SORBONNE UNIVERSITÉ

INSTITUT de MATHÉMATIQUES de MARSEILLE

Mémoire de fin d'études, master mathématiques pour la modélisation de Sorbonne-Université

Á l'Institut de Mathématiques de Marseille

Etudes numériques d'écoulements de fluides visqueux purs et en milieux poreux

Julien VALENTIN

ENCADRÉ PAR

Philippe ANGOT

13 novembre 2022

Table des matières

Introduction

Le sujet de mon stage au sein de l'équipe d'Analyse Appliquée de l'Institut de Mathématiques de Marseille devait être consacré à l'étude numérique de l'écoulement d'un fluide dans une fracture, en deux dimensions d'espace. Nous aurions eu alors une superposition de trois couches : une couche poreuse sur une couche fluide, les deux sur une seconde couche poreuse. Ce genre de modèle trouve des applications très variées. Les interactions *fluides poreux* se trouvent aussi bien dans les sciences médicales que les géosciences, ou encore dans le domaine de l'énergie. On commence par modéliser les écoulements monophasiques puis on peut étudier le transport de polluants, leur infiltration, leur interaction avec une autre phase.

On pense par exemple aux infiltrations d'eau de mer dans les nappes phréatiques par infiltration dans un sol, solide certes mais pas imperméable. On peut aussi imaginer une fissure dans une canalisation, et tenter de quantifier l'impact en terme de perte en eau dans un réseau de distribution. Enfin, on peut penser à des vaisseaux sanguins détériorés... Les applications sont, donc, très vastes, et touchant à des domaines stratégiques.

Je n'ai pas réussi à faire plus qu'effleurer le sujet initialement prévu par mon tuteur Philippe Angot, l'essentiel de mon temps ayant été consacré à découvrir le domaine de la modélisation des milieux continus en physique, s'en est suivie une longue phase d'appréhension de l'analyse des modèles proposés pour les écoulements de fluides visqueux ; enfin, mais bien tard, il m'a fallu découvrir la méthode des volumes finis et les quelques subtilités qui la rendent différente des différences finies. La majeur partie du temps ayant été consacrée à l'apprentissage des pré-requis d'une part et des bonnes pratiques d'autre part, je me suis trouver à cours de temps pour achever le stage.

Tout d'abord, un chapitre préliminaire est consacré aux concepts des volumes finis et à leur application au langage Python. J'ai hésité avant de mettre les codes en eux-mêmes ; ils sont présents car font partie intégrante du stage, et le format mémoire ne présente en principe pas de contrainte de pagination. Ils n'y seraient pas dans d'autres contexte. Du reste, ils justifient en partie la nécessité de choisir des technologies, et font donc entièrement partie de la réflexion lorsqu'on aborde un sujet tel que l'analyse numérique.

Le deuxième chapitre introduira le système de Stokes, la théorie permettant d'obtenir une formulation variationnelle et introduira la notion de condition inf sup, résultat primordial pour obtenir l'existence et l'unicité d'une solution. On présentera alors un maillage admissible pour la simulation des problèmes de type Stokes stationnaire et deux solutions exactes qui serviront à valider les schéma proposés. Ensuite on passera un peu de temps sur les problèmes de Stokes dépendant du temps et on utilisera une méthode de projection scalaire pour simuler les tourbillons de Taylor-Green.

Ensuite viendra le moment d'étudier l'écoulement de fluides visqueux monophasiques dans un milieu poreux. On validera également le comportement du code développé pour le système de Stokes dans ce contexte nouveau mais assez proche. En particulier, on étudiera numériquement le lien entre les différents termes de l'équation de Brinkman, posé comme un problème de Stokes généralisé.

Enfin, et pour conclure, on proposera une tentative d'implémentation d'un saut, nul, entre une couche poreuse et un milieu fluide. Cette proposition étant visiblement un échec, les erreurs obtenues étant en ordre de grandeur, trop grandes. Cela dit, on verra que la tendance est à la convergence malgré tout.

Chapitre 1

Préliminaires sur les volumes finis et le langage Python

Ce chapitre est dédié à de brefs rappels sur l'analyse numérique par la méthode des volumes finis et quelques exemples de problèmes traités en langage Python. On détaillera les outils et quelques fonctions utiles pour assembler les différents opérateurs rencontrés dans la suite du mémoire. On commence par quelques éléments de théorie largement connus avant de passer à quelques subtilités liées au langage Python et aux librairies disponibles pour ce langage et spécialisées en calcul scientifique. Le lecteur souhaitant utiliser les éléments de codes proposés ici devra s'assurer de disposer des packages **Matplotlib**, **Numpy** ainsi que **Scipy**. Ce chapitre est largement inspiré de [EGH00].

1.1 Exemple introductif

On utilise un problème de Poisson en 1.D pour se familiariser avec le calcul numérique en Python. Pour les résultats d'existence, d'unicité et de régularité de la solution du problème continu, on pourra consulter par exemple [Bré83].

L'analyse du schéma obtenu par la méthode des volumes finis sur un maillage admissible est donné dans l'introduction et développé dans le chapitre 5 de [EGH00]. On trouve en particulier un argument pour l'existence ainsi que l'unicité d'une solution au problème discret, une définition de l'erreur de consistance locale dans le cadre des volumes finis, où il est question de la consistance des flux - voire la comparaison, dans le même ouvrage, avec la notion de consistance au sens des différences finies puis d'une méthode mixte d'éléments finis. Enfin, est étudiée la convergence en norme pour une solution $\mathcal{C}^2$.

