point (x,y) par rapport au centre de gravité du triangle considéré. Ainsi, pour $\mathbb{P}^2$ les fonctions de base sont :

$$\begin{aligned}
\phi_0(x,y) &= \lambda_0(x,y)(2\lambda_0(x,y)-1) \\
\phi_1(x,y) &= \lambda_1(x,y)(2\lambda_1(x,y)-1) \\
\phi_2(x,y) &= \lambda_2(x,y)(2\lambda_2(x,y)-1) \\
\phi_3(x,y) &= 4\lambda_0(x,y)\lambda_1(x,y) \\
\phi_4(x,y) &= 4\lambda_0(x,y)\lambda_2(x,y) \\
\phi_5(x,y) &= 4\lambda_1(x,y)\lambda_2(x,y)
\end{aligned}$$

En prenant $v = \phi$ dans les formulations variationnelles, alors sur chaque triangle K du maillage on doit avoir

$$\left\{ \begin{array}{rcl}
\nu \displaystyle\sum_{i,j} u_i \int_K \nabla\phi_i \nabla\phi_j - \sum_{i,j} p_i \int_K \lambda_i \nabla\phi_j &=& 0 \\
\displaystyle\sum_{i,j} u_i \int_K \lambda_j \nabla\phi_i - \epsilon \sum_{i,j} p_i \int_K \lambda_i\lambda_j &=& 0
\end{array} \right. \tag{3}$$

Pour calculer les intégrales on utilise une formule d'intégration numérique. Par exemple, on peut utiliser une formule exacte pour les polynômes de degré 2 :

$$\int_{\widehat{K}} f(x)dx \simeq \sum_{k=0}^{2} w_k f(x_k)$$

où w_k sont les poids d'intégrations, x_k les points d'intégration et $\widehat{K}$ est le triangle de référence.

Pour faire le lien entre la formulation variationnelle et le problème matriciel, on va introduire les formes linéaires discrètes suivantes :

- Soit $u \in X_h$, $w_h(u,\phi) = \displaystyle\sum_{i,j} u_i \int_K \nabla\phi_i\nabla\phi_j$,
- Soit $p \in Y_h$, $b_h(p,\phi) = \displaystyle\sum_{i,j} p_i \int_K \lambda_i \nabla\phi_j$
- Soit $p \in Y_h$, $m_{1,h}(p,\lambda) = \displaystyle\sum_{i,j} p_i \int_K \lambda_i\lambda_j$, le terme de masse $\mathbb{P}^1$ utilisé pour stabiliser le système.

1.3.1 Discrétisation temporelle de Navier-Stokes

En ce qui concerne la discrétisation de la partie instationnaire on utilise le fait que le terme $\frac{\partial u}{\partial t} + u \cdot \nabla u$ est approché par :

$$\frac{u^{n+1} - u^n(\chi^n)}{\Delta t} \simeq \frac{\partial u}{\partial t} + u \cdot \nabla u$$

où $\chi^n(x) \approx x - \Delta t u^n(x)$ est une fonction de transport qui donne la position de la particule x au temps t^n.

On obtient donc le schéma temporel d'ordre 1 suivant :

$$\left\{ \begin{array}{rcl} \frac{u^{n+1}}{\Delta t} - \nu \Delta u^{n+1} + \nabla p^{n+1} & = & \frac{u^n(\chi^n)}{\Delta t} \\ \nabla \cdot u^{n+1} & = & 0 \end{array} \right. \tag{4}$$

Résoudre Navier-Stokes revient donc à résoudre Stokes avec en un terme de masse $\mathbb{P}^2$ supplémentaire. Puis le second membre est mis à jour à chaque pas de temps.

On va maintenant définir les matrices associées à ce problème pour expliciter le système linéaire à résoudre.

1.4 Matrices associées au problème

Dans cette partie nous résoudrons uniquement le problème de Stokes : c'est à dire qu'il y aura pas d'évolution en temps et ni le terme $u \cdot \nabla u$. Pour être plus précis on résout :

$$\left\{ \begin{array}{rcl} -\nu \Delta u + \nabla p & = & 0 \\ \nabla \cdot u & = & 0 \end{array} \right. \tag{5}$$

1.4.1 Changement de variable et triangle de référence

Dans la suite, on va avoir à intégrer numériquement les fonctions de formes λ_i et ϕ_i. Pour ce faire on va d'abord expliciter les valeurs des intégrales sur le triangle de référence, *i.e* avec les fonctions de forme $\hat{\lambda}_i$ et $\hat{\phi}_i$. Ensuite on effectuera un changement de variable pour passer de K à $\widehat{K}$.

En effet, si on note $\boxed{F_K(\hat{x}) = B_K \hat{x} + b_K}$ la fonction affine telle que $F_K(\widehat{K}) = K$.

