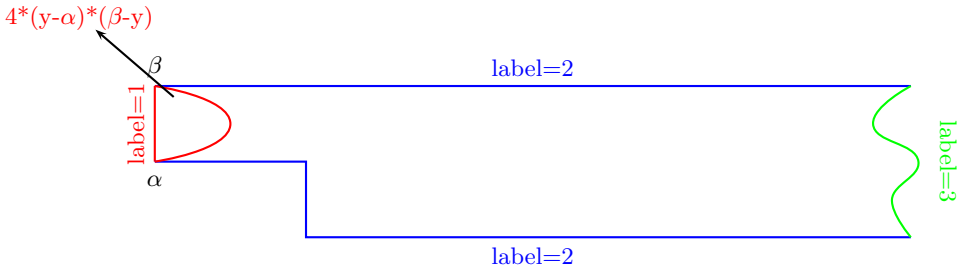
Université de Pierre et Marie CURIE (PARIS VI)
4 Place Jussieu 75005 Paris

ÉCOULEMENT LAMINAIRE DANS UNE CONDUITE

ÉQUATION DE NAVIER-STOKES STATIONNAIRE

Enseigné par M.Frédéric Hecht



NGUYEN Chi Thanh

2008-2009

- Existence et unicité
- Méthode de continuation
- Programmation C++
- Qt Programmation GUI

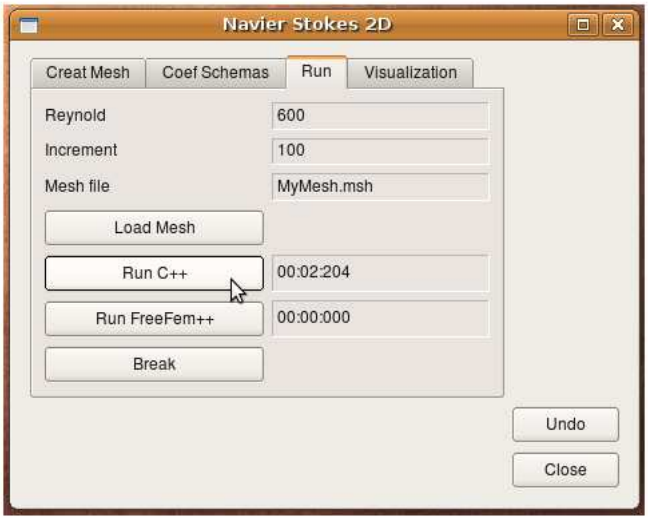


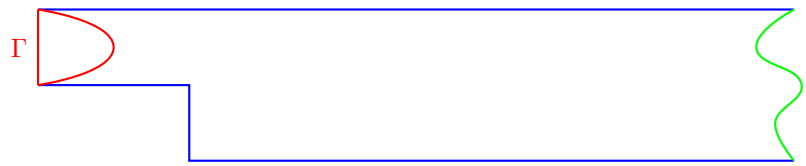
Table des matières

Introduction

Dans le cadre des études de Master 2 de mathématique de la modélisation (ANEDP) à l'université Pierre et Marie Curie, les étudiants suivant le module NM406 (Résolution des EDP par la Méthode des Éléménts Finis) enseigné par M.Frédéric Hecht doivent réaliser un projet de calcul scienfique utilisant la FEM développé en C++.Ce projet est basé sur la bibliothèque servant à développer le logiciel FreeFem++ par le professeur.

Phénomène physique : Le projet consiste à simuler l'écoulement laminaire dans une conduite en résolvant l'équation de Navier-Stokes stationnaire. Un écoulement laminaire est régulier (il ne présente pas trop de variations spatiales ou temporelles), bien souvent stationnaire. La viscosité stabilise et régularise les écoulements de façon générale. Un fluide présentant une viscosité importante s'écoulera de façon laminaire. Un écoulement est caractérisé par son nombre de Reynolds, qui permet de se faire une idée de sa stabilité : quand ce nombre est petit, l'écoulement est laminaire, quand il est grand, l'écoulement est en générale instable et turbulent. La transition entre les écoulements stables et instable voire turbulents est un sujet d'étude important. Dans une conduite, on considère souvent que la transition peut se produire entre 2000 et 3000.

Problème mathématique : Il s'agit en fait d'une solution stable des équations de Navier-Stokes, au sens où si on modifie l'écoulement, il retourne vers la solution laminaire. Le domaine étudié ici est l'un des problème classique : la marche descendante.



La conduite horizontale, la rentrée du fluid à droite avec le profit parabolique représantant la condition Poiseuille donnant le problème mathématique suivant :

$$(\wp) \quad \begin{cases} (\mathbf{u}.\nabla)\mathbf{u} - \nu \triangle \mathbf{u} + \nabla p &= 0 \\ \nabla \mathbf{u} &= 0 \\ \mathbf{u}_\Gamma &= g \end{cases}$$

Ayanat pour but de résoudre ce problème,le projet se compose donc l'étude mathématique (existence et l'unicité de solution, l'implémentation numérique), et l'étude informatique basé sur la programmation C++ pour résoudre ce problème.

Chapitre 1

Études mathématiques

Cause de la limite du temps l'étude du cadre fonctionnelle ne sera pas toute détaillée dans ce projet. Ce rapport ne présente qu'une partie brève de l'étude fonctionnelle de l'équation Navier-Stokes dans ce cas concret pour arriver à l'implémentation informatique, qui est l'essentiel de l'exercice. Pour plus de détails sur la méthode de continuation, il est de préférence de consulter [6, Girault-Raviart p.297-365]

1.1 Cadre fonctionnelle

Ce paragraphe montre brièvement la méthode de continuation servant d'une part à prouver l'existence et l'unicité de la solution, d'autre amener à l'implémentation du schéma numérique de type Newton.