Un problème elliptique en une dimension

Soit à résoudre numériquement le problème de Dirichlet suivant

On considère le problème de Poisson en une dimension. On pose $\Omega := (0,1)$ et

$$\left\{ \begin{array}{rll} u''(x) & = e^x & \forall x \in \Omega \\ u(0) & = 1 & \\ u(1) & = e & \end{array} \right.$$

L'unique solution de ce problème est $u : x \mapsto e^x$. On introduit le maillage $\mathcal{T}$ de pas à l'intérieur Δx constant égal à $1/n_x$ où n_x désigne le nombre de volumes considérés. On introduit deux points aux bords, dont la valeur est donnée par les conditions de Dirichlet. Le Laplacien se discrétise en chaque point intérieur au domaine Ω. Soit K un volume élémentaire, alors

$$\begin{aligned} \int_K \Delta u(x) dx &= \int_{\partial\Omega} (\nabla u \cdot \mathbf{n})(\gamma(s)) d\gamma(s) \\ &= \sum_{\sigma \in \mathcal{E}_K} F_{K,\sigma} \end{aligned}$$

Dans le cas $1.D$, cette somme est réduite à deux termes

$$\sum_{\sigma \in \mathcal{E}_K} F_{K,\sigma} = F_{K,+} - F_{K,-}$$

Parmi les très nombreux choix imaginables pour approcher les flux, un exemple est le choix d'une différence finie centrée

$$F_{K|L,\sigma} \sim \frac{U_K - U_L}{\Delta x}$$

Le second membre demande un nouveau choix : comment approcher la valeur de f au sein du volume K ? A nouveau, existent énormément de manières de procéder et un choix courant est d'utiliser la valeur moyenne de f sur K. Une fois fixé, cela conduit à l'approximation

$$\int_K f(x)dx \sim \Delta x f_K$$

où f_K est l'approximation choisie.

On a donc un schéma discret à l'intérieur, à deux subtilités près : on remarque que lorsque K est le premier (respectivement le dernier) volume de la discrétisation, l'approximation du flux fait intervenir des points fictifs : $U_{-1/2}$ ou $U_{n_x+1/2}$. On utilise alors l'extrapolation linéaire sur la face 0, ou n_x :

$$\frac{U_{1/2} + U_{-1/2}}{2} = 1 \iff U_{-1/2} = 2 - U_{1/2}$$
$$\frac{U_{n_x+1/2} + U_{n_x-1/2}}{2} = e \iff U_{n_x+1/2} = 2e - U_{n_x-1/2}$$

Et on substitue ces formules dans l'expression des flux, ce qui modifie le second membre et un facteur devant un noeud existant. On obtient donc

$$U_0 = 1$$
$$\frac{U_{i+3/2} - 2U_{i+1/2} + U_{i-1/2}}{\Delta x} = \Delta x f_i \qquad \forall i \in [\![1, n_x]\!]$$
$$U_{n_x+1} = e$$

Une implémentation Python possible pour ce problème est la suivante.

```
from matplotlib import pyplot
import numpy

u_func = lambda x: numpy.exp(x)
f_func = lambda x: numpy.exp(x)
x0 = 0.
x1 = 1.
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096])
dx_ = numpy.empty(nx_.shape[0])
err = numpy.empty(nx_.shape[0])

for l, nx in enumerate(nx_):
    dx = (x1 - x0)/ nx
    dx_[l] = dx
    x_f = numpy.linspace(x0, x1, nx+1)
    x_c = numpy.linspace(x0+dx/2, x1-dx/2, nx)
    x_ = numpy.concatenate([[x0], x_c, [x1]], axis=0)
    u = numpy.concatenate([[u_func(x0)], u_func(x_c), [u_func(x1)]], axis=0)

    U = numpy.zeros((nx+2))
    b = f_func(x_c)
    A = -2/dx**2 * numpy.eye(nx) + 1/dx**2 * numpy.diag(numpy.ones(nx-1), 1) + \
        1/dx**2 * numpy.diag(numpy.ones(nx-1), -1)
    # Conditions de Dirichlet
    A[0, 0] -= 1/dx**2
    b[0] -= 2*u_func(x0)/dx**2
    A[-1, -1] -= 1/dx**2
    b[-1] -= 2*u_func(x1)/dx**2
```

```
    # Résolution
    U[1:-1] = numpy.linalg.solve(A, b)
    U[0]    = u_func(x0)
    U[-1]   = u_func(x1)

    # Estimation de la norme L^2
    err[l] = numpy.sqrt(dx) * numpy.linalg.norm(U-u, ord=2)

pyplot.figure()
pyplot.plot(x_, U, label=r"$(U_i)$", marker='x')
pyplot.plot(x_, u, label=r"$u(x)$")
pyplot.legend()
pyplot.suptitle("Graphe de la solution.")
pyplot.show()

pyplot.figure()
pyplot.loglog(nx_, err, label="Erreur", marker="x")
pyplot.loglog(nx_, dx_, "r--", label=r"$y = dx$")
pyplot.loglog(nx_, dx_**2, "g--", label=r"$y = dx^2$")
pyplot.legend()
pyplot.show()
```