De cette notation découle directement que si on note $\hat{x} \in \widehat{K}$ et $\hat{v}$ une fonction définie sur $\widehat{K}$ alors

$$\boxed{v(x) = \hat{v} \circ F_K^{-1}(x)}.$$

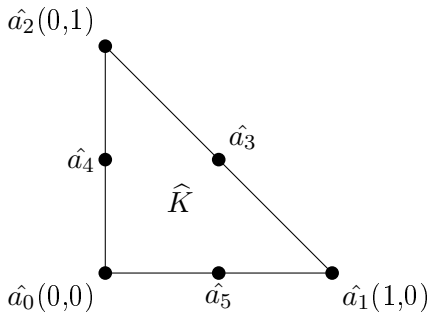

FIGURE 2 – Triangle de référence

De la même manière, puisque $DF_K = F_K$ [5] avec DF_K la différentielle de F_K on a aussi que

$$\boxed{\nabla v(x) = [B_K^{-1}]^\top \nabla \hat{v}(\hat{x})}.$$

Enfin la dernière formule importante est la suivante :

$$\boxed{B_K = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}}, \; b_K = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

où les sommets a_i du triangle K ont pour coordonnées (x_k, y_k) avec $k = 1, \cdots, 3$.

On en déduit donc :

$$|det(B_K)| = 2|K|$$

et

$$\boxed{B_K^{-1} = \frac{1}{2|K|} \begin{pmatrix} y_3 - y_1 & -(x_3 - x_1) \\ -(y_2 - y_1) & x_2 - x_1 \end{pmatrix}}.$$

Grâce à ces formules on va pouvoir calculer les matrices élémentaires sur K si on connaît les matrices élémentaires sur $\widehat{K}$ et l'aire de K sans avoir à utiliser d'intégration numérique.

1.4.2 Matrices élémentaires

Les matrices élémentaires du système de Stokes sont de 3 types : matrices de rigidité, matrice de masse et matrices de divergence.

Pour les matrices de rigidité on doit calculer sur chanque triangle K :

$$W_{i,j}^{k,l} = \int_K \frac{\partial \phi_i}{\partial x_k} \frac{\partial \phi_j}{\partial x_l}$$

5. Car F_K est affine

où ϕ_i est la fonction de base de X_h.

Après calculs, on obtient les trois matrices élémentaires de rigidité de taille 6×6 sur $\widehat{K}$:

$$\widehat{W}^{1,1} = \frac{1}{6}\begin{pmatrix} 3 & 1 & 0 & 0 & 0 & -4 \\ 1 & 3 & 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & -8 & 0 \\ 0 & 0 & 0 & -8 & 8 & 0 \\ -4 & -4 & 0 & 0 & 0 & 8 \end{pmatrix}, \widehat{W}^{2,2} = \frac{1}{6}\begin{pmatrix} 3 & 0 & 1 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 & -4 & 0 \\ 0 & 0 & 0 & 8 & 0 & -8 \\ -4 & 0 & -4 & 0 & 8 & 0 \\ 0 & 0 & 0 & -8 & 0 & 8 \end{pmatrix}$$

$$\widehat{W}^{1,2} = \frac{1}{6}\begin{pmatrix} 3 & 0 & 1 & 0 & -4 & 0 \\ 1 & 0 & -1 & 4 & 0 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & -4 & -4 \\ 0 & 0 & -4 & -4 & 4 & 4 \\ -4 & 0 & 0 & -4 & 4 & 4 \end{pmatrix}$$

La matrice de masse élémentaire est définie par :

$$M_{i,j} = \int_K \phi_i \phi_j dx$$

Dans le cas $\mathbb{P}^2$ la matrice M est de taille 6×6 et on a :

$$M = \frac{|K|}{180}\begin{pmatrix} 6 & -1 & -1 & -4 & 0 & 0 \\ -1 & 6 & -1 & 0 & -4 & 0 \\ -1 & -1 & 6 & 0 & 0 & -4 \\ -4 & 0 & 0 & 32 & 16 & 16 \\ 0 & -4 & 0 & 16 & 32 & 16 \\ 0 & 0 & -4 & 16 & 16 & 32 \end{pmatrix}$$

Enfin les matrices élémentaires de divergence sont définies sur chaque triangle K par :

$$B_{i,j}^{(l)} = \int_K \frac{\partial \phi_i}{\partial x_l} \omega_j dx \text{ avec } l \in \{1, 2\}$$

où ω_j est la fonction de base de Y_h.

