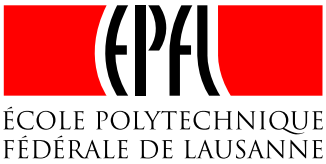


Lignes de courant de fluides incompressibles

Gwenol Grandperrin

Assistants : Simone Deparis, Davide Detomi

Professeur : Alfio Quarteroni

Semestre de printemps 2007-2008

Table des matières

1 Introduction

Dans ce projet nous voulons trouver et représenter les lignes de courant d'un fluide incompressible. Il est en effet possible de calculer les lignes de courant comme des lignes équipotentielles d'un champ qui sont la solution d'une équation aux dérivées partielles.

Nous considèrerons trois problèmes modèles dont nous discuterons l'existence et l'unicité des solutions. Ensuite nous résoudrons des problèmes correspondant à ces modèles en approchant les solutions par éléments finis. En particulier nous verrons comment ramener le calcul des lignes de courant à la résolution du problème de Poisson.

Finalement nous expliquerons une méthode permettant la construction d'un solveur éléments finis 2D permettant la résolution des différents problèmes étudiés.

2 Quelques résultats importants

2.1 Résultats d'analyse fonctionnelle

Commençons par donner quelques définitions et résultats classiques d'analyses fonctionnelles. Ces derniers nous serons utiles par la suite pour traiter les différents problèmes.

2.1.1 Définition (Espace de Sobolev)**.** *On considère l'espace $L^2(\Omega)$ définit par*

$$L^2(\Omega) = \left\{ f : \Omega \to \mathbb{R} \text{ mesurable au sens de Lebesgue} \middle| \int_\Omega |f(x)|^2 \, dx < \infty \right\}.$$

*On désigne par $H^k(\Omega)$ l'*espace de Sobolev *d'ordre k définit par*

$$H^k(\Omega) = \{ f \in L^2(\Omega) \mid D^\alpha f \in L^2(\Omega), |\alpha| \leq k \}$$

où $D^\alpha f$ est la dérivée de f prise au sens des distributions. De plus on notera

$$H_0^k(\Omega) = \{ f \in H^k(\Omega) \mid f = 0 \text{ sur } \partial\Omega \}$$

$$H_D^1 = \left\{ f \in H^1 \mid f = 0 \text{ sur } \Gamma_D, D \subset \partial\Omega, D \text{ de mesure non nulle} \right\}.$$

2.1.2 Lemme (Egalité de Green)**.** *Soit $u : \Omega \to \mathbb{R}$ une fonction de classe $C^2(\overline{\Omega})$.*

$$\nabla \cdot (v \nabla u) = v \Delta u + \nabla u \nabla v.$$

2.1.3 Théorème (Lax-Milgram)**.** *Soient H un espace de Hilbert muni de la norme $\|.\|_H$, $a(u,v) : H \times H \to \mathbb{R}$ une forme bilinéaire et $F(v) : H \to \mathbb{R}$ une fonctionnelle linéaire continue. Supposons de plus que $a(u,v)$ soit continue, i.e :*

$$\exists M > 0 : \quad |a(u,v)| \leq M \|u\|_H \|v\|_H \qquad \forall u, v \in H,$$

et coercive, i.e :

$$\exists \alpha > 0 : \quad |a(v,v)| \geq \alpha \|v\|_H^2 \qquad \forall v \in H.$$

Alors il existe une unique solution $u \in H$ au problème

$$\text{Trouver } u \in H : \quad a(u,v) = F(v) \qquad \forall v \in H.$$

De plus la norme de la solution est bornée par les données du problème :

$$\|u\|_H \leq \frac{1}{\alpha} \|F\|_{H'}$$

où H' est l'espace dual de H et $\|.\|_{H'}$ sa norme.

2.2 Méthode de quadrature de Gauss

Bien souvent le calcul d'intégrales dans le cas bidimensionnel (voire tridimensionnel) peut s'avérer très difficile. C'est pourquoi dans la pratique, on utilise des méthodes numériques pour approcher de telles intégrales. La méthode de quadrature de Gauss est la méthode que nous utiliserons dans ce projet.

2.2.1 Cas unidimensionnel

Soit $f : [-1,1] \to \mathbb{R}$ une fonction lisse et intégrable. On souhaite calculer

$$\int_{-1}^{1} f(x)dx.$$

Une telle intégrale peut être approchée grâce à une formule d'intégration numérique du type

$$\int_{-1}^{1} f(x)dx = \sum_{l=1}^{n} f(\hat{\xi}_l)\hat{w}_l + R \cong \sum_{l=1}^{n} f(\hat{\xi}_l)\hat{w}_l$$

où n est le nombre de *points de quadrature*, $\hat{\xi}_l$ est le l^{e}*point d'intégration*, $\hat{w}_l$ est le *poids* du l^{e}point d'intégration et R est le reste.

Différentes méthodes existent pour déterminer les $\hat{\xi}_l$ et les $\hat{w}_l$, nous ne nous intéresserons qu'à la méthode de quadrature de Gauss.

2.2.2 Cas bidimensionnel

Soit $f : [-1,1]^2 \to \mathbb{R}$ une fonction lisse et intégrable. On souhaite calculer

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy.$$

En fait nous allons appliquer successivement le cas monodimensionnel sur chacune des coordonnées.

$$\begin{aligned}\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy &\cong \int_{-1}^{1}\left(\sum_{l=1}^{n} f(\hat{\xi}_l, y)\hat{w}_l^1\right)dy \\ &\cong \sum_{k=1}^{m}\sum_{l=1}^{n} f(\hat{\xi}_l, \hat{\eta}_k)\hat{w}_l^1\hat{w}_k^2\end{aligned}$$

où $\hat{\xi}_1, \ldots, \hat{\xi}_n$ et $\hat{w}_1^1, \ldots, \hat{w}_n^1$ sont les points et les poids liés à la variable x, et $\hat{\eta}_1, \ldots, \hat{\eta}_m$ et $\hat{w}_1^2, \ldots, \hat{w}_m^2$ ceux liés à la variable y.

2.2.3 Remarque. *Soit $g : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ une fonction lisse et intégrable. On souhaite calculer*

$$\int_\Omega g d\Omega.$$

Il suffit de nous ramener grâce à un changement de variable à l'intégrale sur $[-1,1]^n$ pour pouvoir appliquer notre formule de quadrature.

Dans le cas où l'on souhaite intégrer g sur des triangles, se référer au paragraphe 2.2.7.

2.2.4 Définition (Polynôme de Legendre)**.** *Soit $n \in \mathbb{N}$. Les polynômes du type*

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2-1)^n$$

sont appelés polynômes de Legendre.

2.2.5 Lemme. *Les polynômes de Legendre peuvent être définis via la relation de récurrence suivante :*

- $P_0(x) = 1$,
- $P_1(x) = x$,
- $P_{n+1}(x) = \frac{1}{n+1}\left((2n+1)xP_n(x) - nP_{n-1}(x)\right)$ *avec* $n \geq 2$.

2.2.6 Proposition (Quadrature de Gauss)**.** *Soit $n \geq 1$ et soient $\hat{\xi}_1, \ldots, \hat{\xi}_n$ les n racines du polynôme de Legendre $P_n(x)$. Alors en posant*

$$\hat{w}_k = \frac{2}{(1-\xi_k^2)P_n'(\xi_k)^2}$$

on obtient une quadrature d'ordre $2n-1$.

2.2.7 Remarque. *Il existe des tables dans certains ouvrages qui donnent également les poids et les points de Gauss pour les simplexes. C'est le cas de l'ouvrage de Ern et Guermond [3] aux pages 317 à 319.*

3 Problèmes modèles

Nous allons considérer quelques problèmes qui nous servirons par la suite.

3.1 Laplacien avec conditions de Dirichlet

On va s'intéresser au problème suivant :

$$\begin{cases} -\Delta u &= f \quad \text{dans } \Omega \\ u &= 0 \quad \text{sur } \partial\Omega \end{cases}$$

où Ω est un domaine ouvert et borné de $\mathbb{R}^n$ et $f \in L^2(\Omega)$.

Formulation variationnelle

Soit $v \in H_0^1(\Omega)$ une fonction de test. Alors

$$\int_\Omega -\Delta u \cdot v = \int_\Omega f \cdot v.$$

Développons encore le terme de gauche.

$$\begin{aligned}\int_\Omega -\Delta u \cdot v &= -\int_\Omega \nabla \cdot (v \nabla u) + \int_\Omega \nabla u \nabla v \\ &= -\int_{\partial\Omega} v \nabla u \cdot \nu + \int_\Omega \nabla u \nabla v\end{aligned}$$

où ν est la normale extérieure à Ω. Comme $v \in H_0^1(\Omega)$, nous avons que v s'annule sur $\partial\Omega$. Ainsi notre problème sous forme variationnelle s'écrit

$$\left\{ \begin{array}{l} \text{Trouver } u \in H_0^1(\Omega) \text{ tel que} \\ \displaystyle\int_\Omega \nabla u \nabla v = \int_\Omega fv \quad \forall v \in H_0^1(\Omega) \end{array} \right.$$

3.2 Laplacien avec conditions de Neumann

On va s'intéresser au problème suivant :

$$\left\{ \begin{array}{rcll} -\Delta u &=& f & \text{dans } \Omega \\ \frac{\partial u}{\partial \nu} &=& g & \text{sur } \partial\Omega \end{array} \right.$$

où Ω est un domaine ouvert et borné de $\mathbb{R}^n$ et $f, g \in L^2(\Omega)$.

Formulation variationnelle

Soit $v \in H^1(\Omega)$. On obtient alors

$$\int_\Omega -\Delta u v = \int_\Omega fv.$$

Développons le membre de gauche.

$$\begin{aligned}\int_\Omega -\Delta u v &= -\int_\Omega \nabla(v \nabla u) + \int_\Omega \nabla u \nabla v \\ &= -\int_{\partial\Omega} v \nabla u \nu + \int_\Omega \nabla u \nabla v \text{ par le théorème de la divergence} \\ &= -\int_{\partial\Omega} \frac{\partial u}{\partial \nu} v + \int_\Omega \nabla u \nabla v\end{aligned}$$

En utilisant la condition de Neumann on obtient le problème sous sa forme faible

$$\left\{ \begin{array}{l} \text{Trouver } u \in H^1(\Omega) \text{ tel que} \\ \displaystyle\int_\Omega \nabla u \nabla v = \int_\Omega fv + \int_{\partial\Omega} gv \end{array} \right.$$

3.3 Laplacien avec conditions mixtes

On va s'intéresser au problème suivant :

$$\left\{ \begin{array}{rcll} -\Delta u &=& f & \text{dans } \Omega \\ u &=& g_1 & \text{sur } \Gamma_D \\ \frac{\partial u}{\partial \nu} &=& g_2 & \text{sur } \Gamma_N \end{array} \right.$$

où Ω est un domaine ouvert et borné de $\mathbb{R}^n$ et $f, g_1, g_2 \in L^2(\Omega)$.

Formulation variationnelle

Soit $v \in H^1_D(\Omega)$. On obtient alors

$$\int_\Omega -\Delta u v = \int_\Omega f v.$$

Développons le membre de gauche.

$$\begin{aligned}
\int_\Omega -\Delta u v &= -\int_\Omega \nabla(v \nabla u) + \int_\Omega \nabla u \nabla v \\
&= -\int_{\partial\Omega} v \nabla u \nu + \int_\Omega \nabla u \nabla v \text{ par le théorème de la divergence} \\
&= -\int_{\partial\Omega} \frac{\partial u}{\partial \nu} v + \int_\Omega \nabla u \nabla v
\end{aligned}$$

Or comme $v \in H^1_D(\Omega)$,

$$\int_{\partial\Omega} \frac{\partial u}{\partial \nu} v = \underbrace{\int_{\Gamma_D} \frac{\partial u}{\partial \nu} v}_{=0} + \int_{\Gamma_N} \frac{\partial u}{\partial \nu} v = \int_{\Gamma_N} g v$$

En utilisant la condition de Neumann on obtient le problème sous sa forme faible

$$\left\{ \begin{array}{l} \text{Trouver } u \in H^1_D(\Omega) \text{ tel que} \\ \displaystyle\int_\Omega \nabla u \nabla v = \int_\Omega f v + \int_{\Gamma_N} g v \end{array} \right.$$

3.4 Discussion sur l'existence et l'unicité de solutions

Nous allons maintenant nous intéresser à l'existence et l'unicité des problèmes que nous venons de voir. Pour montrer l'existence et l'unicité de la solution, nous allons vérifier les hypothèses du théorème de Lax-Milgram.