laquelle rend les deux graphes

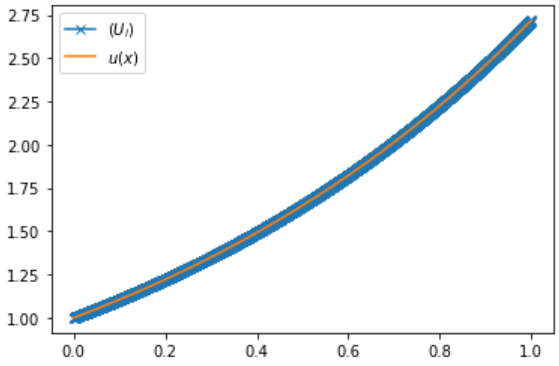

FIGURE 1.1 – Graphes des solutions analytiques et approchées

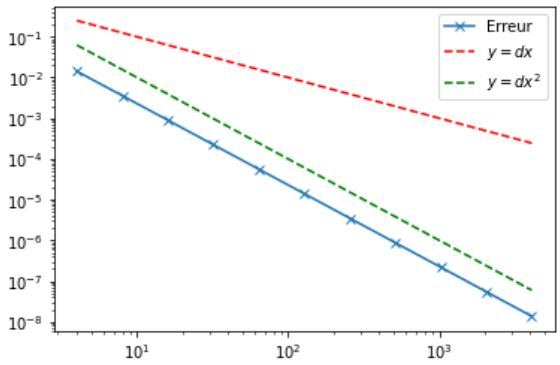

FIGURE 1.2 – Estimation de l'erreur $L^2(\Omega)$ en fonction du nombre de volumes n

1.2 Problème modèle en dimension deux

De nouveau, l'analyse du problème de Poisson avec des conditions aux limites variées est traité dans [Bré83]. On a un résultat d'existence et d'unicité ainsi que l'étude de différentes régularités pour l'unique solution, cela en fonction de données variées. On trouvera aussi les résultats d'injections et de leurs propriétés des espaces Sobolev dans différents espaces fonctionnels plus classiques.

La définition d'un maillage admissible est donné chapitre 9, hypothèses 9.1 de [EGH00] dans un contexte assez général. Nous n'utiliserons que des maillages rectangulaires, éventuellement décalés dans une ou les deux dimensions.

Ensuite vient la donnée de résultats continus connus comme l'inégalité de Poincaré, sous forme discrète pour des conditions de Dirichlet sur chaque bord. Sous réserve de connexité d'Ω, il est possible de l'obtenir avec Dirichlet sur une face seulement.

La section 9.3 démontre l'existence et l'unicité d'une solution discrète pour le problème de Dirichlet par l'estimation de la norme $H_0^1(\Omega$ en fonction du diamètre de Ω et de la norme $L^2(\Omega)$ du second membre.

Ensuite la notion de convergence est largement étudiée pour différentes régularités des données, et donc de la solution.

Un exemple

Soit $\Omega := (0,1) \times (0,1)$, ouvert connexe borné lipschtizien de $\mathbb{R}^2$. Soit $u : x, y \mapsto cos(2\pi x)sin(2\pi y)$, analytique sur Ω. Cette fonction est solution du problème de Dirichlet 2.D

$$\begin{cases} \Delta u(x,y) & = -8\pi^2 u(x,y) & \forall x, y \in \Omega \\ u_{|\partial\Omega} & = u^D(x,y) & \forall x, y \in \partial\Omega \end{cases}$$

On discrétise Ω en un ensemble de $n_y \times n_x$ volumes élémentaires. On introduit les pas d'espace en chaque dimension $\Delta x := 1/n_x$ et $\Delta y := 1/n_y$. Les arêtes verticales sont de mesure uniforme Δy et les arêtes horizontales de mesure Δx. Les centres des faces horizontales (à gauche et à droite des volumes) ont pour absisses les éléments de $\{x_i\}_{i\in[\![0,n_x]\!]} = \{i\Delta x\}$ et pour ordonnées $\{y_{j+1/2}\}_{j\in[\![0,n_y-1]\!]} = \{(j+1/2)\Delta y\}_j$, les centres des faces verticales (au-dessus et en dessous des volumes) ont pour absisses $\{x_{i+1/2}\}_{i\in[\![0,n_x-1]\!]} = \{(i+1/2)\Delta x\}_i$ et pour ordonnées $\{y_j\}_{j\in[\![0,ny]\!]} = \{j\Delta y\}_j$. Enfin, les centres des volumes sont aux absisses et ordonnées d'indice semi-entières.

Les noeuds de vitesse U_h sont positionnés aux centres des volumes et également sur le bord $\partial\Omega$ où ils sont donnés par les conditions de Dirichlet, c.f la figure suivante

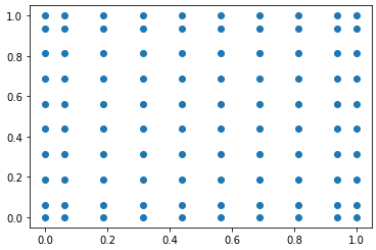

FIGURE 1.3 – Maillage structuré uniforme de $\bar{\Omega} = [0,1]^2$ en 8×8 volumes à l'intérieur. Les points bleus à l'intérieur en sont les centres.