Dans le cas $\mathbb{P}^2 \setminus \mathbb{P}^1$ on a 2 matrices élémentaires de taille 6×3 à calculer sur le triangle de référence $\widehat{K}$:

$$\widehat{B}^{(1)} = \frac{1}{6}\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 2 \\ -1 & -1 & -2 \\ 1 & -1 & 0 \end{pmatrix}, \widehat{B}^{(2)} = \frac{1}{6}\begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & -1 \\ -1 & -2 & -1 \end{pmatrix}$$

1.4.3 Matrice globale

On met le système (3) sous la forme $A.X = b$ avec A une matrice inversible, $X = (u_1, u_2, p)$ notre inconnue et b un vecteur qui contient les conditions de bord.

Pour construire la matrice globale A on procède en deux étapes : tout d'abord on construit les matrices élémentaires sur chaque triangle du maillage et ensuite on assemble toutes les matrices élémentaires dans la matrice globale.

Soit K un triangle du maillage, la matrice globale A s'écrit sur le triangle K :

$$A = \begin{pmatrix} W & 0 & -B^{(1)} \\ 0 & W & -B^{(2)} \\ -(B^{(1)})^\top & -(B^{(2)})^\top & -\epsilon M \end{pmatrix}$$

avec $W = W^{1,1} + W^{2,2}$.

2 Implémentation informatique

2.1 Architecture générale

Pour simplifier le développement, on a utilisé un compte *GoogleCode* qui permet de rassembler un gestionnaire de code source et un wiki au même endroit. On a choisi d'utiliser *Mercurial* comme gestionnaire de code source. Pour récupérer le code source associé au projet il suffit donc de taper dans le terminal : `hg clone https://oli.tissot@code.google.com/p/projetnm406-en-ot/`.

Afin de respecter le paradigme objet du C++, nous avons découpé notre programme en différentes classes. Nous allons ici détailler les plus importantes ainsi que les différentes dépendances à des bibliothèques externes.

Pour représenter notre maillage en deux dimensions on utilise la classe *Maillage*. On a commencé par définir des classes pour représenter les sommets, segments et triangles. Ces petites classes contiennent aussi quelques méthodes utiles pour la suite comme le calcul de l'aire ou de la longueur. De base, un maillage est un ensemble de trois tableaux : un pour les sommets, un pour les segments du bord et un pour les triangles. Néanmoins, vu qu'on utilise des éléments $\mathbb{P}^2$ on aura aussi besoin des segments internes pour pouvoir numéroter nos degrés de liberté [6]. On a choisi de stocker ces segments sous forme de map [7] en dehors de la classe *Maillage* pour ne pas l'alourdir car elle est très souvent passée en paramètre. Nous aurons aussi besoin [8] de connaître les triangles adjacents : c'est un simple tableau que nous avons donc ajouté directement au sein de la classe. Nous ne détaillerons pas les méthodes et renverrons directement le lecteur au code source qui est commenté.

Remarque. *Notre classe ne lit qu'un seul format de maillage mais elle renvoie une erreur si le format n'est pas bon.*

Pour représenter les matrices creuses associées au problème, on a défini une classe spéciale *MapMatrix*. Comme son nom l'indique, cette classe est basée sur l'utilisation d'une map pour représenter la matrice. Cette façon de faire n'est pas optimale [9] mais elle est beaucoup plus simple à coder, et sur de petits problèmes comme le nôtre, la différence est minime. Afin de rendre l'utilisation naturelle on a surchargé l'opérateur $=$. Il a aussi fallu coder une méthode qui convertit notre *MapMatrix* au format *CSR* afin de pouvoir utiliser notre solveur de systèmes linéaires.

6. cf. 2.2.2
7. idem
8. cf. 2.3.1
9. On a une complexité en $O(ln(n))$ en écriture car la mémoire est ré-allouée dynamiquement (ou presque) contre $O(1)$ si la mémoire est déjà allouée. C'est-à-dire qu'il faudra allouer la mémoire avant de commencer l'assemblage ce qui est assez complexe.

Pour résoudre notre système linéaire on utilise *UMFPACK*. C'est un solveur direct, basé sur une décomposition LU multifrontale, pour matrices creuses [10]. L'intérêt d'utiliser un solveur direct est double :

- On ne commet pas d'erreur lors de cette étape du calcul.
- On n'a pas besoin d'hypothèse sur la structure de la matrice.

En contrepartie, le temps de calcul est plus lent. Ici, puisque notre problème est en 2D, le temps de calcul avec *UMFPACK* reste très correct. Nous vous renvoyons au *Makefile* du projet pour connaître précisément les dépendances de *UMFPACK*, mais il faut installer le package *SuiteSparse* [11] ainsi qu'une version de *BLAS* [12].

2.2 Assemblage des matrices globales

Pour assembler la matrice globale A on utilise une routine du type :

Algorithme 1: Assemblage de la matrice globale

```
for K ∈ T_h do
    for i = 0 to 15 do
        I = numglob(K, i);
        for j = 0 to 15 do
            J = numglob(K, j);
            A(I, J) += Aelem(i, j);
        end
    end
end
```

Ensuite, pour rajouter les conditions de Dirichlet on utilise une méthode de pénalisation exacte [13] :

Où tgv est une "Très Grande Valeur" [14] et les e_i correspondent aux degrés de liberté $\mathbb{P}^2$ [15] sur le segment e (Figure 3).