Méthode de continuation : Cette méthode, par la construction d'une fonctionnelle F d'un certain d'espace de départ et d'espace d'arrivé :

$$\begin{array}{rl} \Lambda \times X & \longmapsto \chi \\ (\lambda, u) & \longmapsto F(\lambda, u) \end{array}$$

on peut prouver que la résolution du problème :

$$F(\lambda, u) = 0 \tag{1.1}$$

est équivalent au problème :

$$(\wp) \quad \left\{ \begin{array}{rl} (\mathbf{u}.\nabla)\mathbf{u} - \nu \triangle \mathbf{u} + \nabla p & = 0 \\ \nabla \mathbf{u} & = 0 \\ \mathbf{u}_\Gamma & = g \end{array} \right. \tag{1.2}$$

Pour comprendre ce mécanisme, on a besoin de quelques définitions des espaces fonctionnelles et ses opérateurs :

Espaces fonctionnels :

$$\begin{array}{ll} I & \subset \Re \\ X & = H^1(\Omega)^N \times L^2_0(\Omega) \\ Y & = H^{-1}(\Omega)^N \times \{g \in H^{\frac{1}{2}}(\Gamma)^N, \int_{\partial\Omega} g.n d\sigma = 0\} \end{array}$$

Opérateur de Stokes : Soit le problème de Stokes :

$$\left\{\begin{array}{rl} (-\nu \triangle \mathbf{u} + \nabla p & = f \\ \nabla \mathbf{u} & = 0 \\ \mathbf{u}_\Gamma & = g \end{array}\right.$$

On peut démontrer l'existence et l'unicité de ce problème dans X. De sorte que l'on peut définir un isomorphisme :

$$T : \begin{array}{rl} Y & \longmapsto X \\ \left[\begin{array}{c} f \\ g \end{array}\right] & \longmapsto \left[\begin{array}{c} u \\ p \end{array}\right] \end{array}$$

Remarque : T est linéaire, continue donc différentialble.

Opérateur auxiliaire : On définit encore un autre opérateur :

$$G : \begin{array}{rl} I \times X & \longmapsto Y \\ (\lambda, \left[\begin{array}{c} v \\ q \end{array}\right]) & \longmapsto - \left[\begin{array}{c} \lambda(f - (v.\nabla)v) \\ g \end{array}\right] \end{array}$$

On peut remarquer encore que l'application partielle $G(\lambda, .)$ est linéaire continue, donc il est facile à calculer sa différentielle.

Opérateur de continuation : On s'amène finalement à définir l'opérateur $F = Id + T \circ G$, c'est à dire :

$$F : \begin{array}{rl} I \times X & \longmapsto X \\ (\lambda, \left[\begin{array}{c} u \\ p \end{array}\right]) & \longmapsto F(\lambda, \left[\begin{array}{c} u \\ p \end{array}\right]) = Id \left[\begin{array}{c} u \\ p \end{array}\right] + T \circ G(\lambda, \left[\begin{array}{c} u \\ p \end{array}\right]) \end{array}$$

De sorte que comme G, $F(\lambda, .)$ est une application linéaire continue de X à Y, que l'on peut calculer facilement la différentielle de cettet application partielle :

$$F(\lambda, \bullet) : \begin{array}{rl} X & \longmapsto X \\ \left[\begin{array}{c} u \\ p \end{array}\right] & \longmapsto Id \left[\begin{array}{c} u \\ p \end{array}\right] + T \circ G(\lambda, \left[\begin{array}{c} u \\ p \end{array}\right]) \end{array}$$

Partant de ces principales constructions d'espaces et opérateurs, en utilisant ensuite la théorie des fonctions implicites, on se ramène à prouver l'équivalent des problème (1.1) et (1.2). L'étude fonctionnelle prouve que $\left[\begin{array}{c} u \\ p \end{array}\right]$ de X est solution du problème (1.2) si et seulement si $\left[\begin{array}{c} u \\ \frac{p}{\nu} \end{array}\right]$ de X est solution du problème $F(\frac{1}{\nu}, \bullet) = 0$.

1.2 Cadre numérique

Issu de la conclution de la dernière section, on peut utiliser le schéma de Newton pour résoudre le problème $F(\frac{1}{\nu}, \bullet) = 0$. Soit $\left[\begin{array}{c} u \\ \frac{p}{\nu} \end{array}\right]$ solution, par le théorème d'inversion local, on a :

$$F(\lambda, \left[\begin{array}{c} u \\ \frac{p}{\nu} \end{array}\right]) = 0 \Longleftrightarrow \partial_{up} F_{(u,p)}(\frac{1}{\nu}, \bullet) \text{est un isomorphisme.}$$

Pour la résolution numérique, on peut utiliser le schéma de Newton :

$$\begin{bmatrix} u^{n+1} \\ \frac{p^{n+1}}{\nu} \end{bmatrix} = \begin{bmatrix} u^n \\ \frac{p^n}{\nu} \end{bmatrix} - \partial_{up} F^{-1}_{(u^n, \frac{p^n}{\nu})}(\frac{1}{\nu}, \bullet) \circ F(\frac{1}{\nu}, \begin{bmatrix} u^n \\ \frac{p^n}{\nu} \end{bmatrix})$$

En calculant la différentielle de F par les différentielles de G et de T, on arrive au :

Schéma de Newton :

$$(N) \quad \begin{cases} -\triangle \mathbf{u}^{n+1} + \frac{1}{\nu}[(\mathbf{u}^n.\nabla)\mathbf{u}^{n+1} + (\mathbf{u}^{n+1}.\nabla)\mathbf{u}^n] + \nabla p^{n+1} &= \frac{1}{\nu}(\mathbf{u}^n.\nabla)\mathbf{u}^n \\ \mathrm{div}\mathbf{u}^{n+1} &= 0 \\ \mathbf{u}^{n+1}_{|\Gamma_D} &= g \end{cases}$$

Partant de ce schémas, en affectant que $\mathbf{uold} = \mathbf{u}^n$ est la solution calculée de l'itération précédante, $\mathbf{u} = \mathbf{u}^{n+1}$ est la solution à calculer, on récrit pour chaque itération, le problème devient :

$$(N) \quad \begin{cases} -\triangle \mathbf{u} + \frac{1}{\nu}[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}] + \nabla p &= \frac{1}{\nu}(\mathbf{uold}.\nabla)\mathbf{uold} \\ \mathrm{div}\mathbf{u} &= 0 \\ \mathbf{u}_{|\Gamma_D} &= g \end{cases}$$

Soit $\begin{bmatrix} \mathbf{v} \\ q \end{bmatrix}$ dans le même espace, en multipliant et intégrant, le problème équivalent à la :

Formulation Variationnelle :

$$\begin{cases} -\int_\Omega \triangle\mathbf{u}.\mathbf{v} + \frac{1}{\nu}\int_\Omega[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}].\mathbf{v} & \\ \qquad + \int_\Omega \nabla p.\mathbf{v} - \int_\Omega q\mathrm{div}\mathbf{u} &= \frac{1}{\nu}\int_\Omega(\mathbf{uold}.\nabla)\mathbf{uold}.\mathbf{v} \\ \mathbf{u}_{|\Gamma_D} &= g \end{cases}$$