Ce maillage est *admissible* au sens des hypothèses 9.2 de [EGH00].

On ne numérote que les inconnues à l'intérieur du domaine, les autres étant résolues ensuite, en commençant par celle située en bas à gauche et en suivant l'ordre lexicographique, du bas vers le haut. La correspondance entre l'indice k de l'inconnue et son repérage (j, i) sur la grille se fait par la relation $k = j(n_x+2)+i$. Réciproquement, $i = k\%(n_x+2)$, reste de la division euclidienne (écrite avec le symbole Python) et $j = k//(n_x+2)$, quotient de la division euclidienne (symbole Python).

Formulation du système linéaire Le système linéaire associé au problème est donné par

$$\frac{\Delta y}{\Delta x}\left(U_{h,j,i+1} - 2U_{h,j,i} + U_{h,j,i+1}\right) + \frac{\Delta x}{\Delta y}\left(U_{h,j+1,i} - 2U_{h,j,i} + U_{h,j-1,i}\right) = (\Delta y \Delta x) f_{j,i}$$

si $U_{h,j,i}$ est à l'intérieur du domaine Ω et sans flux au bord, sans arête incluse dans $\partial\Omega$. Si $U_{h,j,i}$ est au centre d'un volume possédant une arête sur un bord, alors un des points $U_{h,j+1,i}$, $U_{h,j-1,i}$, $U_{h,j,i+1}$, $U_{h,j,i-1}$ au moins est fictif. Dans ce cas on substitue sa valeur dans le bilan de flux.

Par exemple, si $j = 0$, alors $\frac{U_{h,0,i}+U_{h,-1,i}}{2} = u^D_{0,i}$, et on peut substituer $U_{h,-1,i}$ dans la formule précédente par une expression faisant intervenir un noeud existant et une donnée qui passe au second membre. Pour ce maillage, la technique est identique sur chaque bord. Plutôt que rallonger ce texte, passons au code et à son résultat.

```
from matplotlib import pyplot
import numpy

u_func = lambda x, y: numpy.cos(2*numpy.pi*x)*numpy.sin(2*numpy.pi*y)
f_func = lambda x, y: -8*numpy.pi**2 * numpy.cos(2*numpy.pi*x) * \
                      numpy.sin(2*numpy.pi * y)
x0 = 0.
x1 = 1
y0 = 0.
y1 = 1

nx_ = numpy.array([4, 8, 16, 32, 64, 128])
dx_ = numpy.empty(nx_.shape[0])
err = numpy.empty(nx_.shape[0])

for l, nx in enumerate(nx_):
    ny = nx
    dx = (x1 - x0) / nx
    dy = (y1 - y0) / ny
    dx_[l] = dx
    print("===== Simulation sur {}x{} volumes. =====".format(ny, nx))

    # Discrétisation spatiale
    x_f = numpy.linspace(x0, x1, nx+1)
    x_c = numpy.linspace(x0+dx/2, x1-dx/2, nx)
    x_  = numpy.concatenate([[x0], x_c, [x1]])

    y_f = numpy.linspace(y0, y1, ny+1)
    y_c = numpy.linspace(y0+dy/2, y1-dy/2, ny)
    y_  = numpy.concatenate([[y0], y_c, [y1]])

    X_c, Y_c = numpy.meshgrid(x_c, y_c)
    X_, Y_   = numpy.meshgrid(x_, y_)

    # Vecteur solution analytique
    u = u_func(X_, Y_)

    # Initialisation
    U = numpy.empty((ny+2, nx+2))
```

```
    indices_bords = {
            'bas': [i for i in range(nx)],
            'droit': [(j+1)*nx-1 for j in range(ny)],
            'haut': [(ny-1)*nx+i for i in range(nx)],
            'gauche': [j*nx for j in range(ny)]
        }

    # Système linéaire
    b = (dx*dy) * f_func(X_c, Y_c)
    b = b.flatten()

    A = -2 * (dy/dx + dx/dy) * numpy.eye(nx*ny) + \
        dy/dx * numpy.diag(numpy.ones(nx*ny-1), 1) + \
        dy/dx * numpy.diag(numpy.ones(nx*ny-1),-1) + \
        dx/dy * numpy.diag(numpy.ones(nx*ny-nx), nx) + \
        dx/dy * numpy.diag(numpy.ones(nx*ny-nx),-nx)

    # Modifications des noeuds ayant un flux au bord
    for k in indices_bords['bas']:
        A[k, k] -= dx/dy
        b[k] -= 2 * dx/dy * u_func(x_c[k%nx], y0)
    for k in indices_bords['droit']:
        if k < nx*ny-1: A[k, k+1] = 0
        A[k, k] -= dy/dx
        b[k] -= 2 * dy/dx * u_func(x1, y_c[k//nx])
    for k in indices_bords['haut']:
        A[k, k] -= dx/dy
        b[k] -= 2 * dx/dy * u_func(x_c[k%nx], y0)
    for k in indices_bords['gauche']:
        if k > 1: A[k, k-1] = 0
        A[k, k] -= dy/dx
        b[k] -= 2 * dy/dx * u_func(x0, y_c[k//nx])

    U[1:-1, 1:-1] = numpy.linalg.solve(A, b).reshape((ny, nx))
    U[0, :] = u_func(x_, y0)
    U[-1,:] = u_func(x_, y1)
    U[:, 0] = u_func(x0, y_)
    U[:,-1] = u_func(x1, y_)

    err[l] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)

pyplot.imshow(numpy.abs(U-u))
pyplot.colorbar()
pyplot.show()

pyplot.loglog(nx_, dx_, 'r--', label=r"$\Delta x$")
pyplot.loglog(nx_, dx_**2, 'b--', label=r"$\Delta x^2$")
pyplot.loglog(nx_, err, label='x', marker="x")
pyplot.legend()
pyplot.show()
```