Commençons par montrer la continuité de la forme bilinéaire $a(u,v) = \int_\Omega \nabla u \nabla v$.

$$\begin{aligned}
|a(u,v)| &= \left| \int_\Omega \nabla u \nabla v \right| \\
&\leq \|\nabla u\|_{L^2} \|\nabla v\|_{L^2} \text{ par l'inégalité de Cauchy-Schwarz} \\
&= |u|_{H^1} |v|_{H^1} \\
&\leq \|u\|_{H^1} \|v\|_{H^1}
\end{aligned}$$

Montrons la continuité de la fonctionnelle F. En fait il nous suffit de montrer qu'elle est bornée.

$$\begin{aligned}
|F(v)| &= \left| \int_\Omega f v + \int_{\partial\Omega} g v \right| \\
&\leq \left| \int_\Omega f v \right| + \left| \int_{\partial\Omega} g v \right| \\
&\leq \|f\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} + \|g\|_{L^2(\partial\Omega)} \|v\|_{L^2(\partial\Omega)} \text{ par l'inégalité de Cauchy-Schwarz} \\
&\leq \|f\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} + \|g\|_{L^2(\partial\Omega)} C_T \|v\|_{H^1(\Omega)} \text{ par l'inégalité de Trace} \\
&\leq (\|f\|_{L^2(\Omega)} + C_T \|g\|_{L^2(\partial\Omega)}) \|v\|_{H^1(\Omega)}
\end{aligned}$$

Il nous reste à montrer que $a(u,v)$ est coercive. Pour montrer ce résultat, nous avons besoin de l'inégalité de Poincaré. Malheureusement, elle n'est pas valable pour $H^1(\Omega)$. Donc nous ne pouvons pas affirmer l'existence et l'unicité de la solution du problème du paragraphe 3.2.

En revanche l'inégalité de Poincaré est valable pour H^1_0 ou H^1_D. Supposons donc que $v \in H^1_0$ ou $v \in H^1_D$. Ainsi

$$\begin{aligned} \|v\|^2_{H^1} &= \|v\|^2_{L^2} + |v|^2_{H^1} \\ &\leq C^2_\Omega \, |v|^2_{H^1} + |v|^2_{H^1} \text{ par l'inégalité de Poincaré} \\ &= (1 + C^2_\Omega) \, |v|^2_{H^1} \\ &= (1 + C^2_\Omega) a(v,v). \end{aligned}$$

D'où $a(v,v) \geq \frac{1}{1+C^2_\Omega} \|v\|^2_{H^1}$ avec C_Ω une constante positive.

Ainsi par le théorème de Lax-Milgram, l'existence et l'unicité des solutions des problèmes des paragraphes 3.1 et 3.3 sont démontrées.

4 Problème unidimensionnel

On va s'intéresser au problème suivant :

$$\left\{ \begin{array}{rcll} -\Delta u & = & f & \text{dans } \Omega \\ u & = & 0 & \text{sur } \partial\Omega \end{array} \right.$$

Cependant nous allons nous placer dans le cas particulier où $\Omega \subset \mathbb{R}$.

4.1 Approximation par éléments finis

Nous allons formuler le problème discret éléments finis dans le cas où $\Omega \subset \mathbb{R}$. Remarquons que dans ce cas la formulation du paragraphe 3.1 devient

$$\left\{ \begin{array}{l} \text{Trouver } u \in H^1_0(\Omega) \text{ tel que} \\ \displaystyle\int_\Omega u'v' \quad = \quad \int_\Omega fv \quad \forall v \in H^1_0(\Omega) \end{array} \right.$$

Définissons la forme bilinéaire $a(u,v) = \int_\Omega u'v'$ et la fonctionnelle $F(v) = \int_\Omega fv$ pour obtenir

$$\left\{ \begin{array}{l} \text{Trouver } u \in H^1_0(\Omega) \text{ tel que} \\ a(u,v) \quad = \quad F(v) \quad \forall v \in H^1_0(\Omega) \end{array} \right.$$

Le paragraphe 3.4 nous assure l'existence et l'unicité de la solution pour ce problème.

Nous devons discrétiser notre domaine Ω. Pour cela, on introduit une partition $\tau_h = \{x_0 = a, x_1, \ldots, x_n = b\}$ de $\Omega = [a,b]$ telle que $x_{j+1} - x_j = h$ pour $j = 0, \ldots, n-1$. Ensuite, on définit la famille d'intervalles $I_j = [x_j, x_j + 1]$ pour $j = 0, \ldots, n-1$.
On introduit les éléments finis de Lagrange de degré $r \geq 1$.

$$X^r_h = \left\{ v_h \in C^0([a,b]) \mid v \mid_{I_j} \in \mathbb{P}^r, j = 0, \ldots, n-1 \right\}$$

Soit maintenant $V_h \subset H^1_0(\Omega)$ définit par

$$V_h = \left\{ v_h \in X^1_h \mid v_h(a) = v_h(b) = 0 \right\}.$$

Une base possible pour cet espace est $\{\varphi_1, \ldots, \varphi_{n-1}\}$ où

$$\varphi_i(x) = \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & \text{si } x_{i-1} \leq x \leq x_i \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & \text{si } x_i \leq x \leq x_{i+1} \\ 0 & \text{sinon} \end{cases}$$

Remarquons que $\varphi_i(x_j) = \delta_{ij}$.

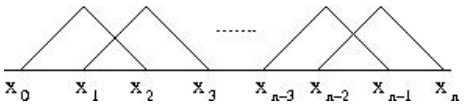


FIG. 1 – Illustration des éléments de la base

On peut maintenant exprimer l'approximation de la solution u_h dans cette base. C'est à dire $u_h = \sum_{j=1}^{n-1} u_j \varphi_j$. En choisissant successivement $v = \varphi_i$ pour $i = 1, \ldots, n-1$ on obtient le système d'équations

$$\sum_{j=1}^{n-1} u_j a(\varphi_j, \varphi_i) = F(\varphi_i) \quad i = 1 \ldots, n-1$$

En posant $a_{ij} = a(\varphi_j, \varphi_i)$ et $f_i = F(\varphi_i)$ on trouve le problème

$$\begin{cases} \text{Trouver } \mathbf{u_h} \in \mathbb{R}^{n-2} \text{ tel que} \\ A\mathbf{u_h} = \mathbf{f} \end{cases}$$

avec $A = (a_{ij})$, $\mathbf{u_h} = (u_1, \ldots, u_{n-1})^T$ et $\mathbf{f} = (f_1, \ldots, f_{n-1})^T$.

4.2 Résolution du problème

Il suffit de résoudre le système linéaire $A\mathbf{u_h} = \mathbf{f}$. Intéressons nous plus particulièrement au calcul de la matrice A.
On part de la définition des φ_i et on obtient

$$\varphi_i'(x) = \begin{cases} \frac{1}{h} & \text{si } x_{i-1} \leq x \leq x_i \\ -\frac{1}{h} & \text{si } x_i \leq x \leq x_{i+1} \\ 0 & \text{sinon} \end{cases}$$

A partir de là, on obtient par calculs

$$a(\varphi_j, \varphi_j) = \int_a^b (\varphi_j'(x))^2 = \int_{x_{j-1}}^{x_{j+1}} \frac{1}{h^2} = \frac{2}{h},$$

et pour $j = 1, \ldots, n-2$

$$\begin{aligned} a(\varphi_j, \varphi_{j+1}) &= \int_a^b \varphi_j'(x)\varphi_{j+1}' \\ &= \int_{x_j}^{x_{j+1}} -\frac{1}{h^2} \\ &= -\frac{1}{h} \\ a(\varphi_{j+1}, \varphi_j) &= a(\varphi_j, \varphi_{j+1}) \\ &= -\frac{1}{h} \end{aligned}$$

Ainsi A est une matrice tridiagonale.

$$A = \begin{pmatrix} \frac{2}{h} & -\frac{1}{h} & & 0 \\ -\frac{1}{h} & \ddots & \ddots & \\ & \ddots & & -\frac{1}{h} \\ 0 & & -\frac{1}{h} & \frac{2}{h} \end{pmatrix}$$

5 Problème bidimensionnel

5.1 Formulation d'un problème aux limites (Dirichlet)

Soit le domaine $\Omega = (0,1)^2$ et considérons la fonction $u(x,y) = \sin(2\pi x)\sin(2\pi y)$. Nous aimerions formuler le problème au limite

$$\begin{cases} -\Delta u &= f & \text{dans } \Omega \\ u &= g & \text{sur } \partial\Omega \end{cases}$$

Pour cela nous allons déterminer les expressions pour f et g dans $L^2(\Omega)$. La fonction f s'obtient par calcul direct

$$f(x,y) = -\Delta u = 8\pi^2 \sin(2\pi x)\sin(2\pi y).$$

Pour trouver g, il suffit de remarquer que $u(x,y)_{|\partial\Omega} = 0$. D'où $g \equiv 0$.

5.2 Formulation d'un problème aux limites (mixtes)

Soit le domaine $\Omega = (0,1)^2$ et la fonction u définis au paragraphe 5.1. On souhaite cette fois-ci définir le problème

$$\begin{cases} -\Delta u &= f & \text{dans } \Omega \\ u &= g_1 & \text{sur } \Gamma_D \\ \frac{\partial u}{\partial \nu} &= g_2 & \text{sur } \Gamma_N \end{cases}$$

avec ν le vecteur normal extérieur à $\partial\Omega$ et

$$\begin{aligned} \Gamma_D &= [0,1]\times\{0\} \cup [0,1]\times\{1\} \\ \Gamma_N &= \{0\}\times(0,1) \cup \{1\}\times(0,1) \end{aligned}$$

Pour cela on doit trouver f,g_1 et g_2 dans $L^2(\Omega)$. Comme pour le paragraphe 5.1, on trouve $f(x,y) = 8\pi^2\sin(2\pi x)\sin(2\pi y)$ et $g_1 = 0$ (car $u(x,y)_{|\Gamma_D} = 0$). Il nous reste à déterminer g_2.

Par définition $\frac{\partial u}{\partial \nu} = \nabla u \cdot \nu$. Ainsi en posant

$$\begin{aligned} \Gamma_{N_1} &= \{0\}\times(0,1), \\ \Gamma_{N_2} &= \{1\}\times(0,1), \end{aligned}$$

on obtient

$$\nabla u = \begin{pmatrix} 2\pi\cos(2\pi x)\sin(2\pi y) \\ 2\pi\sin(2\pi x)\cos(2\pi y) \end{pmatrix}$$

$$\nu_{N_1} = (-1,0) \quad \nu_{N_2} = (1,0)$$

et donc

$$g_2(y) = \begin{cases} -2\pi\sin(2\pi y) & \text{sur } \Gamma_{N_1} \\ 2\pi\sin(2\pi y) & \text{sur } \Gamma_{N_2} \end{cases}$$

5.3 Forme variationnelle

En considérant $H^1_D = \left\{v \in H^1 \mid v = 0 \text{ sur } \Gamma_D\right\}$ et en appliquant la démarche du paragraphe 3.3 on trouve le problème

$$\begin{cases} \text{Trouver } u \in H^1_D \text{ tel que} \\ \displaystyle\int_\Omega \nabla u \nabla v \quad = \quad \int_\Omega fv + \int_{\Gamma_N} g_2 v \end{cases}$$

Définissons la forme bilinéaire

$$a(u,v) = \int_\Omega \nabla u \nabla v$$

et la fonctionnelle

$$F(v) = \int_\Omega fv + \int_{\Gamma_N} g_2 v$$

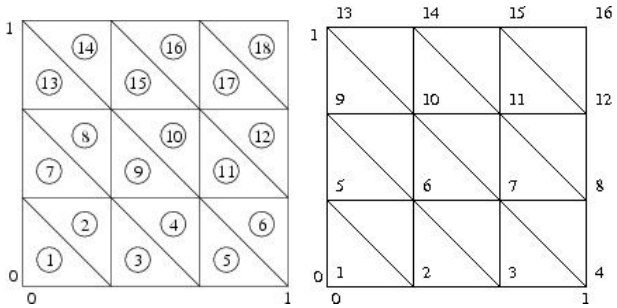
pour obtenir

$$\begin{cases} \text{Trouver } u \in H^1_D(\Omega) \text{ tel que} \\ a(u,v) \quad = \quad F(v) \quad \forall v \in H^1_D \end{cases}$$

Grâce au paragraphe 3.4, on sait que ce problème admet une solution unique.