Regardons maintenant comment est construite la matrice élémentaire `Aelem` et comment fonctionne la méthode `numglob`.

10. C'est le solveur utilisé par Matlab pour les matrices creuses.
11. Du même auteur que *UMFPACK* : Tim Davies (University of Florida).
12. Si possible optimisée pour le processeur de la machine où doit s'exécuter le programme.
13. Ou encore méthode TGV.
14. 10^{30} dans notre programme
15. Car on ne met des conditions de Dirichlet que pour la vitesse.

Algorithme 2: Méthode de pénalisation exacte

```
for e ∈ Γ do
    for i = 0 to 1 do
        A(e_i, e_i) = tgv ;
        Fh(e_i) *= tgv ;
    end
end
```

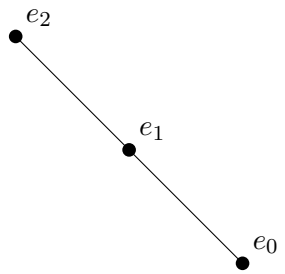

FIGURE 3 – Segment e

2.2.1 Assemblage des matrices élémentaires

Il existe 2 méthodes pour assembler les matrices élémentaires :
- On calcule directement les intégrales en utilisant une formule de quadrature.
- On calcule les intégrales sur $\widehat{K}$ puis on utilise un changement de variable pour se ramener à K.

On a choisi d'utiliser la deuxième méthode, comme présentée dans [JFS]. Cette méthode présente l'avantage d'être moins coûteuse en terme de calcul. En effet, il faudrait utiliser des formules de quadratures exactes à l'ordre 4 pour être assuré de calculer exactement les intégrales, sans quoi le schéma ne serait plus forcément stable et l'étude de l'erreur deviendrait beaucoup plus compliquée.

Donc, dans un premier temps [16], on calcule les contributions élémentaires pour $\widehat{K}$: c'est ce que fait la méthode `MatricesElementairesKchap`. Ensuite, on calcule les contributions élémentaires pour K à partir des contributions élémentaires pour $\widehat{K}$: c'est ce que fait la méthode `MatricesElementaires`.

Remarque. *Pour faciliter l'écriture de ces 2 fonctions on a utilisé les classes de tableaux définies dans le package* RNM [17].

16. Avant la boucle sur les triangles.
17. Disponible sur la page personnelle de F. Hecht.

2.2.2 Numérotation des degrés de liberté

Pour écrire `numglob`, on se pose la question suivante : comment numéroter les degrés de liberté du problème ?

En effet, on rappelle que les degrés de liberté $\mathbb{P}^2$ sont à la fois les sommets du triangle mais aussi les milieux de ses segments (Figure 4). Mais nous

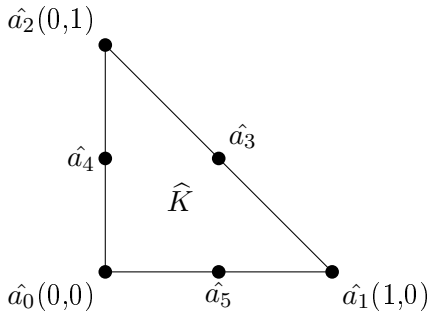

FIGURE 4 – Triangle de référence

ne disposons que des numéros des sommets, des segments du bord et des triangles du maillage. A priori, il ne semble donc pas simple de numéroter les milieux des segments de chaque triangle de manière unique.

Une solution à ce problème est de construire les segments internes au maillage. En effet, on peut alors identifier les segments avec leur milieu et donc leur associer le même numéro global. Cette construction n'est pas un problème simple car il faut gérer les doublons : $[AB] = [BA]$. Une solution est d'utiliser le conteneur `map` de la *STL* : notre tableau de segments sera alors une `map< pair<int,int>, int >`. L'algorithme est alors très simple à écrire et il existe déjà une méthode `find` qui permet de retrouver le numéro associé à un segment. Néanmoins, une `map` est assez lourde en mémoire [18] donc nous avons décidé de la stocker en dehors de notre maillage. La méthode qui construit les segments internes a donc le prototype suivant : `void Maillage::buildEdge(map< pair<int,int>, int >&) const`.