Qui rend les figures

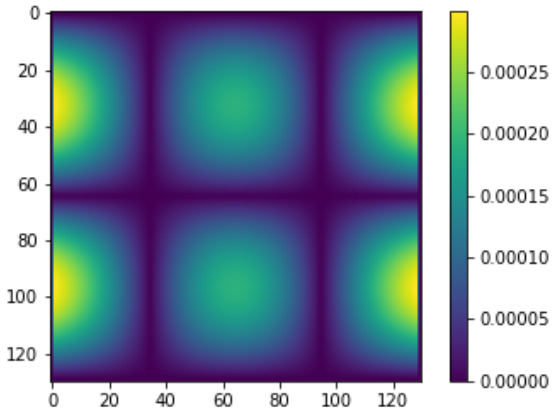

FIGURE 1.4 – Champ d'erreurs $|U_h - u_h|$ pour $n = 128^2$ volumes

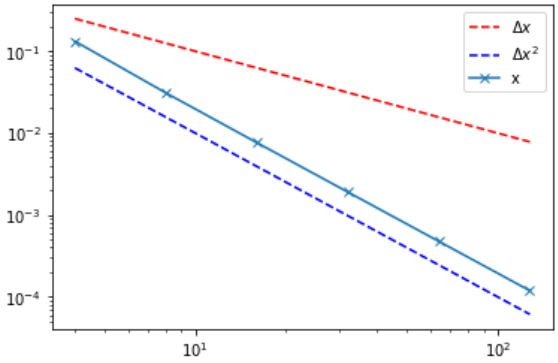

FIGURE 1.5 – Estimation de l'erreur $L^2(\Omega)$ en fonction du nombre de volumes n

Bien que le graphe soit satisfaisant, nous faisons face ici à deux problèmes majeurs : le temps de calcul ainsi que la complexité mémoire de l'algorithme. Le temps de calcul pour ce programme est d'en moyenne 44.8s avec un écart de type de 13s, comme on le voit sur la figure 1.6

D'autre part, si cette exécution est effectuée sur une grille carrée de taille 128×128, le passage à 256×256 n'est plus possible sur un ordinateur personnel. En effet, l'implémentation du système linéaire en tant qu'objet plein demande dans ce cas 32 GiB de données pour stocker la matrice du système linéaire dans le format *float64*. On peut le vérifier sur la figure 1.7

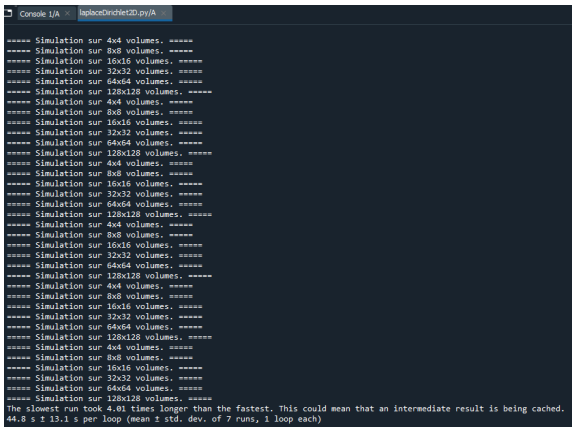

FIGURE 1.6 – Sortie console de la commande *magic* **%timeit** de l'enironnement interactif de Python : sur 7 essais, il rend la durée moyenne de l'exécution et son l'écart-type.

FIGURE 1.7 – Erreur rendue lors de l'exécution de la simulation pour 256×256 volumes.

Afin de palier à ces problèmes, on retente l'expérience avec l'implémentation en matrices creuses. Ces objets sont implémentés dans la bibliothèque **Scipy**, plus précisément dans la sous-librairie **Scipy.sparse**. Une matrice creuse est une manière de représenter les matrices en ne gardant en mémoire que ses entrées non-nulles. Pour l'implémenter, on prescrit trois tableaux, contenant la ligne de la donnée à stocker, la ligne puis la colonne où elle se trouve. Par exemple, la représentation creuse de

$$\begin{pmatrix} 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 5 \end{pmatrix}$$

est donnée par le triplet

donnée	9	8	7	6	5
ligne	1	2	3	5	5
colonne	1	3	2	4	5

Laplacien de Dirichlet 2.D revisité

On reprend un problème de Dirichlet en deux dimensions d'espace avec maintenant la déclaration de l'opérateur laplacien en matrices creuses. Cette fois, on considère les noeuds de l'inconnue discrète U_h sur les faces des volumes élémentaires et on les comptes dans le système

On vérifie que ce changement de convention n'a aucune incidence notable sur la résolution du problème.

```
from matplotlib import pyplot
import numpy
from numpy import cos, pi, sin
import scipy
from scipy import sparse
from scipy.sparse import linalg
```