En développant terme à terme de la première égalité, on a :

$$\begin{aligned} -\int_\Omega \triangle\mathbf{u}.\mathbf{v} &= \int_\Omega \nabla\mathbf{u}.\nabla\mathbf{v} - \int_{\partial\Omega} \triangle\mathbf{u}.(\mathbf{v}.\mathbf{n})\mathrm{d}\sigma \\ +\frac{1}{\nu}\int_\Omega[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}].\mathbf{v} &= +\frac{1}{\nu}\int_\Omega[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}].\mathbf{v} \\ +\int_\Omega \nabla p.\mathbf{v} &= -\int_\Omega p\mathrm{div}\mathbf{v} + \int_{\partial\Omega} p.(\mathbf{v}.\mathbf{n})\mathrm{d}\sigma \\ -\int_\Omega q\mathrm{div}\mathbf{u} &= -\int_\Omega q\mathrm{div}\mathbf{u} \\ \frac{1}{\nu}\int_\Omega(\mathbf{uold}.\nabla)\mathbf{uold}.\mathbf{v} &= \frac{1}{\nu}\int_\Omega(\mathbf{uold}.\nabla)\mathbf{uold}.\mathbf{v} \end{aligned}$$

Avec la condition de sortie $-\int_{\partial\Omega} \triangle\mathbf{u}.(\mathbf{v}.\mathbf{n})\mathrm{d}\sigma = +\int_{\partial\Omega} p.(\mathbf{v}.\mathbf{n})\mathrm{d}\sigma$ il reste donc la formulation variationnelle pour une itération :

$$\begin{cases} \int_\Omega \nabla\mathbf{u}.\nabla\mathbf{v} + \frac{1}{\nu}\int_\Omega[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}].\mathbf{v} & \\ \qquad - \int_\Omega p\mathrm{div}\mathbf{v} - \int_\Omega q\mathrm{div}\mathbf{u} &= \frac{1}{\nu}\int_\Omega(\mathbf{uold}.\nabla)\mathbf{uold}.\mathbf{v} \\ \mathbf{u}_{|\Gamma_D} &= g \end{cases}$$

Le dernier travail de l'étude numérique est donc définir les espaces discrets. Soit le domaine de calcul Ω, $\partial\Omega$ est la partie du bord de domaine dont la vitesse s'annule. Plusieurs choix sont possible, mais dans ce projet, on travail avec un exemple qui marche très bien dans ce cas, l'élement fini de type Taylor-Hood (P1-P2). On définit les espaces :

$$\begin{aligned} X_h &= \left\{\mathbf{v} \in C^0(\overline{\Omega})^d; \mathbf{v}_{|\kappa} \in \mathrm{P}_2^d \quad \forall\kappa \in T_h, \mathbf{v}_{|\partial\Omega} = 0\right\} \\ M_h &= \left\{q \in C^0(\overline{\Omega}); q_{|\kappa} \in \mathrm{P}_1 \quad \forall\kappa \in T_h\right\} \cap \mathrm{L}^2_0(\Omega) \end{aligned}$$

Le problème discret : La question qui se pose dans le cadre discret est donc trouver $\begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} \in X_h \times M_h$ tels que $\forall \begin{bmatrix} \mathbf{v} \\ q \end{bmatrix} \in X_h \times M_h$, on a :

$$\begin{cases} \int_\Omega \nabla\mathbf{u}.\nabla\mathbf{v} + \frac{1}{\nu}\int_\Omega[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}].\mathbf{v} \\ \qquad\qquad - \int_\Omega p\mathrm{div}\mathbf{v} - \int_\Omega q\mathrm{div}\mathbf{u} &= \frac{1}{\nu}\int_\Omega(\mathbf{uold}.\nabla)\mathbf{uold}.\mathbf{v} \\ \qquad\qquad \mathbf{u}_{|\Gamma_D} &= g \end{cases}$$

1.3 Reynold

La résolution de l'équation de Navier-Stokes a été toujours difficile pour les nombres de Reynolds très grand, qui représente le rapport entre les forces d'inertie et les forces visqueuses. Si le nombre de Reynolds est très grand, l'équation devient fortement non-linéaire car les phénomènes convectifs dominent. Les non-linéarités produiront des effets instationnaires pour un forçage stationnaire, des brisures de symétries par rapport aux conditions aux limites initiales, en d'autre termmes la turbulence, qui explose le système.

La méthode adopte donc une stratégie qui résoud l'équation en auguementant le nombre de Reynolds petit à petit. Soit la solution $\mathbf{u}_{REY}$ correspond au nombre de Reynolds=REY, on résoud le système en retrouvant les solutions $\mathbf{u}_{rey}$ correspond à $rey < REY$, et puis on augemente rey pour atteindre REY. L'algorithme est :

$$\begin{cases} \text{while } (rey < REY \quad) \\ \quad \{ \\ \qquad \text{while } (\|\mathbf{u} - \mathbf{uold})\| \geqslant 10^{-6}) \\ \qquad\quad \text{Résoudre } \mathbf{u} = \mathbf{uold} - \mathrm{dF}^{-1}_{\mathbf{uold}}(\mathrm{F}(\mathbf{uold})); \\ \qquad rey+ = incrementRey; \\ \quad \} \end{cases}$$

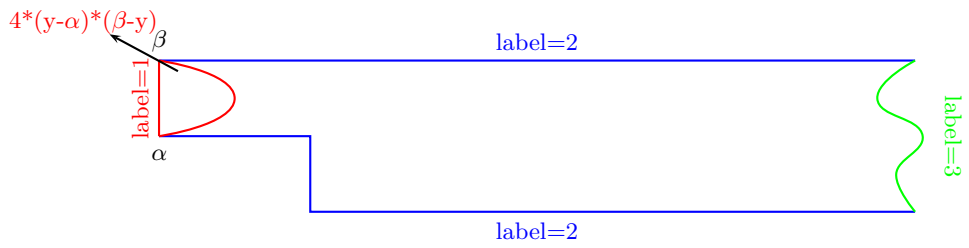
Dans ce projet, particulièrement dans ce programme en C++ et son domaine de calcul (La marche descendante), la résolution peut atteindre Reynolds=1400 pour les auguementations de 50 ou 100, partant de Reynolds=1.

Chapitre 2

Études informatiques

2.1 Création de maillage

Comme présenté, le domaine de calcul est une marche descendante de la forme :



Les degré de liberté setrouvant sur le label1 seront affectées par la condition de Dirichelet qui forme un profil parabolique de fluid entré. Sur le label2 la vitesse sera null. Sur le label3, les termes sont considéré comme l'intérieur du domaine (sont à calculer).