Une fois ce problème résolu il faut décider d'une façon de "ranger" les différentes inconnnues du problème dans notre vecteur solution. Nous avons choisi la convention suivante :

$$S = \begin{pmatrix} U_{10} \\ U_{11} \\ U_{20} \\ U_{21} \\ P \end{pmatrix}$$

18. Plus qu'un simple tableau.

avec,

- S le vecteur solution du système linéaire.
- $\overrightarrow{U} = (U_1, U_2)$ et $U_1, U_2 \in \mathbb{R}$.
- U_{10} le vecteur composé des valeurs de U_1 pour les points du maillage.
- U_{11} le vecteur composé des valeurs de U_1 pour les milieux des côtés des triangles du maillage.
- U_{20} le vecteur composé des valeurs de U_2 pour les points du maillage.
- U_{21} le vecteur composé des valeurs de U_2 pour les milieux des côtés des triangles du maillage.
- P le vecteur composé des valeurs de P pour les points du maillage.

Remarque. *Les méthodes qui s'occupent de ce rangement sont* `int Maillage::numedge(int, map< pair<int,int>, int >&, int) const` *et* `int Maillage::numglob(int, map< pair<int,int>, int >&, int) const`. *Elles prennent en paramètre notre tableau de segment (qui doit déjà être construit bien entendu).*

2.3 Assemblage du second membre associé à Navier-Stokes

Pour calculer ce second membre, et si on reprend les notations définies dans la partie précédente, on va avoir besoin de calculer des intégrales de la forme :

$$\int_K \phi_i(x)\ u^n \circ \chi^n(x) dx$$

On commence par essayer de calculer $u^n \circ \chi^n(x)$. Grâce à un développement limité au premier ordre [19] et grâce à la définition de χ^n on obtient :

$$\chi^n(x) \approx x - \Delta t\, u^n(x)$$

La difficulté vient du fait qu'on ne connait la valeur de u^n que sur les degrés de liberté $\mathbb{P}^2$: les points du maillage, et les milieux des segments internes. Or $\chi^n(x)$ n'a aucune raison, a priori, d'appartenir à un point du maillage ou un milieu d'une arrête interne. Pire, il se pourrait très bien que $\chi^n(x)$ n'appartienne même pas au maillage !

2.3.1 Promenade sur le maillage

Si on suppose pour l'instant que $\chi^n(x) \in \mathcal{T}_h$ [20], le problème qu'on se pose peut être reformulé de la façon suivante : étant donné un point à l'intérieur, trouver le triangle qui contient ce point. En effet, si on connaissait ce triangle,

19. Le même raisonnement est aussi possible en faisant un DL à l'ordre 2, cf. [GF] pour plus de détails mais la suite de la mise en pratique est essentiellement la même.

20. Ce qui est vrai partout sauf là où du fluide entre, c'est-à-dire sur Γ_6.

on pourrait utiliser la formule suivante : $u^n \circ \chi^n(x) \approx \sum_{i=0}^{5} u^n(a_i)\,\phi_i(x)$ avec a_i les degrés de liberté $\mathbb{P}^2$ sur le triangle en question.

Ce problème peut sembler très simple en apparence mais sa résolution n'est pas du tout triviale [21]. En effet, une méthode naïve comme tester tous les triangles n'a aucune chance de fonctionner dès que le maillage devient précis.

Une façon de faire très répandue est d'utiliser la méthode dite de la "Promenade" [22] : on se promène de triangle en triangle sur le maillage en diminuant la distance entre le triangle et le point recherché (Figure 5).

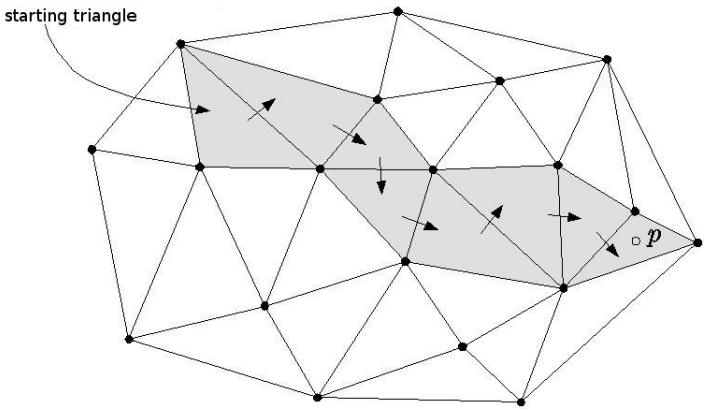

FIGURE 5 – Promenade sur un maillage

Nous avons donc ajouté une méthode `Walk` à l'intérieur du fichier où est implémentée notre classe *Maillage*. Pour fonctionner, `Walk` a besoin de connaître les triangles adjacents à un triangle donné, c'est pourquoi nous avons aussi rajouté un tableau à notre classe *Maillage* `TheAdjacencesLink` ainsi qu'une méthode qui l'initialise `BuildAdj` [23]. La méthode `Walk` nous renvoie les coordonnées barycentriques du point recherché à l'intérieur du triangle recherché, on peut donc facilement récupérer une valeur de $u^n \circ \chi^n(x)$ en sommant les valeurs de $u^n(a_i)$ pondérées par les coordonnées barycentriques qu'on vient de récupérer [24].