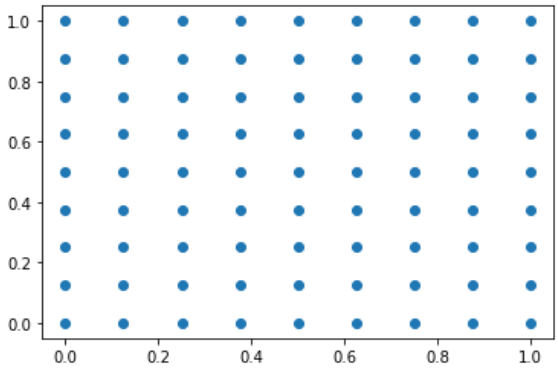

FIGURE 1.8 – Localisation des noeuds en lesquels est définie U_h

```
# SETUP
u_func  = lambda x, y : sin(x)*cos(y)
f_func = lambda x, y : 2 * u_func(x, y)

# DISCRETISATION SPATIALE
## horizontale
xi = 0.
xf = 2*pi
L  = (xf-xi)
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 256, 512, 1024])
dx_ = L/nx_
## verticale
yi = 0.
yf = 2*pi
H  = (yf-yi)

# INITIALISATION DES ERREURS
err = numpy.zeros(nx_.shape)

# ITERATION SPATIALE
for l, nx in enumerate(nx_):

    dx = dx_[l]
    # DISCRETISATION SPATIALE CARREE
    ny = nx
    dy = H/ny
    ## Info
    print("> Simulation sur maillage {}x{}".format(ny, nx))

    # CONSTRUCTION DE LA GRILLE
    x_  = numpy.linspace(xi, xf, nx+1)
    y_  = numpy.linspace(yi, yf, ny+1)
    x_u, y_u = numpy.meshgrid(x_, y_)

    # DECLARATION DE L'OPERATEUR (-) LAPLACIEN DE DIRICHLET DISCRET CREUX
    n = 5 * (ny-1)*(nx-1) + 2*nx + 2*ny

    lignes   = numpy.zeros(n)
    colonnes = numpy.zeros(n)
    donnees  = numpy.zeros(n)
    compteur = 0
```

```
    b = dy*dx * f_func(x_u, y_u)
    b[0, :] = u_func(x_, yi)
    b[-1,:] = u_func(x_, yf)
    b[:, 0] = u_func(xi, y_)
    b[:,-1] = u_func(xf, y_)

    for j in range(ny+1):
        for i  in range(nx+1):

            if i == 0 or i == nx or j == 0 or j == ny:
                lignes[compteur]   = j * (nx+1) + i
                colonnes[compteur] = j * (nx+1) + i
                donnees[compteur]  = 1
                compteur += 1

            else:
                lignes[compteur:compteur+5] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                colonnes[compteur+4] = (j+1) * (nx+1) + i
                donnees[compteur]   = 2 * (dy/dx + dx/dy)
                donnees[compteur+1] = -dy/dx
                donnees[compteur+2] = -dy/dx
                donnees[compteur+3] = -dx/dy
                donnees[compteur+4] = -dx/dy
                compteur += 5

    A = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+1), (ny+1)*(nx+1))).tocsr(

    # RESOLUTION
    U = linalg.spsolve(A, b.flatten()).reshape((ny+1, nx+1))

    # ANALYSE
    u = u_func(x_u, y_u)
    err[l] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)

# FIGURES
pyplot.figure()
pyplot.loglog(nx_, err, 'x-', label=r"$||U-u||_{L^2}$")
pyplot.loglog(nx_, dx_, '--', label=r"$\Delta x$")
pyplot.loglog(nx_, dx_**2, '--', label=r"$\Delta x^2$")
pyplot.legend()
pyplot.show()

pyplot.figure()
pyplot.imshow(numpy.abs(U-u))
pyplot.colorbar()
pyplot.show()
```

On trouve les courbes

Afin de pouvoir comparer aisément les temps d'exécutions avec d'autres méthodes, on redonne l'analyse temporelle pour l'analyse du schéma entre 4 et 512 volumes par dimension. On trouve plutôt

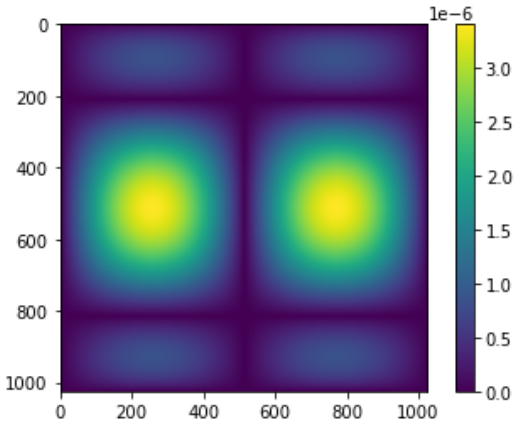

FIGURE 1.9 – Erreur rendue lors de l'exécution de la simulation pour 1024×1024 volumes.