5.4 Approximation par éléments finis

Considérons τ_h le maillage qui subdivise le carré $[0,1]^2$ en m^2 carrés de largeur $h = \frac{1}{m}$ (voir fig. 2). Ce dernier est composé de simplexes K_i, $i = 1, \ldots, 2m^2$ que l'on numérote par ligne (voir fig. 2(a)). On numérote ensuite chacun des noeuds par ligne en commençant par le coin en bas à gauche.



(a) (b)

FIG. 2 – Maillage de Ω pour $m = 3$

Afin de simplifier nos calculs, nous allons utiliser un simplexe de référence $\hat{K}$ dont les noeuds sont $\hat{a}_1 = (0,0)$, $\hat{a}_2 = (1,0)$ et $\hat{a}_3 = (0,1)$ (voir fig. 3).

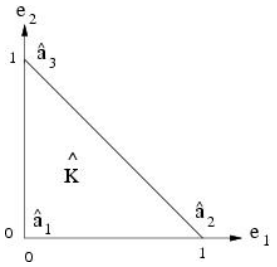


FIG. 3 – Simplexe de référence

Pour chacun de ces noeuds, on peut introduire une fonction de base $\hat{\varphi}_i$ satisfaisant la propriété d'interpolation de Lagrange

$$\hat{\varphi}_i(\hat{a}_j) = \delta_{ij}$$

En utilisant cette propriété, on obtient les trois fonctions de base $\varphi_i : \hat{K} \rightarrow [0,1]$ $(i = 1,2,3)$ telles que pour tout $x = (x_1, x_2) \in \hat{K}$

$$\begin{aligned} \varphi_1(x) &= 1 - x_1 - x_2 \\ \varphi_2(x) &= x_1 \\ \varphi_3(x) &= x_2 \end{aligned}$$

On introduit ensuite les transformations affines T_K avec $K \in \tau_h$ qui envoient $\hat{K}$ sur le simplexe K du maillage.
On introduit l'espace des éléments finis

$$X_h^r = \left\{ v_h \in C^0(\overline{\Omega}) \mid v_{h|K} \in \mathbb{P}^r, \forall K \in \tau_h \right\}$$

Soit maintenant $V_h \subset H_D^1(\Omega)$ définit par

$$V_h = \left\{ v_h \in X_h^1 \mid v_{h|\Gamma_D} = 0 \right\}.$$

Une base possible pour cet espace est $\{\varphi_1, \ldots, \varphi_{(m+1)^2}\}$ où les φ_i sont obtenus grâce à $\{\hat{\varphi}_1, \hat{\varphi}_2, \hat{\varphi}_3\}$ (base de l'élément de référence) via les transformations T_K avec $K \in \tau_h$.

A l'aide de cette base, on peut procéder comme au paragraphe 4.1 en exprimant l'approximation u_h dans celle-ci et en posant $v = \varphi_i$. On obtient alors le système

$$\left\{ \begin{array}{l} \text{Trouver } \mathbf{u_h} \in \mathbb{R}^{(m-1)(m+1)} \text{ tel que} \\ A\mathbf{u_h} \quad = \quad \mathbf{f} \end{array} \right.$$

avec $A = (a_{ij})$ où $a_{ij} = a(\varphi_j, \varphi_i)$ et $\mathbf{f} = (f_i)$ où $f_i = F(\varphi_i)$.

Il s'agit donc de résoudre le système linéaire $A\mathbf{u_h} = \mathbf{f}$ pour obtenir la solution.

5.5 Résolution du problème avec le solveur `Matlab`

A l'aide du solveur décrit dans le paragraphe 7 on peut obtenir la solution du problème posé au paragraphe 5.1.

On commence par définir les fonctions f et g (voir source 1).

```
f=inline('8*pi*pi*sin(2*pi*x)*sin(2*pi*y)');
g=inline('x*2*pi*sin(2*pi*y)-(1-x)*2*pi*sin(2*pi*y)');
```

Source 1 – Définition des fonctions f et g

Ensuite on génère un maillage régulier avec la fonction `RegularMesh` en spécifiant la finesse du maillage souhaitée, i.e. le nombre d'éléments m sur le bord. Par exemple pour réalisé le maillage décrit dans l'exemple 7.1.1, nous avons posé $m = 3$ (voir source 2).

```
Mesh = regularMesh(3);
```

Source 2 – Construction du maillage

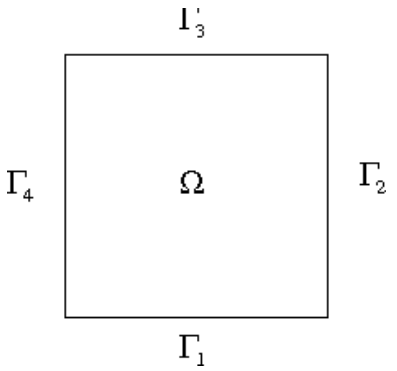


FIG. 4 – Identification des bords

Puis on définit les conditions au bord via la variable `bc` en tenant compte des identifiants des bords de la figure 4 (voir source 3).

```
bc.dirichlet=[1,3];
bc.neumann=[2,4];
```

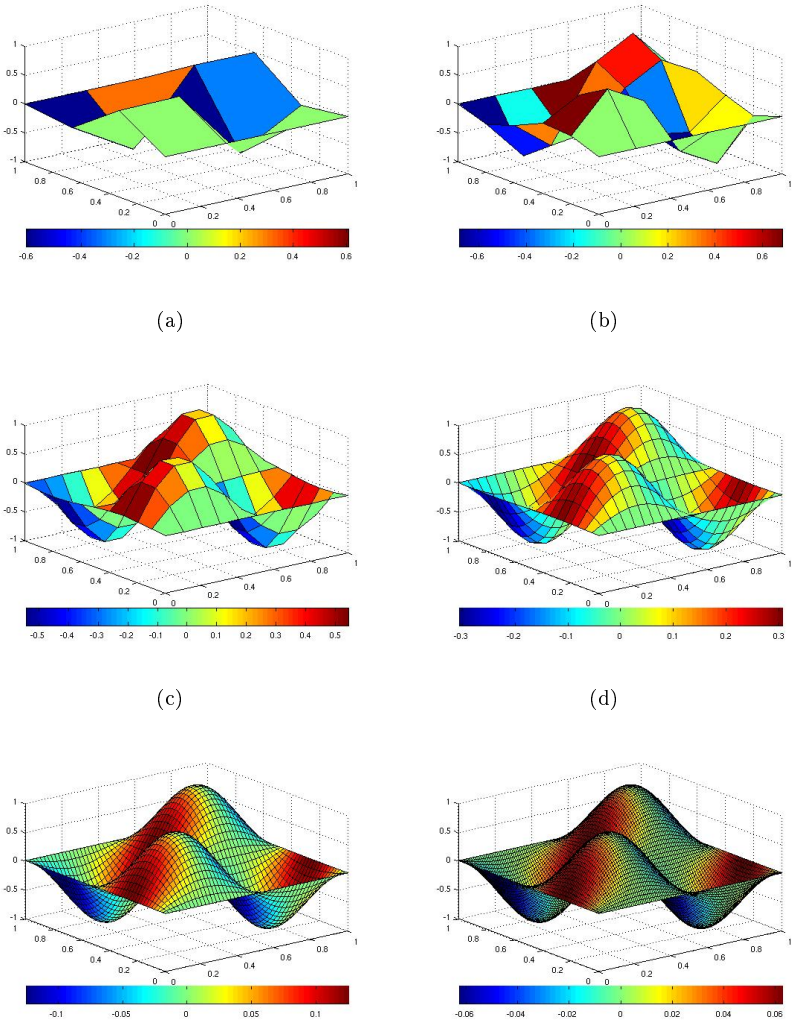
Source 3 – Initialisation de la variable `bc`

Il suffit alors de résoudre le système grâce au solveur `solver_2D` et d'afficher la solution avec la fonction `plotSolution` (voir source 4).

```
u=solver_2d(Mesh,bc,f,g);
plotSolution(Mesh,u);
```

Source 4 – Résolution du système et affichage de la solution

La figure 5 montre la solution obtenue avec différentes valeurs pour m. Comme nous le voyons, la solution s'affine à mesure que m devient grand. Nous allons nous intéresser à l'erreur commise lors de l'approximation numérique du problème.



(a) (b)

(c) (d)

(e) (f)

FIG. 5 – Graphes de la solution obtenue pour $m = 3$ (a), $m = 5$ (b), $m = 10$ (c), $m = 20$ (d), $m = 50$ (e) et $m = 100$ (f).

5.6 Calcul de l'erreur

De par la construction du problème que nous avons effectuée au paragraphe 5.2, nous savons que la solution exacte du problème est donnée par

$$u(x, y) = \sin(2\pi x) \sin(2\pi y).$$

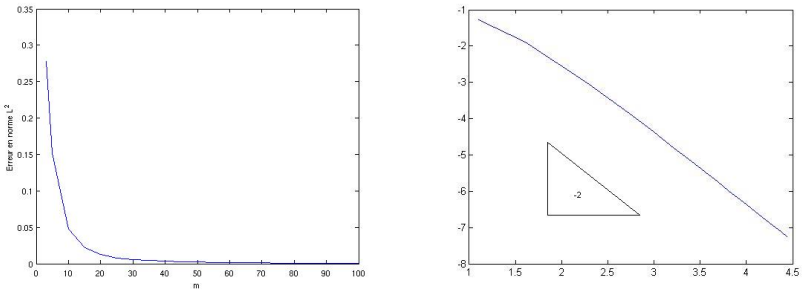
Nous pouvons donc définir la fonction `uexact` de la solution exacte et la passer en argument à notre solveur pour obtenir l'erreur en norme $L^2(\Omega)$. La source 5 montre comment obtenir un graphe de l'erreur en fonction de m.

```
%On definit les valeurs de m souhaitees.
m=[3,5:5:100];
E=zeros(1,length(m));
for i=1:length(m)
  %Construction du maillage
  Mesh = regularMesh(m(i));
  %Resolution du systeme et calcul de l'erreur
  [u E(i)]=solver_2d(Mesh,bc,f,g,uexact);
end

%Affichage du graphe
plot(m,E);
xlabel('m');
ylabel('Erreur en norme L^2');
```

Source 5 – Génération du graphe de l'erreur en fonction de m

La figure 6 montre la réduction de l'erreur en fonction de m pour notre problème.



(a) (b)

FIG. 6 – Graphe de l'erreur en norme L^2 en fonction de m.

Vérifions que ceci correspond bien au résultat théorique.

5.6.1 Théorème. *Soit le problème elliptique suivant*

$$\begin{cases} \text{Trouver } u \in V \text{ tel que} \\ a(u,v) \quad = \quad F(v) \quad \forall v \in V \end{cases}$$

Soit u_h une approximation de Galerkin de u obtenu en considérant des polynômes de degré r. Alors si $u \in H^{p+1}(\Omega)$, $p > 0$, alors

$$\|u - u_h\|_{L^2(\Omega)} \leq Ch^{s+1} |u|_{H^{s+1}(\Omega)}$$

avec $s = \min(r, p)$ et C indépendant de h et u.