Création de Maillage : La création de maillage se fait par Freefem++, en utilisant le script : CreatMesh.edp. Ce script lit les données dans le fichier MeshCorner.txt contenant les trois coins du maillage et un coeficient indiquant la densité de noeuds pour le maillage. En l'éxécutant, il crée un fichier MyMesh.msh contenant les données de maillage, fait une visualisation de maillage créé.

```
prompt$FreeFem++ CreatMesh.edp
```

2.2 Conditions de bord

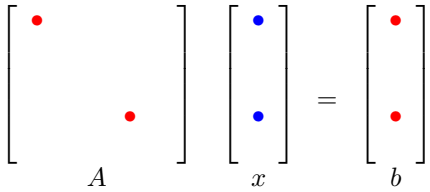
La stratégie pour affecter la condition de Dirichelet au bord sur le label1 est :

```
parcourir les arrêtes au bord
 {
  Trouver l'élément fini qui contient l'arrête;
  Parcourir les df locale
   {
    Si ((Label==1) et (df est sur l'arrête))
     {
      affecter la condition de bord au vecteur global;
     }
   }
 }
```

Donc l'implémentation dans C++ est :

```
for (int ke=0;ke<Th.nbe;ke++){
 int kf,k=Th.BoundaryElement(ke,kf);
 FElement FK(Vh[k]);
 int nbdf=FK.NbDoF();
 int Label=Th.borderelements[ke].lab;
 for (int df =0;df<nbdf;++df){//condition Dirichelet
  if(((Element::onWhatBorder[kf][FK.DFOnWhat(df)])!=0)&&(Label==1)){
    int p=FK.DFOnWhat(df);Rd P(FK.pPi_h(p));
    dirichelet[FK(df)]=Poissieulle(P,FK.FromASubFE(df));
    }
  }
}
```

Ensuite pour la résolution linéaire générale, on a une méthode de très grande valeur (TGV) servant à résoudre le système de type Ax=b excepté de quelques points du vecteur x :



Sur la matrice A, les points rouges sont sur la diagonale, de même lignes que ceux du vecteur b. En mutipliant les points rouges par une valeur très très grande (de l'ordre de 10^{30}) avant de résoudre le système, après la résolution, les points bleus prendront la même valeur que les points rouge du vecteur b avant de mutiplication. L'implémentation du code de cette méthode est suivant :

```
for(int ke=0;ke<Th.nbe;++ke){
 int kf,k=Th.BoundaryElement(ke,kf);
 FElement FK(Vh[k]);
 int ndf=FK.NbDoF();int Label=Th.borderelements[ke].lab;
  for (int df =0;df<ndf;++df){
  if (Element::onWhatBorder[kf][FK.DFOnWhat(df)] && (Label!=3)){
   if(FK.FromASubFE(df) < d){
    int i=FK(df);
    A[make_pair(i,i)] = tgv;
    b[i] = tgv*dirichelet[i];
    kkk++;
   }
  }
 }
}
```

2.3 Passage Locale-Globale

L'idée fondamentale du problème est à partir de la formulation variationnelle, on doit construire un système linéaire à résoudre. La question est comment construire ce système linéaire dans le cadre général. Cette question se pose sur un principe basique : le passage local-globale. Soit, l'espace fonctionnel discret $X_h \times M_h$ (FESpace) qui se construit par le maillage (Mesh) et le type d'élément fini (TypeOfFE). Un FESpace se compose aussi des FElement, qui décrit toutes les données dans un élément fini. Une fonction discrète dans FESpace est caractérisée par ses NBOFDF degré de liberté (NBOFDF est le nombre de degrée de liberté globale), localement, elle est caractérisé par ses nbofdf degré de liberté (nbofdf est le nombre de degré de liberté locale). L'algorithme pour toute opération sur les fonctions est dont de parcourir les FElement, effectuer les opérations sur les df locale, rassembler les résultat par le passage ses df au DF (degré de liberté global). En supposant que l'on a construit une matrice locale ML, le passage à la matrice M globale se fait par :

```
for(int i=0;i<nbDoF;i++)//Local==>GLOBALE
  for(int j=0;j<nbDoF;j++)
    M[make_pair<int,int>(FK(i),FK(j))] +=ML(i,j) ;
```

Il en est de même pour les vecteurs. Supposont que le vecteur local VL a été construit, le passage au vecteur b global se fait par :

```
for(int idf=0;idf<nbDoF;idf++)
  b[FK(idf)]+=VL[idf];
```

2.4 Passage Global-Local

Cette étape est necessaire dans le cas où le schéma est itératif, utilise les valeurs de la solution calculé de l'itération précédante pour calculer la solution après. C'est l'étape sachant une fonction $UOLD$ calculée, comment lire ces valeurs locales, afin de les passer dans le matrice A à doite et le vecteur b à gauche pour le système A.X=b. Les valeurs necessaires ne sont pas exactement les df de $UOLD$ au points d'interpolation, mais ce sont des valeurs de $UOLD$ aux points d'intégration. Pour chaque FElement, on doit donc parcourir les points d'intégration de la méthode d'intégration choisie, retrouver les valeurs necessaires de $UOLD$ sur ces points (valeurs de $UOLD$ et valeur de la dérivée de $UOLD$). Ces valeurs seront stoquées dans un vecteur local $uold$

```
for (int ipq=0;ipq<qf.n;ipq++){//qf.n=7
  /*======Les valeurs de UOLD local=====*/
 for(int ic=0;ic<d;ic++){
 uold(ic)=FK(qf[ipq],UOLD,ic,op_id);
   for(int ioperator=0;ioperator<d;ioperator++){
   Duold(ic,ioperator)=FK(qf[ipq],UOLD,ic,op_div[ioperator]);
   }
 }
}
```

2.5 Matrice locale

La construction de la matrice locale est fondamentale dans tout le programme. C'est cette étape qui implémente le schéma de l'algorithme mathématique. Partant le la formulation variationnelle :

$$\begin{aligned} \int_\Omega \nabla\mathbf{u}.\nabla\mathbf{v} + \frac{1}{\nu}\int_\Omega[(\mathbf{uold}.\nabla)\mathbf{u} + (\mathbf{u}.\nabla)\mathbf{uold}].\mathbf{v} & \\ - \int_\Omega p\mathrm{div}\mathbf{v} - \int_\Omega q\mathrm{div}\mathbf{u} \quad &= \frac{1}{\nu}\int_\Omega(\mathbf{uold}.\nabla)\mathbf{uold}.\mathbf{v} \end{aligned} \tag{2.1}$$