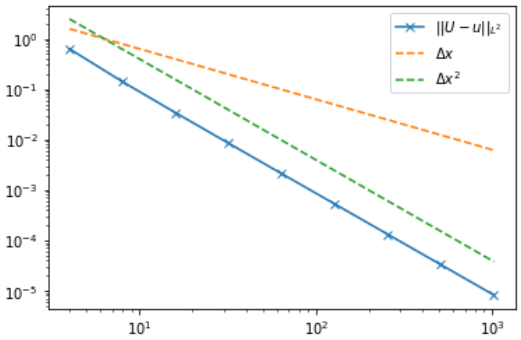

FIGURE 1.10 – Estimation de l'erreur $L^2(\Omega)$ pour des maillages entre 4×4 et 1024×1024 volumes.

```
44.3 s ± 1.13 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.11 – Estimation de la durée de l'exécution pour 1024×1024 volumes.

```
6.31 s ± 85 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.12 – Estimation de la durée de l'exécution pour 512×512 volumes.

1.3 Résolution par méthode G.M.R.E.S

Expérience numérique

On reprend exactement la simulation de la partie précédente, avec au maximum 512 volumes par dimension, en substituant à

```
U = linalg.spsolve(A, b.flatten()).reshape((ny+1, nx+1))
```

la ligne

```
U = linalg.gmres(A, b.flatten())[0].reshape((ny+1, nx+1))
```

et on reproduit l'estimation de la durée. On trouve, pour cette méthode de résolution, les graphes suivants

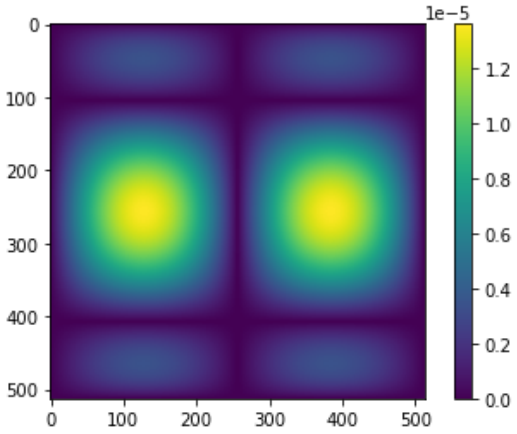

FIGURE 1.13 – Erreur rendue lors de l'exécution de la simulation pour 512×512 volumes.

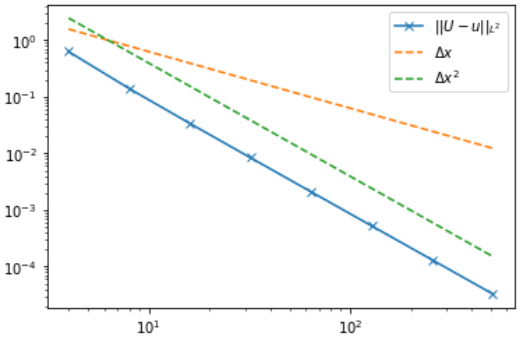

FIGURE 1.14 – Estimation de l'erreur $L^2(\Omega)$ pour des maillages entre 4×4 et 512×512 volumes.

```
6.52 s ± 214 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.15 – Compte-rendu de l'estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes.

En guise de commentaire, on remarque que la convergence est bien assurée mais le gain de temps est très incertain, en fait le temps moyen est même supérieur à celui affiché par la méthode directe. L'appel à la méthode G.M.R.E.S n'est donc ici d'aucune utilité.

1.4 Résolution par la méthode BI.C.G.STAB

Expérience numérique

Une dernière fois on essaie de résoudre le système linéaire modèle à l'aide de cette nouvelle méthode. La résolution se fait en substituant à la ligne

```
U = linalg.spsolve(A, b.flatten()).reshape((ny+1, nx+1))
```

la ligne

```
U = linalg.bicgstab(A, b.flatten())[0].reshape((ny+1, nx+1))
```

L'estimation du temps d'exécution rend

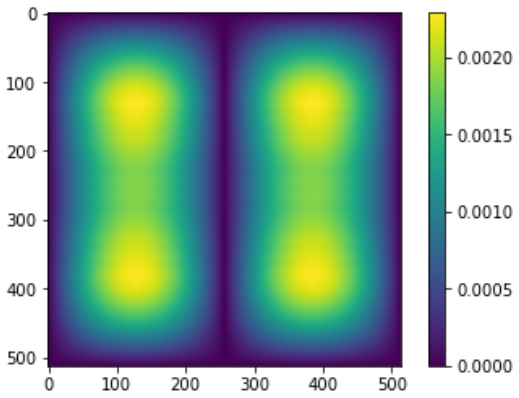

FIGURE 1.16 – Erreur rendue lors de l'exécution de la simulation pour 512×512 volumes.

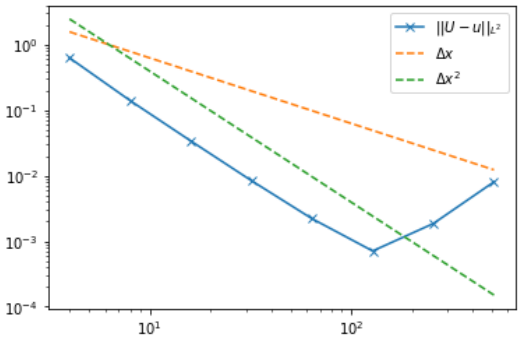

FIGURE 1.17 – Estimation de l'erreur $L^2(\Omega)$ pour des maillages entre 4×4 et 512×512 volumes.