Dans notre cas, nous cherchons $u \in H^1(\Omega)$. En fait cette solution est telle que $u \in H^2(\Omega)$ $(p = 1)$ et nous utilisons des polynômes de degré 1. De plus grâce à la structure particulière de notre maillage (i.e. régulier et structuré),

nous obtenons $h = \max_{K \in \tau_h} \text{diam}(K) = \frac{\sqrt{2}}{m}$. Notre approximation devient donc grâce au théoreme :

$$\|u - u_h\|_{L^2(\Omega)} \leq C \frac{2}{m^2} |u|_{H^2(\Omega)}$$

Ainsi on voit que l'erreur va décroitre proportionnelement à m^2. Donc en échelle logarithmique nous devrions avoir une pente d'environ -2 ce qui correspond à ce que l'on voit sur la figure 6(b).

6 Calcul des lignes de courant d'un fluide incompressible en deux dimensions

6.1 Sur les lignes de courant

Comme annoncé dans l'introduction, nous allons nous intéresser au calcul des lignes de courant de l'écoulement d'un fluide incompressible.

6.1.1 Définition (Lignes de courant)**.** *Soit $\vec{u}$ le champ des vecteurs de la vitesse. Une ligne de courant $\vec{c}(t)$ est un chemin tel que*

$$(\vec{c}(t))' = \vec{u}(c(t))$$

où t est le temps.

6.1.2 Définition (Fonction de courant)**.** *Soit $\vec{u} = (u_x, u_y, 0)$ le vecteur vitesse et $\vec{\Psi} = (0, 0, \psi)$ où ψ dépend de la position (x, y). On appelle $\vec{\Psi}$ fonction de courant si, par définition :*

$$\vec{u} = \nabla \times \vec{\Psi}. \tag{1}$$

6.1.3 Remarque. *Dans la dynamique des fluides, on considère l'équation dite* de continuité

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0.$$

où ρ est la densité, $\vec{u}$ la vitesse fluide et t le temps. Cette dernière représente la conservation de la masse (i.e. la masse qui entre dans le système est égale à la masse qui en ressort).

Dans le cadre d'un fluide incompressible, la densité ρ est constante, donc l'équation peut être réécrite comme suit

$$\nabla \cdot \vec{u} = 0.$$

L'équation 1 est cohérente avec léquation de la continuité. En effet

$$\vec{u} = \nabla \times \vec{\Psi} \Rightarrow u_x = \frac{\partial \psi}{\partial y}, u_y = -\frac{\partial \psi}{\partial x}$$

et donc

$$\begin{aligned} \nabla \cdot \vec{u} &= \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \\ &= \frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial^2 \psi}{\partial y \partial x} \\ &= 0. \end{aligned}$$

6.1.4 Propriété. *Les fonctions de courant sont constantes le long des lignes de courant.*

Démonstration. Soient $\vec{u} = (u_x, u_y)$ le champ de vitesse du fluide, $\vec{\Psi} = (0, 0, \psi)$ une fonction de courant et $\vec{c} = \vec{c}(t)$ une ligne de courant. Nous devons vérifier que $\frac{\partial \psi}{\partial \vec{c}} = 0$. Par définition on a $\vec{u} = \nabla \times \vec{\Phi}$. Or ceci est équivalent à

$$u_x = \frac{\partial \psi}{\partial y} \qquad u_y = -\frac{\partial \psi}{\partial x}$$

Donc

$$\begin{aligned} \frac{d\psi}{d\vec{c}} &= \frac{\partial \psi}{\partial \vec{c}} \frac{d\vec{c}}{dt} \\ &= \nabla\psi(\vec{c}) \cdot (\vec{c})' \\ &= \nabla\psi(\vec{c}) \cdot \vec{u}(\vec{c}) \\ &= u_x(\vec{c}) \frac{\partial \psi}{\partial x}(\vec{c}) + u_y(\vec{c}) \frac{\partial \psi}{\partial y}(\vec{c}) \\ &= -u_x(\vec{c}) u_y(\vec{c}) + u_x(\vec{c}) u_y(\vec{c}) \\ &= 0. \end{aligned}$$

□

Cette propriété montre donc que les fonctions de courant sont un outil pour déterminer les lignes équipotentielles. Il nous manque encore une stratégie pour calculer de telles fonctions. Or il suffit de remarquer que

$$\Delta\psi = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} = -\omega$$

où $\vec{\omega} = (0, 0, \omega)^T = \nabla \times \vec{u}$. De plus

$$\frac{\partial \psi}{\partial \nu} = \nabla\psi \cdot \nu = \begin{pmatrix} -u_y \\ u_x \end{pmatrix} \begin{pmatrix} \nu_x \\ \nu_y \end{pmatrix} = -u_y \nu_x + u_x \nu_y.$$

Donc pour trouver les fonctions de courant, il suffit de résoudre le problème

$$\begin{cases} -\Delta\psi &= \omega & \text{dans } \Omega \\ \frac{\partial \psi}{\partial \nu} &= -u_y \nu_x + u_x \nu_y & \text{sur } \partial\Omega \end{cases}$$

6.2 Problème de calcul de lignes de courant

Soit $\Omega \subset \mathbb{R}^n$. On considère le problème

$$\begin{cases} \text{Trouver } \psi \in V \text{ tel que} \\ \displaystyle\int_\Omega \nabla\psi \nabla v &= \displaystyle\int_\Omega \left(\frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} \right) v \\ & \quad + \displaystyle\int_{\partial\Omega} (-u_y \nu_x + u_x \nu_y) v \qquad \forall v \in V. \end{cases}$$

où $\underline{\vec{u}} = (u_x, u_y)$ est *connu*, V est à déterminer et $\nu = (\nu_x, \nu_y)$ est la normale exterieure au domaine. Ainsi le but est de trouver ψ et non pas la vitesse $\underline{\vec{u}}$!

On souhaite retrouver la forme forte du problème et en particulier connaître le type de conditions au bord (Dirichlet, Neumann ou mixte).

6.2.1 Formulation variationnelle

Dans le paragraphe 3.2, nous avions un problème de la forme

$$\left\{ \begin{array}{l} \text{Trouver } \psi \in V \text{ tel que} \\ \displaystyle\int_\Omega \nabla\psi\nabla v \quad = \quad \int_\Omega fv + \int_{\partial\Omega} gv. \end{array} \right.$$

Par identification avec notre problème de départ on déduit que $V = H^1(\Omega)$,

$$f(x) = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} = \omega$$

et

$$g = -u_y\nu_x + u_x\nu_y.$$

Ainsi notre problème sous sa forme forte (avec conditions de Neumann) s'écrit

$$\left\{ \begin{array}{rcll} -\Delta\psi & = & \omega & \text{dans } \Omega \\ \frac{\partial\psi}{\partial\nu} & = & -u_y\nu_x + u_x\nu_y & \text{sur } \partial\Omega \end{array} \right.$$

Par le paragraphe 3.4, on sait que malheureusement notre problème n'admet pas de solution unique ! Par contre si φ est une solution, l'espace des solutions est donné par

$$\{\varphi + c, c \in \mathbb{R}\}.$$

Donc la solution est unique à une constante près. Une façon de remédier à ce problème serait de fixer la valeur de ψ en un point.

6.2.2 Remarque. *Comme nous l'avions vu au paragraphe 5.4, le problème sous sa forme variationnelle peut être ramené à la résolution d'un sytème linéaire du type* $A\mathbf{u_h} = \mathbf{f}$. *Sous* `Matlab`, *la commande pour résoudre un tel système est donnée par la source 6.*

```
uh=A\f;
```

Source 6 – Résolution d'un système linéaire sous `Matlab`

Malheureusement si la solution n'est pas unique, `Matlab` *va retourner un message semblable à celui de la source 7.*

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.369783e-017.
```

Source 7 – Erreur retournée par `Matlab`

Une manière d'éviter ce problème est d'utiliser une méthode itérative comme le gradient conjugué pour obtenir la solution (voir source 8).

```
uh=pcg(A,f);
```

Source 8 – Calcul de la solution avec la méthode du gradient conjugué

6.3 Résolution du problème sous `Matlab`

Nous allons commencer par construire la fonction $f = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}$. Pour ce faire nous avons besoin de récupérer la solution u (via une interpolation) et d'en calculer le rotationnel.

6.3.1 Récupération de la solution

Nous allons utiliser l'ensemble de fichiers contenus dans le dossier `streamlines`. Ce dossier contient un certain nombre de maillages (contenus dans le dossier `mesh`) et des solutions de problèmes (contenues dans le dossier `solutions`).

Les scripts `setup.m` et `plotstream` effectue déjà la récupération de la solution $\vec{u}$ et du maillage. En fait, $\vec{u}$ est une solution des équations de Navier-Stokes calculée au préalable sur un maillage différent. Adaptons ces codes à nos besoins. Pour cela on modifie le contenu de `plot_stream`. On construit une interpolation de $\vec{u}$ sur un maillage régulier de $m+1$ noeuds largeur et de hauteur en utilisant la fonction `myinterpol` (voir source 9).

```
disp('Creating f function');
% grid generation for streamline function
x = 0:1/m:1;
y = 0:1/m:1;

% Calculate vectors in each point
for i_=1:size(x,2),
  for j_=1:size(y,2),
    [U(j_,i_,1),U(j_,i_,2)] = ...
        myInterpol(x(1,i_), y(1,j_),u_neu(1:2:end),u_neu(2:2:end),mesh);
            %interpolation de (x,y)
  end
end
```

Source 9 – Interpolation de $\vec{u}$ sur un maillage de m éléments de côté

6.3.2 Obtenir les conditions au bord

Par le paragraphe 3.1, nous savons que les conditions au bord de Neumann sont

$$\frac{\partial \psi}{\partial \nu} = -u_y \nu_x + u_x \nu_y$$

où $\nu = (\nu_x, \nu_y)$ est la normale extérieure au domaine. Or notre domaine étant carré, il vient que les conditions au bord sont données par :

$$\begin{aligned}
\frac{\partial \psi}{\partial \nu} &= -u_x \quad \text{sur } \Gamma_1 \\
\frac{\partial \psi}{\partial \nu} &= -u_y \quad \text{sur } \Gamma_2 \\
\frac{\partial \psi}{\partial \nu} &= u_x \quad \text{sur } \Gamma_3 \\
\frac{\partial \psi}{\partial \nu} &= u_y \quad \text{sur } \Gamma_4
\end{aligned}$$

Ainsi il suffit de récupérer les valeurs de $\vec{u}$ pour les noeuds aux bords du domaine et d'ajuster le signe si besoin est. Stockons les valeurs en ces noeuds dans le tableau `neucond`.

6.3.3 Calcul du rotationnel

Nous connaissons les valeurs de $\vec{u}$ dans chacun des noeuds du maillage. Explicitons le moyen pour calculer $\nabla \times \vec{u}$.

6.3.4 Définition. *Soit $F : \mathbb{R}^2 \to \mathbb{R}^2$. Le rotationnel de $F = (F_1, F_2)$ est donné par*

$$\nabla \times F := \frac{\partial F_2}{\partial x_1} - \frac{\partial F_1}{\partial x_2}$$

Nous souhaitons calculer le rotationnel de notre vitesse fluide $\vec{u}$. Ainsi il est clair que nous allons devoir approcher les dérivées. Pour cela nous allons utiliser la méthode des différences finis (DF) centrées pour l'intérieur du domaine et une autre méthode sur le bord.

Rappelons brièvement ces méthodes. Soit f une fonction continue et considérons un certain nombre de points $x_1, \ldots, x_n$ tels que $x_i - x_{i-1} = h$. Alors la dérivée en ces points est obtenues des manières suivantes :

1. DF centrées : $f'(x_i) \cong \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$,
2. $f'(x_i) \cong \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h}$,
3. $f'(x_i) \cong \frac{f(x_i) - 4f(x_{i-1}) + 3f(x_{i-2})}{2h}$.

6.3.5 Remarque. *Pour calculer f' en n points, nous avons besoin de $n + 2$ points. De plus ces trois méthodes sont d'ordre 2.*

Revenons au calcul du rotationnel pour notre vitesse fluide $\vec{u} = (u_x, u_y)$. Nous connaissons les valeurs des composantes u_x et u_y en différents noeuds et souhaitons calculer $\frac{\partial u_y}{\partial x}$ et $\frac{\partial u_x}{\partial y}$.

Pour obtenir $\frac{\partial u_y}{\partial x}$, nous allons choisir quelle méthode utiliser en fonction de la localisation du point dans lequel nous voulons connaitre la dérivée.

Nous allons utiliser les DF centrées sur tout les points ne se trouvant pas sur les bords gauche et droit du domaine. Ces points sont marqués par des puces sur la figure 7. Pour le bord gauche, nous utiliserons la seconde méthode et pour le bord droit la troisième (représentées par des triangles).

Le calcul de $\frac{\partial u_x}{\partial y}$ est éffectué de manière analogue. On stocke les valeurs de $\nabla \times \vec{u}$ pour les différents noeuds du maillage dans le tableau `rotu`.

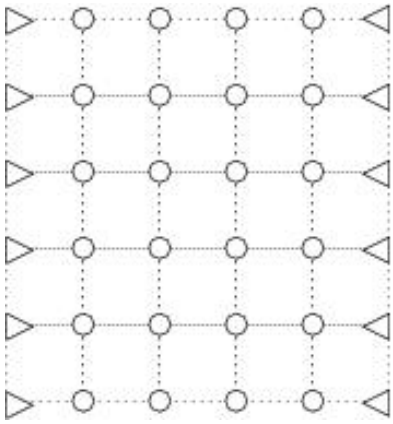


FIG. 7 – Schéma de l'utilisation des méthodes de différences finies

6.3.6 Utilisation du solveur

Comme pour le paragraphe 5.5, nous devons construire une structure `Mesh` et définir les conditions au bord via la variable `bc` (voir source 10).