Remarque. *La méthode implémentée prend en compte le cas où le point sortirait du domaine et dans ce cas-là renvoie le projeté orthogonal sur la frontière.*

21. Et elle a donné lieu à un certain nombre d'articles de recherche...
22. Walk en anglais.
23. Ceci nous a obligé à utiliser d'autres dépendances afin que cette méthode fonctionne notamment le fichier *Hastable.hpp*.
24. On peut même utiliser les coordonnées $\mathbb{P}^2$ grâce aux formules liant les ϕ_i avec les λ_i.

2.3.2 Formule de quadrature

Si nous revenons au problème de départ, il nous faut calculer les intégrales de la forme :

$$\int_K \phi_i(x)\ u^n \circ \chi^n(x) dx$$

Si on veut que ce calcul soit exact, et il le faut car on a calculé exactement les intégrales des matrices locales, on doit utiliser une formule de quadrature exacte au moins à l'ordre 4. Sinon des compensations qui devraient avoir lieu ne se feront pas et une erreur va s'accumuler au cours du temps ce qui fera exploser le schéma. Généralement ce sont des formules très compliquées où on a de fortes chances de faire une erreur de copie. Donc on a utilisé des classes définies dans *FreeFem++* qui définissent des formules de quadrature d'ordre élevée qui ont été testées. La formule qu'on utilise est exacte à l'ordre 5 ce qui assure la stabilité du schéma.

2.4 Visualisation des résultats

Pour visualiser les résultats on utilise 2 routines *OpenGL* différentes. Cette interface de programmation est un standard extrêmement utilisé dans le milieu scientifique car elle assure une compatibilité quelque soit le type de terminal. Ces exécutables sont indépendants du programme principal et doivent être compilés à part. De même que pour *UMFPACK* nous vous renvoyons au *Makefile* pour connaître les dépendances d'*OpenGL*.

Le premier est utilisé pour visualiser une solution P_1 : on l'appelle pour visualiser la pression. Son utilisation est très simple, il prend en paramètre le fichier de maillage et un fichier qui donne la valeur de la fonction en chacun des points du maillage [25].

Le second est utilisé pour visualiser une solution $\mathbb{P}^2$: on l'appelle pour visualiser U_1 ou U_2 [26]. En fait, il s'agit d'une version modifiée de la première routine : au lieu de construire un triangle par triangle du maillage on en construit 4 [27].

Remarque. *On utilise la même méthode qu'expliquée précédemment pour numéroter les degrés de liberté donc si les degrés de liberté sont numérotés différemment la routine ne fonctionne plus. En particulier, si vous sauvegardez une solution $\mathbb{P}^2$ depuis Freefem++ l'affichage ne sera pas bon.*

25. L'ordre des points doit être le même bien entendu.

26. Il n'y a pas d'affichage pour le vecteur vitesse mais Freefem++ peut faire ça si la solution est stockée sous le bon format.

27. Nous avons aussi changé la procédure qui calcule les bornes de la fonction : il faut boucler sur plus de valeurs qu'il n'y a de points sur le maillage.

3 Résultats numériques et comparaisons avec FreeFem++

Pour valider notre code C++ et notre méthode numérique nous avons comparé nos résultats avec ceux du logiciel FreeFem très utilisé dans le monde de la recherche académique et industriel pour résoudre des EDP. Commençons avec le problème de Stokes :

3.1 Problème de Stokes

Dans toute la suite, on se placera avec une viscosité ν égale à 0.01 $m^2.s^{-1}$. On va comparer la pression, la vitesse horizontale u_1 et la vitesse verticale u_2 que nous avons obtenues avec notre code C++ et celles obtenues avec FreeFem++ sur le même maillage.

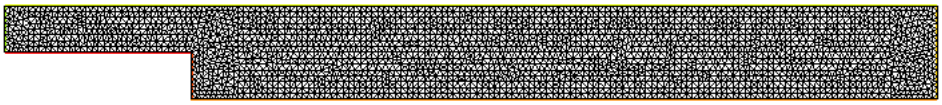

FIGURE 6 – Maillage utilisé lors des tests

3.1.1 Résultats obtenus

Voilà les résultats que l'on obtient :