On la détaille dans le cas 3D. Dans ce cas, soit :

$$\begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} u1 \\ u2 \\ u3 \\ p \end{bmatrix}, \begin{bmatrix} \mathbf{v} \\ q \end{bmatrix} = \begin{bmatrix} v1 \\ v2 \\ v3 \\ q \end{bmatrix}, \begin{bmatrix} \mathbf{uold} \\ pold \end{bmatrix} = \begin{bmatrix} uold1 \\ uold2 \\ uold3 \\ pold \end{bmatrix}$$

La formulation (2.1) détallée dans le cas 3D sera :

$$\begin{aligned}
&\int_\Omega \begin{bmatrix} \frac{\partial u1}{\partial x}\frac{\partial v1}{\partial x} + \frac{\partial u1}{\partial y}\frac{\partial v1}{\partial y} + \frac{\partial u1}{\partial z}\frac{\partial v1}{\partial z} \\ \frac{\partial u2}{\partial x}\frac{\partial v2}{\partial x} + \frac{\partial u2}{\partial y}\frac{\partial v2}{\partial y} + \frac{\partial u2}{\partial z}\frac{\partial v2}{\partial z} \\ \frac{\partial u3}{\partial x}\frac{\partial v3}{\partial x} + \frac{\partial u3}{\partial y}\frac{\partial v3}{\partial y} + \frac{\partial u3}{\partial z}\frac{\partial v3}{\partial z} \end{bmatrix} \\
&+\tfrac{1}{\nu}\int_\Omega \left[\begin{pmatrix} uold1\frac{\partial u1}{\partial x}v1 + uold2\frac{\partial u1}{\partial y}v1 + uold3\frac{\partial u1}{\partial z}v1 \\ uold1\frac{\partial u2}{\partial x}v2 + uold2\frac{\partial u2}{\partial y}v2 + uold3\frac{\partial u2}{\partial z}v2 \\ uold1\frac{\partial u3}{\partial x}v3 + uold2\frac{\partial u3}{\partial y}v3 + uold3\frac{\partial u3}{\partial z}v3 \end{pmatrix} + \begin{pmatrix} u1v1\frac{\partial uold1}{\partial x} + u2v1\frac{\partial uold1}{\partial y} + u3v1\frac{\partial uold1}{\partial z} \\ u1v2\frac{\partial uold2}{\partial x} + u2v2\frac{\partial uold2}{\partial y} + u3v2\frac{\partial uold2}{\partial z} \\ u1v3\frac{\partial uold3}{\partial x} + u2v3\frac{\partial uold3}{\partial y} + u3v3\frac{\partial uold3}{\partial z} \end{pmatrix} \right] \\
&-\int_\Omega \begin{bmatrix} p\frac{\partial v1}{\partial x} \\ p\frac{\partial v2}{\partial y} \\ p\frac{\partial v3}{\partial z} \end{bmatrix} \\
&-\int_\Omega \; q\left[\frac{\partial u1}{\partial x} + \frac{\partial u2}{\partial y} + \frac{\partial u3}{\partial z}\right] \\
&= \tfrac{1}{\nu}\int_\Omega \begin{bmatrix} uold1\frac{\partial uold1}{\partial x}v1 + uold2\frac{\partial uold1}{\partial y}v1 + uold3\frac{\partial uold1}{\partial z}v1 \\ uold1\frac{\partial uold2}{\partial x}v2 + uold2\frac{\partial uold2}{\partial y}v2 + uold3\frac{\partial uold2}{\partial z}v2 \\ uold1\frac{\partial uold3}{\partial x}v3 + uold2\frac{\partial uold3}{\partial y}v3 + uold3\frac{\partial uold3}{\partial z}v3 \end{bmatrix}
\end{aligned}$$

A rappeler que dans ce cas, l'élément fini utilisé est de type Taylor-Hood (P1-P2), avec dimension 3D, chaque fonction se compose de 3 composantes de vitesse et une composante de presssion. Ce qui découpe la matrice locale en bloque.

BOUCLE pour chaque FElement
Parcourir les points d'intégration
Calcul des coef d'intégration ;
parcourir les composantes de la fonctions de solution
parcourir les composantes de la fonctions de test
parcourir les df de composante
rajouter des coeufficients dans la matrice locale ;

Pour parcourir les points d'intégrations :

```
for (int ipq=0;ipq<qf.n;ipq++){//qf.n=7
 w=0.;uold=0.;Duold=0.;
 QuadraturePoint Pq(qf[ipq]);
 //les coef pour le schemas general
 R coefadvect= CADVECT*mes* Pq.a;
 R coefgradgrad = CGRADGRAD*mes*Pq.a;
 R coefqdivv = CQDIVV*mes*Pq.a;
 R coefpp =1e-10*mes*Pq.a;
 FK.BF(whatd, Pq,w);
```

Calcul des valeurs locales de *uold* :

```
for(int ic=0;ic<d;ic++){
 uold(ic)=FK(qf[ipq],UOLD,ic,op_id);
 for(int ioperator=0;ioperator<d;ioperator++){
 Duold(ic,ioperator)= FK(qf[ipq],UOLD,ic,op_div[ioperator]);
 }
}
```

et pour parcourir les composantes des fonctions et ses df :

```
for(int ic=0;ic<d;ic++){//ic=composante v.
  for(int jc=0;jc<d;jc++){//jc=composante u
  int idfbegin=FK.dfcbegin(ic); int idfend=FK.dfcend(ic);
  int jdfbegin=FK.dfcbegin(jc); int jdfend=FK.dfcend(jc);
  ........
```

2.6 Passage au membre droite

Dans les cas des schémas itératif, le passage de la solution calculée au memebre à droite est aussi important et se fait de façon similaire que la construction de la matrice du système A.x=b. Comme la construction de la matrice du système, ce passage necessite en premier la construction locale des valeurs de la fonctions calculée aux points d'intégrations. La construction locale se fait en premier par parcourir localement les composantes de la fonction, puis parcourir les df locales de cette composante :

```
for(int ic=0;ic<d;++ic){
int dfbegin=FK.dfcbegin(ic),dfend=FK.dfcend(ic);
 for(int i=dfbegin;i<dfend;i++){
   for(int i_derivate=0;i_derivate<d;i_derivate++){
      //(uold.grad)uold.vi
      VL[i] +=coefadvec*uold(i_derivate)
              *Duold(ic,i_derivate)*w(i,ic,op_id);
   }
 }
}
```

Après chaque calcul local, il ne reste à faire l'implément dans le vecteur global :

```
.........
   for(int idf=0;idf<nbDoF;idf++)
      b[FK(idf)]+=VL[idf];
............
```

2.7 Structure du programme et utilisation

Les programmes de calcul est codé en C++. Quelques script de FreeFem++ sont programmés aussi utilisant le même algorithme mathématique, afin de comparer la solution Cpp-FFpp pour une validation sure des calcul C++.