```
Mesh = regularMesh(m); %Construction du maillage
bc.dirichlet=[0]; %Conditions au bord
bc.neumann=[1,2,3,4];
```

Source 10 – Initialisation du maillage et des labels des conditions au bord

Il nous faut encore définir, les fonctions f et g du problème. En effet nous ne pouvons pas utiliser directement les tableaux `neucond` et `rotu` car le solveur requière que f et g soit des fonctions *inline*.

Malheureusement les fonctions *inline* ne peuvent pas appeler les deux variables citées ci-dessus. Il faut donc créer des fonctions dans des fichiers séparés permettant d'accéder à de telles données, disons `F` et `fneu`. Il suffit alors de construire les fonctions *inline* comme indiqué dans la figure 11

```
g=inline('fneu(x,y)');
f=inline('F(x,y)');
```

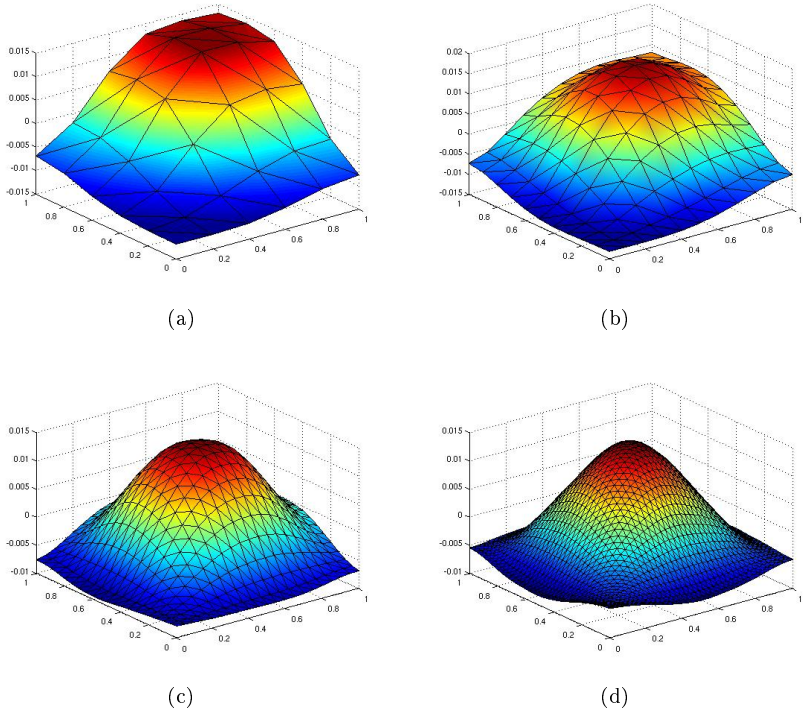
Source 11 – Définition des fonctions f et g pour le solveur

Nous avons alors tous les éléments nécessaires pour utiliser le solveur. Il suffit alors de procéder comme dans le paragraphe 5.5 (voir source 12).