FIGURE 7 – P calculée

FIGURE 8 – P *FreeFem++*

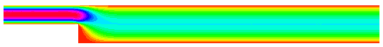

FIGURE 9 – U1 calculée

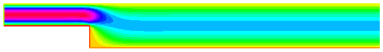

FIGURE 10 – U1 *FreeFem++*

FIGURE 11 – U2 calculée

FIGURE 12 – U2 *FreeFem++*

3.1.2 Discussion

On peut voir qu'on obtient quasiment le même motif pour la pression mais *FreeFem++* ne normalise pas à la même valeur. Pour u_2 les résultats

sont aussi très proches, dans les 2 cas on voit bien un tourbillon apparaître au milieu du graphique. Enfin pour u_1, là encore *FreeFem++* confirme que ce nous obtenons avec notre programme a l'air correct. Néamoins, avec FreeFem on observe une légère trainée bleu au milieu du maillage qui n'apparaît pas aussi clairement sur l'image obtenue avec notre code. Il faut tout de même garder en tête que l'échelle de couleur utilisée par *FreeFem++* n'est pas la même que celle utilisée par notre routine d'affichage.

Afin de vérifier que nos résultats étaient bons on a donc calculé la norme L^2 du résidu entre la pression calculée par *FreeFem++* et celle calculée par notre programme. On fait ce calcul pour une batterie de maillages de plus en plus fins pour voir si les deux solutions convergent bien vers une même limite. Voilà les résultats obtenus :

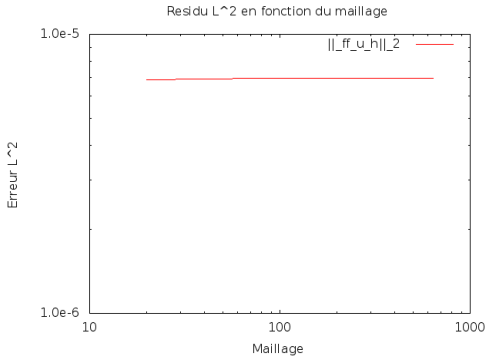

FIGURE 13 – $||p_{ff++} - p_h||_2$ en fonction du maillage

On remarque que l'erreur est faible : de l'ordre de 10^{-6}. Elle est constante selon le maillage ce qui semblerait indiquer que *FreeFem++* et notre programme calculent en fait la même approximation : l'erreur étant dûe aux erreurs d'arrondis. On peut donc raisonablement considérer que notre pression est correcte.

Remarque. *Nous n'avons pas réussi à faire le même calcul avec la vitesse* (u_1, u_2) *car* FreeFem++ *et notre programme ne rangent pas de la même façon les composantes de la vitesse.*

3.2 Problème de Navier-Stokes

Après avoir validé nos développements sur la solution du problème de Stokes, nous avons testé notre programme sur la résolution des équations de Navier-Stokes.

Dans les tests, on s'est placé dans le cas d'une viscosité petite, c'est-à-dire que ce sont les termes non linéaires qui dominent et qu'on devrait observer

l'apparition de tourbillons. Ces tourbillons seront d'autant plus visibles que la viscosité sera petite. On a choisi de simuler avec $\Delta t = 0.1s$ jusqu'à $T_{final} = 10s$ sur le même maillage que pour les tests sur le problème de Stokes.

3.2.1 Résultats obtenus

Dans le cas où $\nu = \frac{1}{100} m^2.s^{-1}$, voici les images obtenues au temps final.

FIGURE 14 – P calculée

FIGURE 15 – P *FreeFem++*

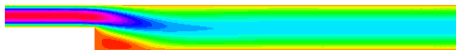

FIGURE 16 – U1 calculée

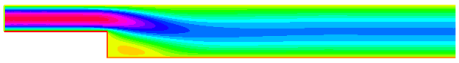

FIGURE 17 – U1 *FreeFem++*

FIGURE 18 – U2 caculée

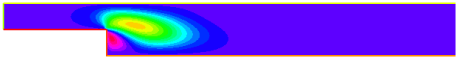

FIGURE 19 – U2 *FreeFem++*

Dans le cas où $\nu = \frac{1}{400} m^2.s^{-1}$, voici les images obtenues au temps final :

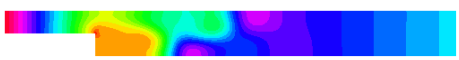

FIGURE 20 – P calculée

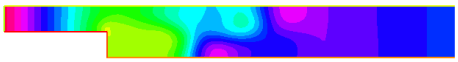

FIGURE 21 – P *FreeFem++*

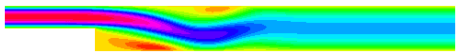

FIGURE 22 – U1 calculée

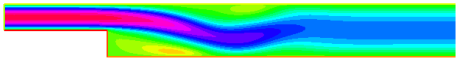

FIGURE 23 – U1 *Freefem++*

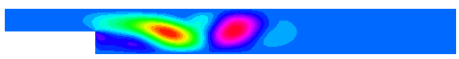

FIGURE 24 – U2 caculée

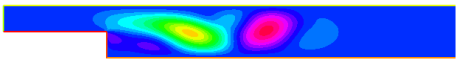

FIGURE 25 – U2 *Freefem++*

3.2.2 Discussion

Les deux solutions semblent très proches que ce soit dans le cas $\nu = \frac{1}{100}$ ou $\nu = \frac{1}{400}$. En particulier, le comportement de notre solution paraît normal : lorsque la viscosité diminue les termes non linéaires deviennent de plus en plus dominants et les tourbillons sont de plus en plus visibles.