A remarquer que les script FreeFem++ ne prend pas d'argument, on doit donc créer les fichiers de texte .txt qui ont pour rôle stoquer les données communes pour tous les calcul.

- MeshCorner.txt : contient les donnée pour créer le maillage.
- CoefSchemas.txt :contient le nombre Reynolds et sont incrémentation
- nameMesh.txt : Dans le cas où plusieurs maillage existe, un seul peut être utilisé pour les calculs, le fichier nameMesh.txt contient le nom du fichier maillage qui sera utilisé pour le calcul.

Pour les scripts de FreeFem++

- CreatMesh.edp : lit les données dans MeshCorner.txt (3 coins et la densité des noeuds)pour créer le maillage
- NSstatNCT.edp : lit les données dans MeshCorner.txt, CoefSchemas.txt,nameMesh.txt pour calculer la solution de Navier-Stokes. La solution est stoquée dans NSsFFpp.sol
- StokesNCT.edp : lit les données dans MeshCorner.txt, CoefSchemas.txt,nameMesh.txt pour calculer la solution de Stokes. La solution est stoquée dans Stokes.sol

```
FreeFem++ CreatMesh.edp
FreeFem++ NSstatNCT.edp
FreeFem++ StokesNCT.edp
```

Pour les programme C++ :

- NavierStokes [meshfile] : Calcule la solution de Navier-Stokes en sortant le résultat dans NSsCpp.sol. Les valeurs pour visualiser la condition de bord Dirichelet est stoquée dans NSsDiriCpp.sol
- Stokes [meshfile] : Calcule la solution de Stokes en sortant le résultat dans StokesCpp.sol. Les valeurs pour visualiser la condition de bord Dirichelet est stoquée dans StokesDiriCpp.sol
- Visualization [Sol1] [Sol2] : crée un script de FF++ pour visualiser les solutions, leur différence. l'argument Sol2 est facultative.
- GUI_NS : fait tout dans un GUI.

```
Prompt$ ./NavierStokes MyMesh.msh
Prompt$ ./Stokes MyMesh.msh
Prompt$ ./Visualization NSsDiriCpp.sol
Prompt$ ./Visualization NSsCpp.sol NSsFFpp.sol
Prompt$ ./GUI\_NS
```

Voici un tableau décrit les interactions entres les fichiers et les programmes :

Fichier à lire	**Programme**	**Fichiers sorties**
MeshCorner.txt	CreatMesh.edp	MyMesh.msh
MeshCorner.txt, CoefSchemas.txt, nameMesh.txt	NSstatNCT.edp	NSsFFpp.sol
MeshCorner.txt, CoefSchemas.txt, nameMesh.txt	StokesNCT.edp	StokesFFpp.sol
*.msh	./NavierStokes	NSsDiriCpp.sol, NSsCpp.sol
*.msh	./Stokes	StokesDiriCpp.sol, StokesCpp.sol
.sol1 [.sol2]	./Visualization	
*.txt	./GUI_NS	*.txt

2.8 Résultats en image

Voici les images solutions de C++, FF++ et la différence des deux solutions :

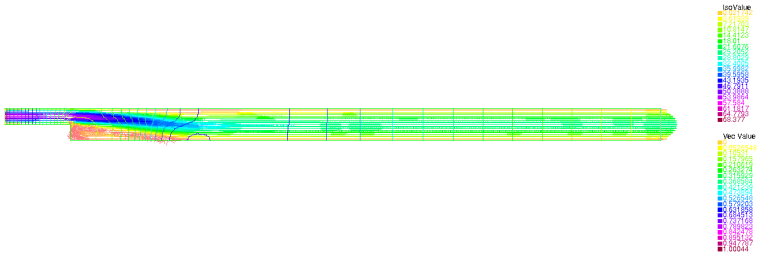


FIG. 2.1 – Solution C++

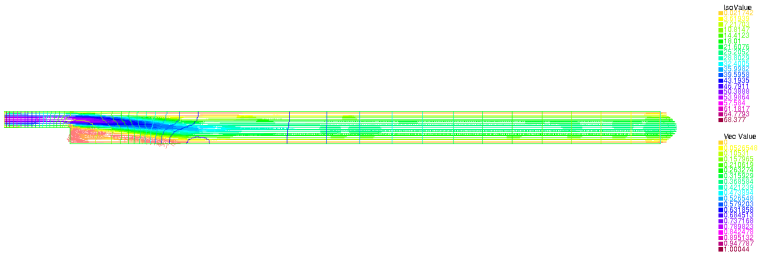


FIG. 2.2 – Solution FF++

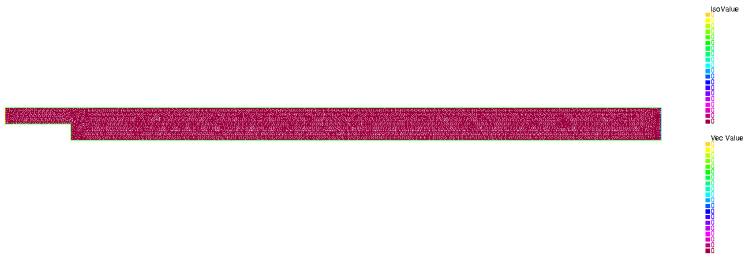


FIG. 2.3 – Différence C++ FF++

Chapitre 3

Qt Programmation GUI

Qt est un framework C++ permettant de développer des applications GUI multiplates-formes. Qt permet aux programmeurs d'employer une seule arborescence source pour des application qui s'éxécuteront sous Windows, Mac OS X, Linux et de nombreuses autres versions d'Unix avec X11. Les bibliothèques et outils Qt font également partie de Qtopia Core. J'ai l'idée de programmation Qt quand je suis tombé sur le siteweb :

- Introduction : http ://www.siteduzero.com/tutoriel-3-11240-introduction-a-qt.html

qui est une très bonne introduction à la programmation Qt. Ensuite, on peut consulter les sites officiels pour une pratique et compréhension plus approfondie :

- Documentation [fr] : http ://doc.qtfr.org/
- Documentation [en] : http ://doc.trolltech.com/

Pour apprendre Qt, j'ai consulté en premier l'introduction sur le siteduzéro pour comprendre la compilation, l'installation de Qt. Ensuite, je suis parti sur le site officielle pour télécharger les exemples, les compiler et les corriger pour approfondir. Pour une lecture plus structurée, on pourra consulter [4]

3.1 Présentation du GUI_NS

Mon programme GUI_NS se compose principalement des objets QLineEdit, QPushButton, QProcess. Les QLineEdit permet de saisir un texte, c'est là que l'utilisateur fait entrer ses données désirées. QPushButton est simplement un ... bouton. QProcess est comme un processu, fait tourner des programmes extérieurs. Le mécanisme général est à chaque fois qu'on presse sur un Bouton, il émet un signal, qui fait appel à q̈uelques choses:Celles-ci peut être d'appel une fonction, démarrer un processu, déclencher le chronomètre, lire les données dans les champs, écrire les données dans les champs, dans les fichiers..ect...Ce mécanisme se réalise grâce à la méthode statique QObject : :connect(Objet1,SIGNAL(...),Objet2,SLOT(...)).