```
phi=solver_2d(Mesh,bc,f,g);
```

Source 12 – Calcul de la solution φ

En affinant le maillage (i.e. en variant le nombre m d'éléments sur le bord), on obtient une solution de plus en plus précise comme en témoigne la figure 8.



(a) (b)

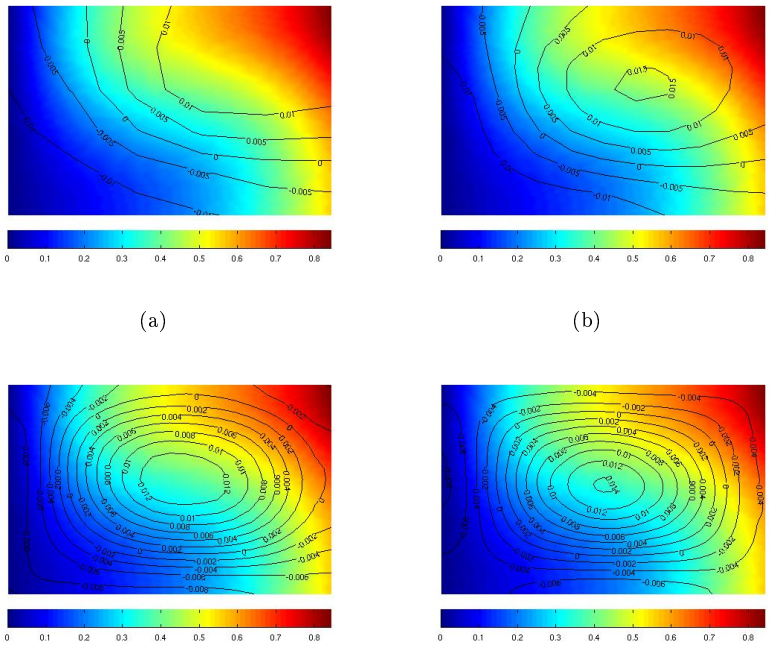
(c) (d)

FIG. 8 – Graphes de la solution obtenue pour $m = 5$ (a), $m = 10$ (b), $m = 20$ (c), $m = 40$ (d).

6.3.7 Affichage des lignes de courant

Dans le paragraphe 6.3.6, nous avons obtenu la fonction de courant ψ. Or comme nous l'avons vue précédemment grâce à la propriété 6.1.4, les lignes équipotentielles de la fonction ψ correspondent aux lignes de courant du fluide.

`Matlab` fournit la fonction `contour` qui permet l'affichage de lignes équipotentielles. La figure 9 donne les lignes de courant obtenues à partir des solutions de la figure 8.



(a) (b)

(c) (d)

FIG. 9 – Graphes des lignes de courants obtenues pour la solution calculée pour $m = 5$ (a), $m = 10$ (b), $m = 20$ (c), $m = 40$ (d).

7 Construction d'un solveur

Dans cette section nous allons nous intéresser à l'implémentation d'un solveur dans `Matlab` afin de pouvoir résoudre les problèmes du type $\Delta u = f$ avec des conditions au bord de Dirichlet homogènes et/ou de Neumann.

7.1 Représentation du maillage

Un aspect essentiel du solveur est la représentation du maillage dans le programme. L'objectif est de pouvoir utiliser n'importe quel maillage, qu'il soit structuré ou non.

D'une manière conceptuelle, le maillage peut être vu comme un ensemble de noeuds reliés entre eux de manière à former des triangles. Pour transposer cet assemblage sous `Matlab`, nous allons créer la structure `Mesh` contenant les variables suivantes :

- `Mesh.Nn` : Nombre de noeuds
- `Mesh.Ne` : Nombre d'éléments
- `Mesh.Nef` : Nombre d'éléments sur le bord
- `Mesh.Ndf` : Nombre de degrés de liberté des éléments
- `Mesh.Ndff` : Nombre de degrés de liberté des éléments sur le bord
- `Mesh.d` : Dimension du problème.

- `Mesh.nodes` :
  Contient la liste des coordonnées des noeuds.
- `Mesh.elements` :
  Contient la description des éléments du maillage. Chaque colonne contient la liste des numéros des noeuds contenus dans l'élément.
- `Mesh.elements_f` :
  Contient la description des éléments du bord du maillage. Chaque colonne contient la liste des numéros des noeuds contenus dans l'élément.
- `Mesh.borderFlags` :
  Contient le numéro du bord sur lequel se trouve le noeud.
- `Mesh.borderFlagsNb` :
  Nombre de bords du problème.

7.1.1 Exemple

Supposons que l'on souhaite construire le maillage de la figure 2. Alors les variables de la structure valent

- `Mesh.Nn` : 16
- `Mesh.Ne` : 18
- `Mesh.Nef` : 12
- `Mesh.Ndf` : 3
- `Mesh.Ndff` : 2
- `Mesh.d` : 2
- `Mesh.nodes` :

$$\begin{bmatrix} 0 & \frac{1}{3} & \frac{2}{3} & 1 & 0 & \frac{1}{3} & \frac{2}{3} & 1 & 0 & \frac{1}{3} & \frac{2}{3} & 1 & 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & 1 & 1 & 1 & 1 \end{bmatrix}$$

- `Mesh.elements` :

$$\begin{bmatrix} 1 & 6 & 2 & 7 & 3 & 8 & 5 & 10 & 6 & 11 & 7 & 12 & 9 & 14 & 10 & 15 & 11 & 16 \\ 2 & 5 & 3 & 6 & 4 & 7 & 6 & 9 & 7 & 10 & 8 & 11 & 10 & 13 & 11 & 14 & 12 & 15 \\ 5 & 2 & 6 & 3 & 7 & 4 & 9 & 6 & 10 & 7 & 11 & 8 & 13 & 10 & 14 & 11 & 15 & 12 \end{bmatrix}$$

- `Mesh.elements_f` :

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 8 & 12 & 16 & 15 & 14 & 13 & 9 & 5 \\ 2 & 3 & 4 & 8 & 12 & 16 & 15 & 14 & 13 & 9 & 5 & 1 \end{bmatrix}$$

- `Mesh.borderFlags` : $[1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]$
- `Mesh.borderFlagsNb` : 4

7.2 Implémentation des fonctions de base pour l'élément de référence

Comme vu précédemment, l'ensemble des calculs nécessaires pour obtenir la matrice A et le vecteur f peuvent être ramenés au simplexe de référence $\hat{K}$. Il convient donc de n'implémenter que les fonctions de base de $\hat{K}$ et leurs dérivées (voir les sources 14 et 15).

De même, il faut définir les fonctions de base φ_1^f et φ_2^f pour l'élément de référence sur le bord $\hat{K}_f$ (voir les sources 16 et 17). Ces dernières sont définies

$\forall x \in [0,1]$ de la manière suivante :

$$\begin{aligned} \varphi_1^f(x) &= 1-x \\ \varphi_2^f(x) &= x \end{aligned}$$

7.3 Construction de A et f

7.3.1 Implémentation des points et des poids de Gauss sous `Matlab`

Notre objectif est d'implémenter la théorie du paragraphe 2.2 pour calculer A et f. Le calcul des polynômes de Legendre est crucial pour pouvoir calculer les points et les poids de Gauss. La première chose à faire est donc de créer la fonction `polyLegendre` permettant de les obtenir sous `Matlab`.

Commençons par remarquer que `Matlab` stocke les polynômes sous forme de vecteur contenant les coefficients de chacune des puissances. Ainsi l'addition et la soustraction de polynômes revient à effectuer ces opérations pour les vecteurs. Par contre le produit de polynômes est obtenus via la fonction `conv(`p`,`q`)` qui effectue la convolution entre p et q.

Ensuite il suffit d'utiliser ces opérations pour implémenter un algorithme de récursion basé sur le lemme 2.2.5 (voir source 18).

Il faut encore construire la fonction `getPointsAndWeights` qui permet d'obtenir les points et les poids de Gauss (voir source 19).

7.3.2 Utilisation de la quadrature de Gauss

Afin de construire la matrice A et le membre de droite F, nous allons devoir intégrer certaines fonctions sur notre domaine Ω. Or ce domaine peut être vu comme la somme des éléments du maillage. Ainsi pour $\Phi : \Omega \to \mathbb{R}$ nous avons que

$$\int_\Omega \Phi(x) = \sum_{K \in \tau_h} \int_K \Phi(x)$$

Nous pouvons également nous ramener au simplexe de référence $\hat{K}$ pour chacun des $K \in \tau_h$ par le changement de variable $x = T_K(\hat{x})$.

$$\int_K \Phi(x) = \int_{\hat{K}} \Phi(T_k(\hat{x})) \det(J_k(\hat{x}))$$

où $J_k(\hat{x}) = \frac{\partial T_k(\hat{x})}{\partial \hat{x}}$ est la matrice jacobienne de la transformation T_K.

$$\begin{aligned} \int_K \Phi(x) &\cong \sum_{l=1}^{n} \Phi(T_k(\hat{\xi}_l)) \hat{w}_l \det(J_k(\hat{\xi}_l)) \\ &\cong \sum_{l=1}^{n} \Phi(\xi_l) w_l \end{aligned}$$

où n est le nombre de points de Gauss, $w_l = \hat{w}_l \det(J_k(\hat{\xi}_l))$ et $\xi_l = T_k(\hat{\xi}_l)$. Remarquons que nous n'avons utilisé qu'un seul poids contrairement à ce que nous avions fait pour le paragraphe 2.2.2. Nous avons simplement utilisé des noeuds et des poids spécialement prévu pour les simplexes (voir remarque 2.2.7). Nous utiliserons d'ailleurs ces noeuds et ces poids pour la suite.

7.3.3 Calcul de la matrice A

Commençons par calculer les coéfficients a_{ij} de la matrice A. Considérons $\{\varphi_1, \dots \varphi_N\}$ la base définie au paragraphe 5.4. En posant $\Phi(x) = \nabla\varphi_i(x)\nabla\varphi_j(x)$ et en utilisant ce que l'on vient de voir, on obtient

$$\begin{aligned} a(\varphi_j, \varphi_i) &= \int_\Omega \nabla\varphi_j(x)\nabla\varphi_i(x)dx \\ &= \sum_{K\in\tau_h} \int_K \nabla\varphi_j(x)\nabla\varphi_i(x)dx \\ &\cong \sum_{K\in\tau_h} \sum_{l=1}^{n} \nabla\varphi_j(\xi_l)\nabla\varphi_i(\xi_l)w_l \end{aligned}$$

On peut remarquer que $\varphi_i(\xi_l) = \hat{\varphi}_i(\hat{\xi}_l)$. Ainsi

$$a(\varphi_j, \varphi_i) \cong \sum_{K\in\tau_h} \sum_{l=1}^{n} \nabla\hat{\varphi}_j(\hat{\xi}_l)\nabla\hat{\varphi}_i(\hat{\xi}_l)w_l$$

7.3.4 Calcul de f

Pour le membre de droite, nous devons calculer

$$F(\varphi_i) = \int_\Omega f\varphi_i + \int_{\Gamma_N} g_2\varphi_i$$

Posons $\Phi^1 = f$ et $\Phi^2 = g_2$. Par ce qui précède, on obtient

$$\begin{aligned} \int_\Omega \Phi^1(x)\varphi_i(x)dx &= \sum_{K\in\tau_h} \int_K \Phi^1(x)\varphi_i(x)dx \\ &\cong \sum_{K\in\tau_h} \sum_{l=1}^{n} \Phi^1(\xi_l)\varphi_i(\xi_l)w_l \end{aligned}$$

Comme $\varphi_i(\xi_l) = \hat{\varphi}_i(\hat{\xi}_l)$, il vient que

$$\int_\Omega \Phi^1(x)\varphi_i(x)dx \cong \sum_{K\in\tau_h} \sum_{l=1}^{n} \Phi^1(\xi_l)\hat{\varphi}_i(\hat{\xi}_l)w_l.$$

Nous devons encore calculer

$$\int_{\partial\Omega} \Phi^2(x)\varphi_i(x)dx$$

Pour ceci on va considérer S_h les faces sur les bord de notre maillage. Nous allons utiliser le segment $\hat{K}^f = [0, 1]$ comme élément de référence. Afin de faire le lien entre cette élément et le maillage, on introduit les transformations T_{K^f} qui envoie le segment de référence $\hat{K}^f$ sur l'élément K^f de S_h.

En procédant de manière analogue à ce qui précède, on obtient

$$\begin{aligned}
\int_{\partial\Omega} \Phi^2(x)\varphi_i(x)dx &= \sum_{K^f\in S_h} \int_{K^f} \Phi^2(x)\varphi_i(x)dx \\
&= \sum_{K^f\in S_h} \int_{\hat{K}^f} \Phi^2(T_{K^f}(\hat{x}))\varphi_i(T_{K^f}(\hat{x}))\det(J_{K^f}(\hat{x}))d\hat{x} \\
&= \sum_{K^f\in S_h} \int_{\hat{K}^f} \Phi^2(T_{K^f}(\hat{x}))\hat{\varphi}_i(\hat{x})\det(J_{K^f}(\hat{x}))d\hat{x} \\
&\cong \sum_{K^f\in S_h} \sum_{l=1}^{n} \hat{w}_l^f \Phi^2(T_{K^f}(\hat{\xi}_l^f))\hat{\varphi}_i(\hat{\xi}_l^f)\det(J_{K^f}(\hat{\xi}_l^f))
\end{aligned}$$

Finalement en posant $w_l^f = \hat{w}_l^f \det(J_{K^f}(\hat{\xi}_l^f))$ et $\xi_l^f = T_{K^f}(\hat{\xi}_l^f)$ on a

$$\int_{\partial\Omega} \Phi^2(x)\varphi_i(x)dx \cong \sum_{K^f\in S_h} \sum_{l=1}^{n} w_l^f \Phi^2(\xi_l^f)\hat{\varphi}_i(\hat{\xi}_l^f)$$

Le membre de droite se calcule donc par

$$F(\varphi_i) \cong \sum_{l=1}^{n} \left(\sum_{K\in\tau_h} w_l \Phi^1(\xi_l)\hat{\varphi}_i(\hat{\xi}_l) + \sum_{K^f\in S_h} w_l^f \Phi^2(\xi_l^f)\hat{\varphi}_i(\hat{\xi}_l^f) \right)$$

7.4 Evaluer une fonction dans un point de Gauss

Les points de Gauss sont en général donnés pour l'élément de référence. Ensuite pour un certain simplexe K, on utilise la transformation affine T_K pour obtenir le point de Gauss dans le maillage global.

Nous allons donner une formule permettant de définir T_K. Soient $\{a_1, a_2, a_3\}$ les noeuds qui définissent le simplexe K, alors

$$T_K(x) = \sum_{l=1}^{3} a_l \hat{\varphi}_l(x).$$

7.5 Calcul du jacobien

Un point technique de la réalisation du solveur est le calcul de la matrice jacobienne pour la transformation $T_K = (T_K^1, T_K^2)$. Notons la J_{T_K}. Par définition cette matrice est définie par

$$J_{T_K} = \begin{pmatrix} \frac{\partial T_K^1}{\partial x_1} & \frac{\partial T_K^1}{\partial x_2} \\ \frac{\partial T_K^2}{\partial x_1} & \frac{\partial T_K^2}{\partial x_2} \end{pmatrix}.$$

Ainsi en utilisant l'expression du paragraphe 7.4, on obtient

$$(J_{T_K})_{k_1,k_2} = \sum_{l=1}^{3} a_l^{k_1} \frac{\partial \hat{\varphi}_l}{\partial x_{k_2}}.$$

7.6 Optimisations pour le calcul numérique

Comme nous venons de le voir, les fonctions de base de $\hat{K}$ sont à plusieurs reprises évaluées dans les points de Gauss du simplexe référence. Ainsi pour éviter de recalculer ces valeurs à chaque fois nous allons les stocker dans les tableaux `base_ref` et `d_base_ref` pour les fonctions de base et leurs dérivées respectivement. On stocke de même les valeurs des fonctions de base des éléments du bord dans les tableaux `base_ref_f` et `d_base_ref_f`.

De même, les $w_l = \hat{w}_l \det(J_K(\hat{\xi}_l))\ \forall l = 1,2,3\ \forall K \in T_h$ du paragraphe 7.3.2 peuvent être stockés dans le tableau `poids_K` pour éviter de les calculer à chaque fois. Ceci est bien sûr valable pour les jacobiens des transformations des éléments du bords qui seront stockés dans le tableau `poids_K_f`.

7.7 Application des conditions au bord

Nous avons déjà discuté du cas des conditions au bord de Neumann lors du calcul du membre de droite f. Il reste néanmoins à implémenter les conditions de Dirichlet homogènes. Ceci revient à imposer des valeurs pour certains noeuds (pour les conditions de Dirichlet homogènes ces valeurs sont égales zéro).

Considérons le i^{e} noeud du maillage et supposons qu'il faille imposer une condition de Dirichlet homogène sur celui-ci. Ceci signifie que l'on connait déjà la valeur pour la i^{e} valeur de u_h :

$$u_h^i = 0$$

Ceci se traduit de la manière suivante pour le système $Ax = f$:

$$\begin{aligned} A_{ii} &= 1 \\ A_{ij} &= 0 \qquad \forall j \neq i \\ f_i &= 0 \end{aligned}$$

```
for i=1:Mesh.Nef
  if(ismember(Mesh.borderFlags(i),bc.dirichlet)) %Si on est sur le bon
      bord
    for ni=1:Mesh.Ndff
      I=Mesh.elements_f(ni,i);%On attrape le numero du noeud
      A(I,:)=zeros(1,Mesh.Nn);
      A(I,I)=1;
      F(I)=0;
    end
  end
end
```

Source 13 – Application des conditions au bord de Dirichlet homogènes

7.8 Calcul de l'erreur d'approximation

L'erreur commise par l'approximation est une information précieuse car elle permet d'observer si notre approximation converge vers la solution exacte. Soient

u_h une approximation de u. L'erreur en norme L^2 est calculée comme suit

$$E = \|u_h - u\|_{L^2(\Omega)} = \left(\int_\Omega |u_h(x) - u(x)|^2 \, dx \right)^{\frac{1}{2}}.$$

Le calcul de E est effectué en utilisant la quadrature de Gauss comme expliqué paragraphe 7.3.2 en posant $\Phi(x) = |u_h(x) - u(x)|^2$.

7.9 Quelques fonctionnalités supplémentaires

Il est important de fournir à l'utilisateur quelques outils permettant une utilisation plus aisée du solveur. Ce paragraphe propose un tour d'horizon de ces derniers.

7.9.1 plotSolution

Une fois calculée par le solveur, la solution u_h est renvoyée sous forme de vecteur de valeurs dont chacune des composantes est la valeur de la solution dans un noeud. Il est facile de faire le graphe de la solution lorsque le maillage est régulier (en utilisant les fonctions `meshgrid` et `surf`). Par contre cela devient plus technique lorsque le maillage est quelconque.

Pour simplifier l'affichage de la solution on définit la fonction `plotSolution` prenant en argument une structure de maillage (comme celle définit au paragraphe 7.1) et la solution u_h. `Matlab` fournit la fonction `trimesh` qui permet d'afficher une surface à partir d'une triangulation. Il suffit alors d'utiliser cette fonction avec la triangulation du maillage contenue dans la variable `Mesh.elements` (voir la source 23).

7.9.2 RegularMesh

Pour un certain nombre de problèmes simples, un maillage structuré comme celui de la figure 2 est suffisant pour résoudre le problème. Il est donc utile de pouvoir générer un tel maillage. La fonction `RegularMesh` construit un tel maillage en se basant sur le nombre d'éléments m se trouvant sur le bord.

7.9.3 getMeshFromPET

`Matlab` contient le composant `pdetool` qui permet, entre autre, d'obtenir des maillages au format (p, e, t) (i.e. p = points, e = edges, t = triangles). Cette représentation est relativement similaire aux informations de la structure `Mesh` mais n'est pas compatible avec le solveur. La fonction `getMeshFromPET` permet de transformer les données (p, e, t) en une structure `Mesh` valide pour le solveur.

7.9.4 plotMesh

Cette fonction permet d'afficher le maillage à l'écran. Les bords sont affichés dans des couleurs différentes pour pouvoir être identifiés

8 Conclusion

Tout au long de ce projet de semestre nous avons considéré différentes formulations du problème de Poisson. Nous avons ensuite expliqué la théorie nécessaire à la construction d'un solveur pour résoudre des problèmes de la forme

$$-\Delta u = f$$

avec conditions de Dirichlet homogènes, de Neumann ou les deux à la fois. Nous avons volontairement construit ce dernier de manière modulaire afin de pouvoir l'améliorer.

Voisi une liste d'améliorations qui pourraient être apportées lors de futurs projets de semestre.

1. Implémenter une autre base éléments finis pour permettre le calcul de la solution via les polynômes de Lagrange de degré deux (i.e. $\mathbb{P}_2$).
2. Implémenter les conditions au bord de Dirichlet non-homogènes.
3. Généraliser le solveur pour lui permettre de résoudre des problèmes de la forme
   $$\mu u + \vec{\beta} \cdot \nabla u - \nu \Delta u = f$$
   où $\mu, \nu \in \mathbb{R}$, $\vec{\beta} \in \mathbb{R}^2$ et $f \in L^2(\Omega)$.

A Code source

Dossier `base`