Néanmoins, ces images peuvent être trompeuses car les échelles de couleur ne sont pas exactement les mêmes dans notre programme et dans *FreeFem++*. En complément, on a donc calculé la norme L^2 du résidu de pression à $t = T_{final}$ et $t = \frac{T_{final}}{2}$ pour les différents maillages. On obtient les courbes suivantes :

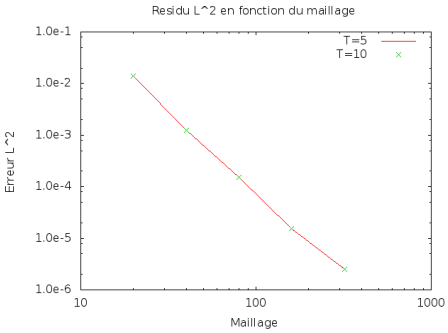

FIGURE 26 – $\nu = \frac{1}{100}$

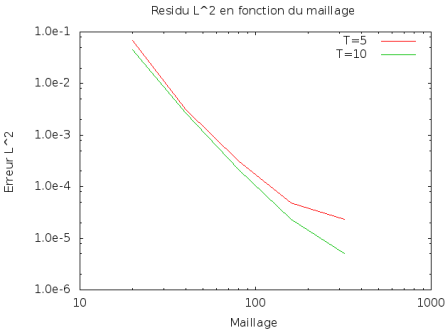

FIGURE 27 – $\nu = \frac{1}{400}$

Là encore, on peut raisonablement conclure que notre schéma numérique est correctement implémenté. Lorsque le maillage est suffisament fin on retrouve le même ordre de grandeur que celle mesurée pour le problème de Stokes.

Remarque. *Comme attendu, notre programme fonctionne plus vite que* FreeFem++ [28]*. En effet, lors de l'assemblage des matrices nous n'utilisons pas d'intégration numérique. Par ailleurs, comme notre code est beaucoup plus spécialisé nous effectuons beaucoup moins de tests que* FreeFem++*.*

Remarque. *Des tests numériques lorsque ν devient très petite* [29] *montrent que notre schéma numérique devient instable. Ceci n'est pas très étonnant car dans ce cas l'écoulement passe en régime turbulent et il faut utiliser des modèles spécifiques* [30] *si on veut le simuler correctement.*

28. Environ deux fois plus vite.
29. De l'ordre de 10^{-4}
30. Comme par exemple les modèles LES ou RANS.

Conclusion

Dans un premier temps, on a explicité la formulation faible du problème de départ et les espaces associés. En choisissant les espaces discrets adéquats[31], on a pu déduire des expressions pour les matrices locales et le second membre. Afin de discrétiser l'équation en temps, on a utilisé la méthode des caractéristiques.

Ensuite, on a présenté la structure de notre programme. On a donc expliqué comment on assemblait les matrices et le second membre. Puis, on a vu comment on pouvait calculer la caractéristique. Pour visualiser les résultats, on a utilisé *OpenGL*.

Enfin, on a effectué quelques tests pour valider notre développement. Ils ont été séparés en deux parties : d'abord sur le problème de Stokes et ensuite sur Navier-Stokes. Pour chaque partie, on a comparé nos résultats avec ceux obtenus avec Freefem++.

Pour aller plus loin on pourrait envisager de comparer l'efficacité de différentes méthodes éléments finis.

31. P_2 pour la vitesse, P_1 pour la pression.

Références

[JFS] J.-F. Scheid, *Analyse numérique des équations de Navier-Stokes*, disponible sur sa page personnelle dans la rubrique enseignement.

[GF] G. Fourestey, *Une méthode des caractéristiques d'ordre deux sur maillages mobiles pour la résolution des équations de Navier-Stokes incompressibles par éléments finis*, Rapport de recherche INRIA n°4448, Avril 2002.

[PCEL] P.Ciarlet, Eric Lunéville, *La méthode des éléments finis. De la théorie à la pratique Tome1 Concepts généraux*, Les Presses de l'ENSTA, 2009.

[AEG] A.Ern, J.-L. Guermond, *Éléments finis : théorie, applications, mise en oeuvre*, Springer, Vol.36, 11 janvier 2002.

[BMR] C.Bernardi, Y.Maday, F.Rapetti, *Discrétisations variationelles de problèmes aux limites elliptiques*, Springer, Vol.45, 2004