Fenetre Creat Mesh :

Dans cette fenetre, on a la connection :

```
QObject::connect(button_CreatMesh,SIGNAL(clicked()),
                 this,SLOT(CreatMeshSLOT()));
```

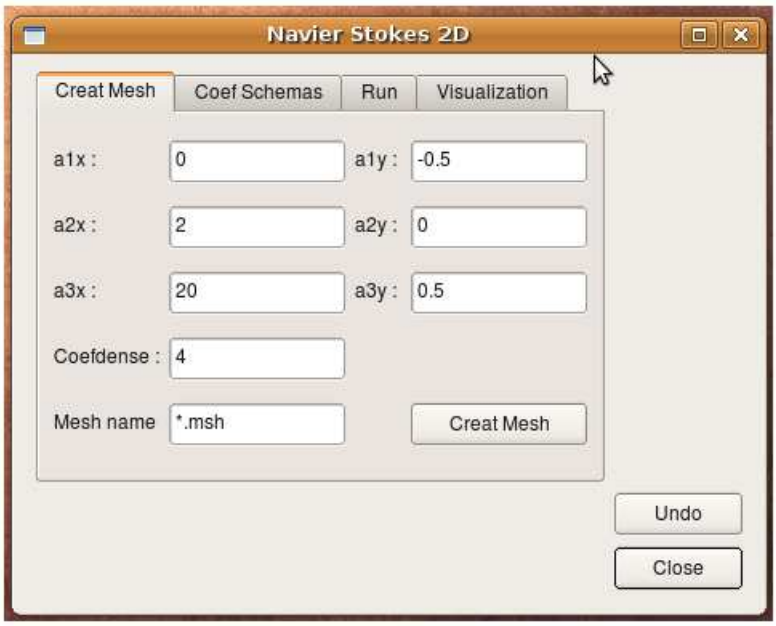


FIG. 3.1 – Fênetre Creat Mesh

qui fait que si je clike sur le bouton CreatMesh, la fonction spéciale (SLOT) CreatMeshSLOT() sera appelée, elle lit les données dans les champs saisies, les écrit dans le fichier MeshCorner.txt, appelle le script CreatMesh.edp :

```
void  CreatMeshTab::CreatMeshSLOT(){
  system("rm -f *.msh");
  double a1x_value=LineEdita1x->text().toDouble();
  double a1y_value=LineEdita1y->text().toDouble();
  double a2x_value=LineEdita2x->text().toDouble();
  double a2y_value=LineEdita2y->text().toDouble();
  double a3x_value=LineEdita3x->text().toDouble();
  double a3y_value=LineEdita3y->text().toDouble();
  int Coefdense_value=LineEditCoefdense->text().toInt();
  {
    ofstream fileout("MeshCorner.txt");
    fileout<<a1x_value<<" "<<a1y_value<<endl
           <<a2x_value<<" "<<a2y_value<<endl
           <<a3x_value<<" "<<a3y_value<<endl
           <<Coefdense_value<<endl;
  }
  system("FreeFem++ CreatMesh.edp");
  char commandeSaveMesh[256];
  strcpy(commandeSaveMesh,"mv MyMesh.msh ");
  strcat(commandeSaveMesh,
         LineEditNameMeshSave->text().toStdString().c_str());
  system(commandeSaveMesh);
  //cout<<commandeSaveMesh<<endl;
}
```

Fenetre Coef Schemas :

De même, la fenetre Coef Schéma a pour connection :

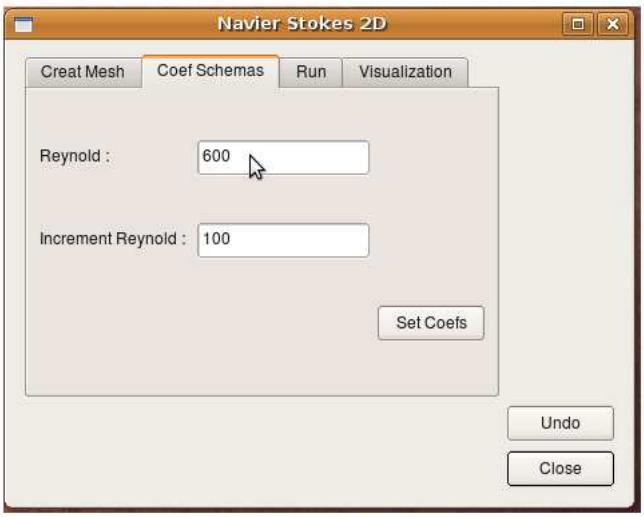


FIG. 3.2 – Fênetre Coef Schéma

```
QObject::connect(button_SetCoef,SIGNAL(clicked()),
                 this,SLOT(SetCoefSLOT()));
```

En appuyant sur le bouton **Set Coef**, on écrit ces données dans le fichier CoefSchemas.txt et modifie les données dans la fenetre Run.