```
function r=phi_ref(i,x)
%i-eme fonction de base du simplexe de reference
% ------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------
% Usage : r=phi_ref([1|2|3],[x1,x2])

if(size(x,2)~=2)
  error('x must be a 2-dimensional vector');
end
switch i
  case 1 %phi_1(x)
    r=1-x(1)-x(2);
  case 2 %phi_2(x)
    r=x(1);
  case 3 %phi_3(x)
    r=x(2);
  otherwise
     error('There are only 3 functions in the base');
end
```

Source 14 – `phi_ref.m`

```
function r=d_phi_ref(i,k)
%Derivee de la i-eme fonction de base du simplexe de reference
%par rapport a x_k
% ------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------
% Usage : r=d_phi_ref([1|2|3],[1|2])

if(k>2 || k<=0)
  error('k must be 1 or 2');
end
switch i
  case 1
    r=-1;
  case 2
    if(k==1)
      r=1;
    else
      r=0;
    end
  case 3
    if(k==2)
      r=1;
    else
```

```
     r=0;
   end
  otherwise
     error('There are only 3 functions in the base');
end
```

Source 15 – `d_phi_ref.m`

```
function r=phi_ref_f(i,x)
%i-eme fonction de base du simplexe de reference
% ---------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ---------------------------------------------
% Usage : r=phi_ref_f([1|2],[x])

if(size(x,2)~=1)
  error('x must be a 1-dimensional vector');
end
switch i
  case 1
    r=1-x;
  case 2
    r=x;
  otherwise
     error('There are only 2 functions in the base');
end
```

Source 16 – `phi_ref_f.m`

```
function r=d_phi_ref_f(i)
%Derive de la i-eme fonction de base du simplexe de reference
%par rapport a x_k
% ---------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ---------------------------------------------
% Usage : r=d_phi_ref_f([1|2])

switch i
  case 1
    r=-1;
  case 2
    r=1;
  otherwise
     error('There are only 2 functions in the base');
end
```

Source 17 – `d_phi_ref_f.m`

Dossier integration

```
function p_n=polyLegendre(n)
%Genere le polynome de legendre d'ordre n
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : p_n=polyLegendre(n)

if(n==0)
  p_n=1;
elseif(n==1)
  p_n=[1 0];
else
  p_m=1;
  p_n=[1 0];
  x=p_n;
  for i=2:n
    a=(2*(i-1)+1)*conv(x,p_n);
    b=(i-1)*p_m;
    b=[zeros(1,length(a)-length(b)),b];
    p_m=p_n;
        p_n=(a-b)/i;
  end
end
```

Source 18 – polyLegendre.m

```
function [x,w]=getPointsAndWeights(dim,n)
%Calcul les points et les poids pour
%la quadrature de Gauss
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : [x,w]=getPointsAndWeights([1|2],n)
% Remark:
% 1) If dim=1, n can be any non-negativ integer
% 2) If dim=2, n=[1|2|3|4|6]

%Si la dimension vaut 1 on calcule facilement les points grace a Legendre
if(dim==1)
  %On obtient l'expression polynomial pour les w_i
  p=polyLegendre(n);
  q=polyder(p);

  %On obtient les racines de p
  x=roots(p);
  w=zeros(1,n);
    for i=1:n
        w(i)=2/((1-x(i)^2)*polyval(q,x(i))^2);
    end
```

```
    %On veut des poids pour l'intervalle [0,1]
    w=w/2;
    x=(x+1)/2;
elseif(dim==2)
  S=0.5; %surface du simplexe de reference
  if(n==1)
    x=[1/3,1/39];
    w=S;
  elseif(n==3)
    x=zeros(3,2);
    x(1,:)=[1/6,1/6];
    x(2,:)=[1/6,2/3];
    x(3,:)=[2/3,1/6];
    w=[S/3,S/3,S/3];
  elseif(n==4)
    x=zeros(4,2);
    x(1,:)=[1/3,1/3];
    x(2,:)=[1/5,1/5];
    x(3,:)=[1/5,3/5];
    x(4,:)=[3/5,1/5];
    w=[-9*S/16,25/48*S,25/48*S,25/48*S];
  elseif(n==6)
    a=0.445948490915965;
    b=0.091576213509771;
    x=zeros(6,2);
    x(1,:)=[a,a];
    x(2,:)=[a,1-2*a];
    x(3,:)=[1-2*a,a];
    x(4,:)=[b,b];
    x(5,:)=[b,1-2*b];
    x(6,:)=[1-2*b,b];
    a=S*0.223381589678010;
    b=S*0.109951743655322;
    w=[a,a,a,b,b,b];
  else
    error('points not yet implemented');
  end
else
  error('dimension not yet implemented');
end
```

Source 19 – `getPointsAndWeights.m`

Dossier `mesh`

```
function Mesh=regularMesh(m)
%Construit une structure MESH pour un maillage
%regulier dont le bord est subdivise en m elements
% -----------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
```

```
% ------------------------------------------------
% Usage : Mesh=regularMesh(m)

n = m+1; %Nombre de noeuds par rangee
Mesh.Nn = n^2; %Nombre de noeuds total
Mesh.Ndf = 3;  %Nombre de degres de liberte des simplexes du maillage
Mesh.Ndff = 2; %Nombre de degres de liberte des bords
Mesh.Ne = 2*m^2; %Nombre d'elements
Mesh.Nef = 4*m; %Nombre de faces sur le bord
Mesh.d = 2;  %Dimension du probleme

%Donne les coordonnes du noeud indique
Mesh.nodes = zeros(Mesh.d,Mesh.Ndf);
for i=1:n
  for j=1:n
    Mesh.nodes(1,indic(i,j,n))=(i-1)/m;
    Mesh.nodes(2,indic(i,j,n))=(j-1)/m;
  end
end

%(tableau de connectivite du maillage)
%->Fait le lien entre le noeud d'un element
%->et le noeud dans le maillage
Mesh.elements = zeros(Mesh.Ndf,Mesh.Ne);
for j=1:m %parcours des lignes
  for i=1:m %parcours des colonnes
    Mesh.elements(1,2*indic(i,j,m)-1)=indic(i,j,n);
    Mesh.elements(2,2*indic(i,j,m)-1)=indic(i+1,j,n); %(*)
    Mesh.elements(3,2*indic(i,j,m)-1)=indic(i,j+1,n); %(**)
    Mesh.elements(1,2*indic(i,j,m))=indic(i+1,j+1,n);
    Mesh.elements(2,2*indic(i,j,m))=indic(i,j+1,n); %(**)
    Mesh.elements(3,2*indic(i,j,m))=indic(i+1,j,n); %(*)
  end
end

%Construction de la liste des faces des elements sur le bord
%->Fait le lien entre un noeud sur une face
%->et le noeud dans le maillage global
Mesh.elements_f = zeros(Mesh.Ndff,Mesh.Nef);
for i=1:m %bord bas
  Mesh.elements_f(1,i)=Mesh.elements(1,2*indic(i,1,m)-1);
  Mesh.elements_f(2,i)=Mesh.elements(2,2*indic(i,1,m)-1);

  %bord droit
  Mesh.elements_f(2,i+m)=Mesh.elements(1,2*indic(m,i,m));
  Mesh.elements_f(1,i+m)=Mesh.elements(3,2*indic(m,i,m));

  %bord haut
  Mesh.elements_f(1,i+2*m)=Mesh.elements(1,2*indic(m-i+1,m,m));
  Mesh.elements_f(2,i+2*m)=Mesh.elements(2,2*indic(m-i+1,m,m));

  %bord gauche
  Mesh.elements_f(2,i+3*m)=Mesh.elements(1,2*indic(1,m-i+1,m)-1);
  Mesh.elements_f(1,i+3*m)=Mesh.elements(3,2*indic(1,m-i+1,m)-1);
```

```
end

%Partition de frontiere
%Dans notre cas le bord est divise en 4
%bas=d1,droite=n1,haut=d2,droite=n2
%di signifie condition de Dirichlet
%ni signifie condition de Neumann
%->Donne l'indice du bord dans lequel
%->se trouve la face.
Mesh.borderFlags = zeros(1,Mesh.Nef);
for i=1:m
  %bord bas
  Mesh.borderFlags(1,i)=1;
  %bord droit
  Mesh.borderFlags(1,i+m)=2;
  %bord haut
  Mesh.borderFlags(1,i+2*m)=3;
  %bord gauche
  Mesh.borderFlags(1,i+3*m)=4;
end
% Nb de bord
Mesh.borderFlagsNb=4;
return
```

Source 20 – `regularMesh.m`

```
function Mesh=getMeshFromPET(p,e,t)
%Creation de la structure MESH en fonction des tableaux
%p,e,t obtenu par exemple avec pdetool
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : Mesh=getMeshFromPET(p,e,t)

%On obtient les noeuds des sommets
Mesh.nodes=p;
Mesh.Nn = size(Mesh.nodes,2); %Nombre de noeuds total
Mesh.Ndf = 3;  %Nombre de degres de liberte des simplexes du maillage

%On obtient le lien entre les sommet de reference
%et ceux du maillage pour chaque simplexe
Mesh.elements=t(1:3,:);
Mesh.Ne = size(Mesh.elements,2); %Nombre d'elements

%On obtient le lien entre les sommet de reference
%et ceux du maillage pour chaque simplexe contenu
%dans le bord
Mesh.elements_f=e(1:2,:);
Mesh.Ndff = 2; %Nombre de degres de liberte des bords
Mesh.Nef = size(Mesh.elements_f,2); %Nombre de faces sur le bord

%On recupere les indices des bords
```

```
Mesh.borderFlags = e(5,:);
Mesh.borderFlagsNb = max(Mesh.borderFlags);

Mesh.d = 2;  %Dimension du probleme
return
```

Source 21 – getMeshFromPET.m

```
function plotMesh(Mesh)
%Effectue le graphe du maillage
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : plotMesh(Mesh)
% Mesh est une structure contenant les donnees
% du maillage

hold on;
%Dessin des triangles
for i=1:Mesh.Ne
    %On recupere le numero des noeuds concernes
    a=Mesh.elements(1,i);
    b=Mesh.elements(2,i);
    c=Mesh.elements(3,i);

    X=[Mesh.nodes(1,a),Mesh.nodes(1,b),Mesh.nodes(1,c),Mesh.nodes(1,a)];
    Y=[Mesh.nodes(2,a),Mesh.nodes(2,b),Mesh.nodes(2,c),Mesh.nodes(2,a)];
    plot(X,Y,'-o','color','k');
end

%Dessin du bord en couleur
c = hsv(Mesh.borderFlagsNb);
for i=1:Mesh.Nef
  %On recupere le numero des noeuds concernes
    a=Mesh.elements_f(1,i);
    b=Mesh.elements_f(2,i);
    X=[Mesh.nodes(1,a),Mesh.nodes(1,b)];
    Y=[Mesh.nodes(2,a),Mesh.nodes(2,b)];
  plot(X,Y,'color',c(Mesh.borderFlags(i),:));
end
hold off;
```

Source 22 – plotMesh.m

```
function plotSolution(Mesh,u)
%Affiche le graphe de la solution
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : plotSolution(Mesh,u)
```

```
% Mesh est une structure contenant les infos du
% maillage et u est la solution obtenue avec le
% solveur

%Detecte le logiciel utilise
Octave = exist('OCTAVE_VERSION','var');
Octaviz= exist('vtk_axis','var');
Matlab=~Octave;
if(Matlab)
    %Dessin de la solution
    trimesh(Mesh.elements',Mesh.nodes(1,:),Mesh.nodes(2,:),u,'FaceColor',
        'interp','EdgeColor','k');
else
    if (Octaviz)
      % code Octave/Octaviz
        else
      % code Octave/Gnuplot
    end
end
```

Source 23 – `plotSolution.m`

Remarque

La majorité des scripts `Matlab` fonctionnent sous Octave. Cependant, en ce qui concerne l'affichage graphique, `Octave` utilise un moteur de rendu graphique externe (`Gnuplot` ou `Octavitz`). Malheureusement ceci rend la pluspart des options d'affichage de `Matlab` inutilisables pour `Octave`.

Ainsi plutôt que de faire deux codes différents, il existe une astuce permettant de détecter si le script est exécuté sous `Octave` ou `Matlab`. Au démarrage, `Octave` crée une variable nommée `OCTAVE_VERSION` et, dans le cas où `Octaviz` est le moteur de rendu, une seconde variable nommée `vtk_axis`. Ainsi il suffit de tester si ces variables ont été créées (à l'aide de la commande `exist`) pour savoir quel programme exécute le script.