Fenetre Run :

La fenetre Run est la plus compliquée, contient :

```
QObject::connect(TimerChronoCpp,SIGNAL(timeout()),
                 this,SLOT(UpdateTimeCppSLOT()));
QObject::connect(TimerChronoFFpp,SIGNAL(timeout()),
                 this,SLOT(UpdateTimeFFppSLOT()));
QObject::connect(RunProcess,SIGNAL(finished(int,QProcess::ExitStatus)),
                 TimerChronoCpp,SLOT(stop()));
QObject::connect(RunProcess,SIGNAL(finished(int,QProcess::ExitStatus)),
                 TimerChronoFFpp,SLOT(stop()));
QObject::connect(button_LoadMesh,SIGNAL(clicked()),
                 this,SLOT(LoadMeshSLOT()));
QObject::connect(button_RunCpp,SIGNAL(clicked()),
                 RunProcess,SLOT(kill()));
QObject::connect(button_RunCpp,SIGNAL(clicked()),
                 this,SLOT(RunCppSLOT()));
QObject::connect(button_RunFFpp,SIGNAL(clicked()),
                 RunProcess,SLOT(kill()));
QObject::connect(button_RunFFpp,SIGNAL(clicked()),
                 this,SLOT(RunFFppSLOT()));
QObject::connect(button_Break,SIGNAL(clicked()),
                 RunProcess,SLOT(kill()));
QObject::connect(button_Break,SIGNAL(clicked()),
                 this,SLOT(ResetTimeSLOT()));
```

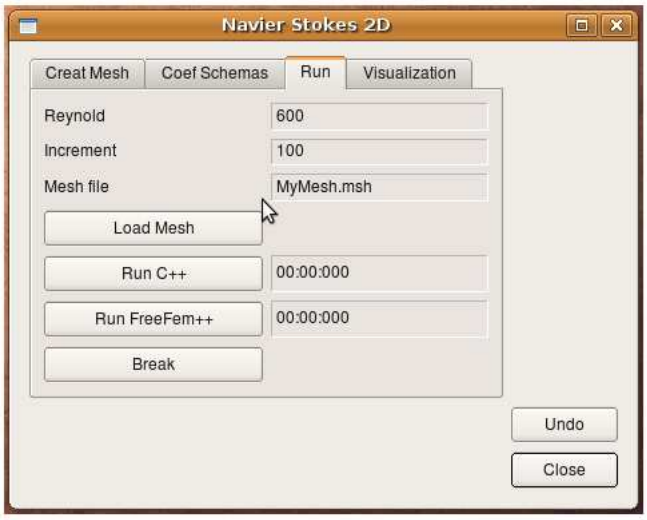


FIG. 3.3 – Fênetre Run

Le bouton **Load Mesh** recherche le fichier de maillage, modifie le text dans le champs Mesh file. Le bouton **Run C++** appelle le programme C++, déclanche le chronomètre Cpp. Le bouton **Run FreeFem++** appelle le programme FF++, déclanche le chronomètre FFpp. Le bouton **Break** arrête tout programme de façon brutale et remet le chronomètre à zéro.

Fenetre Visualization :

Cette fênetre est aussi simple que la fenetre Coef Schéma, qui lit les fichiers de solution, appelle le programme C++ ./visualisation (qui crée encore un script FreeFem++) pour visualiser les solutions. Le bouton **Load Sol1** recherche la première solution à visualiser. Le bouton **Load Sol2** recherche la deuxième solution à visualiser. Le bouton **Visualization** appelle le programme :

```
Prompt$ ./Visualization First.sol Second.sol
```

Si on ne met rien dans le champs saisie de la deuxième solution, **Visualization** appelle :

```
Prompt$ ./Visualization First.sol
```

```
QObject::connect(button_LoadSol1,SIGNAL(clicked()),
                 this,SLOT(LoadSol1SLOT()));
QObject::connect(button_LoadSol2,SIGNAL(clicked()),
                 this,SLOT(LoadSol2SLOT()));
QObject::connect(button_Visualization,SIGNAL(clicked()),
                 this,SLOT(VisualizationSLOT()));
```

Remarque sur QProcess : Quelques SLOT fait appel aux programme externe sous forme processu. Pour ne pas s'occuper des iostandard du programme externe, il faut le regler en mode spécifique utilisant la fonction setProcessChannelMode(mode) de la classe QProcess.

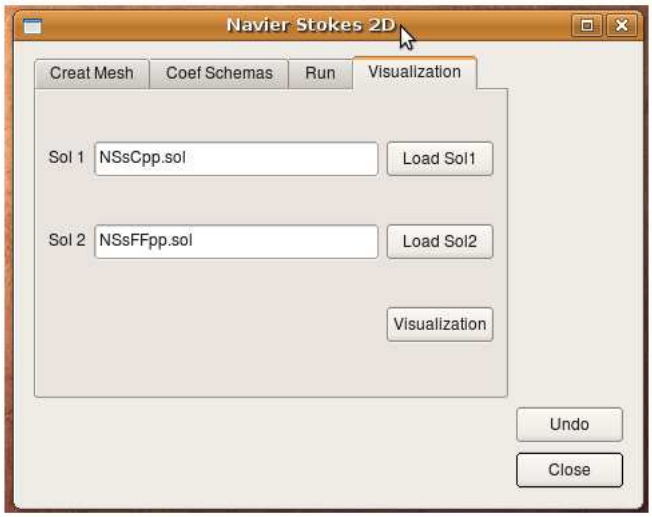


FIG. 3.4 – Fenetre Visualisation

```
Process::setProcessChannelMode(QProcess::ForwardedChannels);
```

Remarque sur QTime et QTimer : Pour la construction de l'horloge de calcul, il faut deux objets qui se ressemble en notation QTime et QTimer. Ces deux objets sont utilisé pour de buts différents QTime s'utilise comme une horloge, tandis que QTimer s'utiliser comme un chronomètre :

```
QTime *Chronometre;QTime *TempsCalcul;
QTimer *TimerChronoCpp;QTimer *TimerChronoFFpp;
```

TimerChronoCpp, TimerChronoFFpp sont utilisés pour chronomètrer les durées de calcul respectivement de C++ et FF++. Ensuite, en utilisant la fonction de QTime : :elapsed(), on peut calculer le temps écoulé souhaité.

3.2 Utilisation

La complilation est assez compliquée utilisant un outil spécifique : qmake. Cet outil génère automatiquement le fichier Makefile pour le projet. Pour cette raison, il faut mettre le programme dans un directoire différent que le projet principale, le compiler, et l'amener dans le projet après la compilation pour l'utiliser.

Bibliographie

[1] Albert Cohen. *Approximation Variationnelle des EDP*. Cours DEA LJLL MN407.

[2] Jean-Côme CHARPENTIER Denis BITOUZÉ. *LaTeX*. PEARSON Education France, 2006.

[3] Frédéric Hecht. *Documentation FreeFem++*. LJLL.

[4] Mark Summerfield Jasmin Blanchette. *Qt4 et C++ Programmation d'interface GUI*. Campus-Press, 2007.

[5] R.H.Gallagher G.Carey J.T.Oden O.C.Zienkiewicz. *Finite Elements in Fluids*. John Wiley Son, 1985. volume 6.

[6] Vivette Girault Pierre-Arnaud Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer-Verlag, 1986.

[7] Francois Thomasset. *Implementation of Finite Element Methods for Navier-Stokes Equations*. Springer-Verlag, 1981.