```
function i = indic(x,y,n)
%Permet d'obtenir le numero d'un noeud dans un
%vecteur relativement au coordonnees de ce noeud
%dans un maillage regulier
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Remark: Cette fonction est une fonction utilisee
% par la fonction regularMesh

  %x,y position sur la grille
  %n numero du noeud
  i = (y-1)*n+x;
return
```

Source 24 – `indic.m`

```
function [x,y,Z]=getxyZ(Mesh,u)
%Permet d'obtenir des triples (x,y,z) de la solution
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : [x,y,Z]=getxyZ(Mesh,u)
% Mesh est une structure contenant les infos du
% maillage et u est la solution obtenue avec le
% solveur

for i=1:Mesh.Ne
    x(i)=Mesh.nodes(1,i);
    y(i)=Mesh.nodes(1,i);
  Z(j,i)=u(i);
end
```

Source 25 – `getxyZ.m`

Source du solveur

```
function [u,E] = solver_2d(Mesh,bc,f,g,uexact)
%Solveur elements finis permettant de resoudre le
%probleme laplacien(u)=f avec conditions au bord
%de Dirichlet homogenes ou/et de Neumann (via g)
% ------------------------------------------------
% Author: Gwenol Grandperrin
% Date : 28.05.08
% ------------------------------------------------
% Usage : [u,E] = solver_2d(Mesh,bc,f,g,uexact)
% Mesh est une structure contenant les infos du
% maillage. f et g doivent etre definies inline
% uexact est la solution exacte du probleme et
% doit etre une fonction inline. uexact est
% obtionnel, mais si elle est definit, alors
% le solveur calculera l'erreur E en norme L2
% commise lors du calcul de u.

%Ajout des dossiers importants
addpath('base');
addpath('integration');
addpath('mesh');

%Parametres de la simulation
par.gp=3; %Nombre de points de Gauss (quadrature)
par.solver='\'; %Solveur utilise
%par.solver='pcg'; %Solveur utilise
```

```
%Evaluation des fonctions de forme locales
%aux points de Gauss
[x,w]=getPointsAndWeights(Mesh.d,par.gp);
base_ref = zeros(Mesh.Ndf,par.gp);
for i=1:par.gp
  for j=1:Mesh.Ndf
    base_ref(j,i)=phi_ref(j,x(i,:));
  end
end

%Evaluation des fonctions de forme locales
%aux points de Gauss (pour la frontiere)
[x_f,w_f]=getPointsAndWeights(Mesh.d-1,par.gp);

%On procede comme precedemment
base_ref_f = zeros(Mesh.Ndff,par.gp);
for i=1:par.gp
  for j=1:Mesh.Ndff
    base_ref_f(j,i)=phi_ref_f(j,x_f(i));
  end
end

%Evaluation des quantites derivees
%au point de Gauss
%==>simplification car nos fonctions phi_n
%==>sont constantes sur les simplexes
d_base_ref=zeros(Mesh.d,Mesh.Ndf);
for i=1:Mesh.d
  for j=1:Mesh.Ndf
    d_base_ref(i,j)=d_phi_ref(j,i);
  end
end

%Evaluation des quantites derivees
%au point de Gauss (pour la frontiere)
d_base_ref_f=zeros(Mesh.d-1,Mesh.Ndf);
for i=1:Mesh.Ndff
  d_base_ref_f(i)=d_phi_ref_f(i);
end

%Calcul du Jacobien et des poids de quadrature
%Ainsi que de la derivee de la forme globale

%Poids du Mesh.Ne -eme element au par.gp-eme point de Gauss
poids_K = zeros(par.gp,Mesh.Ne);

%Derive d-eme de la Mesh.Ndf-eme fonction de base du par.gp-eme noeud
%de la Mesh.Ne-eme fonction de base
d_base_K = zeros(Mesh.d,Mesh.Ndf,par.gp,Mesh.Ne);

for el=1:Mesh.Ne %Pour tout les elements
for l=1:par.gp %Points de Gauss
J = zeros(Mesh.d,Mesh.d); %J(i,j)
```

```
for i=1:Mesh.d
  for j=1:Mesh.d
    for k=1:Mesh.Ndf
      J(i,j)=J(i,j)+Mesh.nodes(i,Mesh.elements(k,el))*d_base_ref(j,k);
    end
  end
end
inv_jac=inv(J);
for i=1:Mesh.d
  for j=1:Mesh.Ndf
    for k=1:Mesh.d
      d_base_K(i,j,l,el)=d_base_K(i,j,l,el)+d_base_ref(k,j)*inv_jac(k,i);
    end
  end
end
poids_K(l,el)=w(l)*det(J);
end
end

%Calcul du Jacobien et des poids de quadrature (pour la frontiere)

%Poids du Mesh.Ne -eme element au par.gp-eme point de Gauss
poids_K_f = zeros(par.gp,Mesh.Nef);

for el=1:Mesh.Nef %Pour tout les elements
for l=1:par.gp %Points de Gauss
gamma=zeros(1,2);
for i=1:Mesh.d
  for k=1:Mesh.Ndff
    gamma(i)=gamma(i)+Mesh.nodes(i,Mesh.elements_f(k,el))*d_base_ref_f(k)
        ;
  end
end

poids_K_f(l,el)=w_f(l)*norm(gamma); %Dans notre cas norm(gamma)=h
end
end

%Construction de la matrice A
A = zeros(Mesh.Nn);
for i=1:Mesh.Ne %Pour chaque element
  for j=1:par.gp %Pour chaque point de Gauss
    for ni=1:Mesh.Ndf
      I=Mesh.elements(ni,i);
      for nj=1:Mesh.Ndf
        J=Mesh.elements(nj,i);
        %Evaluer le gradient des phi_i
        a=0;
        for k=1:Mesh.d
          a=a+d_base_K(k,nj,j,i)*d_base_K(k,ni,j,i);
        end
        A(I,J)=A(I,J)+a*poids_K(j,i);
      end
    end
```

```
  end
end

%Construction du vecteur F
F = zeros(Mesh.Nn,1);
for i=1:Mesh.Ne %Pour chaque element
  for j=1:par.gp %Pour chaque point de Gauss
    %Evaluer les coordonnees du noeud de Gauss - T_{K^m}(xi)
    ptGauss=zeros(1,Mesh.d);
    for k1=1:Mesh.d
      for N=1:Mesh.Ndf
        ptGauss(k1)=ptGauss(k1)+Mesh.nodes(k1,Mesh.elements(N,i))*
            base_ref(N,j);
      end
    end

    %Integration numerique
    for ni=1:Mesh.Ndf
      I=Mesh.elements(ni,i);
      %Evaluer \int f*phi
      a=f(ptGauss(1),ptGauss(2))*base_ref(ni,j);

      %Cumulation
      F(I)=F(I)+a*poids_K(j,i);
    end
  end
end

for i=1:Mesh.Nef %Pour chaque element du bord
  %On regarde sur quel bord on se trouve
    %En fait on test si c'est une condition de Neumann sur ce bord
  if(ismember(Mesh.borderFlags(i),bc.neumann))
    for j=1:par.gp %Pour chaque point de Gauss
      %Evaluer les coordonnees du noeud de Gauss - T_{K^m}(xi)
      ptGauss=zeros(1,Mesh.d);

      if(Mesh.borderFlags(i)==2) %premiere coordonnee
        %ptGauss(1)=1;
      elseif(Mesh.borderFlags(i)==4)
        %ptGauss(1)=0;
      end
      for N=1:Mesh.Ndff
               ptGauss(1)=ptGauss(1)+Mesh.nodes(1,Mesh.elements_f(N,i))*
                   base_ref_f(N,j);
        ptGauss(2)=ptGauss(2)+Mesh.nodes(2,Mesh.elements_f(N,i))*
            base_ref_f(N,j);
      end

      %Integral numerique
      for ni=1:Mesh.Ndff
        I=Mesh.elements_f(ni,i);

        %Evaluer \int_{\Gamma_N} Phi*phi
        a=g(ptGauss(1),ptGauss(2))*base_ref_f(ni,j);
```

```
      %Cumulation
      F(I)=F(I)+a*poids_K_f(j,i);
    end
  end
 end %Sinon est sur le bord avec cond de Dirichlet
end


%Modification de A pour tenir compte
%des conditions de dirichlet
for i=1:Mesh.Nef
 if(ismember(Mesh.borderFlags(i),bc.dirichlet)) %Si on est sur le bon
     bord
  for ni=1:Mesh.Ndff
    I=Mesh.elements_f(ni,i);%On attrape le numero du noeud
    A(I,:)=zeros(1,Mesh.Nn);
    A(I,I)=1;
    F(I)=0;
  end
 end
end

%Calcul de la solution
if strcmp(par.solver,'\')
   u=A\F;
elseif strcmp(par.solver,'pcg')
   u=pcg(A,F);
else
   error('Solveur inconnu');
end

%Calcul de l'erreur ||u-uh|| en norme L2
if nargin == 5 %Si l'on a pas la solution exacte, on le calcul pas l'
    erreur
E=0;
for i=1:Mesh.Ne %Pour chaque element
 for j=1:par.gp %Pour chaque point de Gauss
  %Evaluer les coordonnees du noeud de Gauss - T_{K^m}(xi)
  ptGauss=zeros(1,Mesh.d);
  for k1=1:Mesh.d
    for N=1:Mesh.Ndf
      ptGauss(k1)=ptGauss(k1)+Mesh.nodes(k1,Mesh.elements(N,i))*
          base_ref(N,j);
    end
  end

  %Integration numerique
  uh=0;
  for ni=1:Mesh.Ndf
    I=Mesh.elements(ni,i); %numero du noeud
    uh=uh+base_ref(ni,j)*u(I);
  end
```

```
    %Evaluer \int (u-uh)^2
    a=(uexact(ptGauss(1),ptGauss(2))-uh)^2;

    %Cumulation
    E=E+a*poids_K(j,i);
  end
end
E=sqrt(E);
end

%Affichage de la solution
plotSolution(Mesh,u);

return
```

Source 26 – `solver_2d.m`

Références

[1] M. Discacciati. *Numerical Approximation of Partial Differential Equations (cours)*. 2008.

[2] A. Quarteroni et A. Valli. *Numerical Approximation of Partial Differential Equations*. Number 23.

[3] Alexandre Ern et Jean-Luc Guermond. *Eléments finis : théorie, applications, mise en oeuvre*. Springer, 2002.

[4] Thomas J. R. Hughes. *The Finite Element Method*. Dover, 2000.