```
6.57 s ± 183 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.18 – Compte-rendu de l'estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes.

Cette fois, le problème est plus grave : la méthode diverge pour les trois maillages les plus fins. Si le temps d'exécution est sensiblement équivalent aux deux premières méthodes, directes et GMRES, la méthode BICGSTAB se comporte en fait mal vis-à-vis de l'opérateur implémenté.

1.5 Conclusion de la comparaison des méthodes

Il existe une faille dans ce qui précède. Elle est de l'ordre de l'implémentation, mes tests ne portaient pas uniquement sur la résolution du système linéaire mais également sur l'affichage de l'information de l'avancement de l'algorithme et sur l'affichage des courbes. Ces deux opérations peuvent être pénalisantes selon le langage utilisé et viennent biaiser cette analyse. La même analyse effectuée en supprimant les éléments d'affichage et de calcul d'erreur (on ne conserve que l'assemblage et la résolution de l'équation, pour chaque maillage entre 4×4 ... 512×512, on obtient à nouveau des temps sensiblement équivalents pour la méthode directe et la méthode BICGSTAB, en revanche je ne m'explique pas le bond de durée de la simulation pour la méthode GMRES. On reproduit encore les estimations

```
5.45 s ± 130 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.19 – Compte-rendu de l'estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode directe.

```
32.9 s ± 698 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.20 – Compte-rendu de l'estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode GMRES.

```
5.57 s ± 142 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.21 – Compte-rendu de l'estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode BICGSTAB.

1.6 La question du préconditionnement

De même que les méthodes itératives sont supposées accélérer la convergence des solutions approchées, il est possible de faire un pas supplémentaire dans l'accélération à l'aide des méthodes de l'usage de préconditionnement. Dans cette partie, on reprend l'étude du problème modèle en implémentant un des deux préconditionnements : factorisation **LU**, factorisation **ILU**, ou LU incomplète.

Expérience numérique avec factorisation LU

Le préconditionnement L.U pour les matrices creuses peut s'obtenir, en langage Python, grâce à la classe **LinearOperator** de la sous-librairie **scipy.sparse.linalg**. Une fois l'opérateur creux déclaré, il suffit de continuer par les deux lignes

```
Alu  = scipy.sparse.linalg.splu(A) # Objet contenant une méthode .solve() qui est le préconditionneu
Apre = scipy.sparse.linalg.LinearOperator( ((ny+1)*(nx+1), (ny+1)*(nx+1)), Alu.solve )
```

et la dernière variable est à passer en argument de la fonction de résolution du système

```
U = scipy.sparse.linalg.gmres(A, b.flatten(), M=Apre)[0].reshape((ny+1, nx+1))
```

ou

```
U = scipy.sparse.linalg.bicgstab(A, b.flatten(), M=Apre)[0].reshape((ny+1, nx+1))
```

L'estimation du temps moyen de l'analyse pour le problème de Dirichlet 2.D devient alors, respectivement

On constate un léger avantage de la méthode GMRES pour ce préconditionnement quant à la vitesse d'exécution, mais qu'en est-il des erreurs pour ces deux méthodes ? On donne les couples champ d'erreur et analyse d'abord pour GMRES puis pour BICGSTAB ci-après.

En conclusion, les deux méthodes appelées avec un préconditionnement LU donnent des erreurs $L^2(\Omega)$ sensiblement équivalentes. On constate la convergence à l'ordre 2 en espace et BICGSTAB se montre un rien plus rapide.

```
5.6 s ± 88.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.22 – Estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode GMRES et préconditionnement LU.

```
7.13 s ± 137 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.23 – Estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode BICGSTAB et préconditionnement LU.

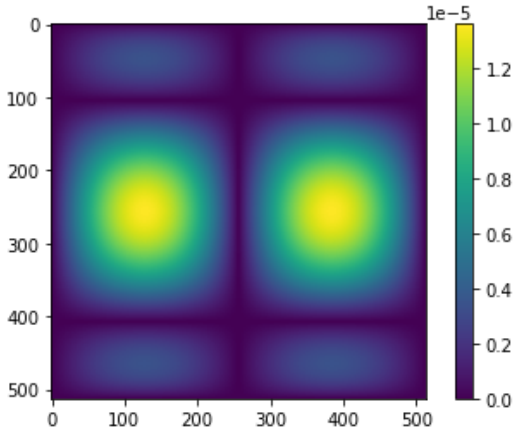

FIGURE 1.24 – Erreur rendue lors de l'exécution de la simulation pour 512×512 volumes avec GMRES / LU.

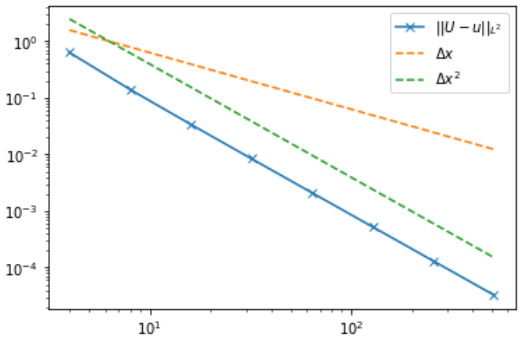

FIGURE 1.25 – Analyse rendue lors de l'exécution de la simulation pour 512×512 volumes avec GMRES / LU.

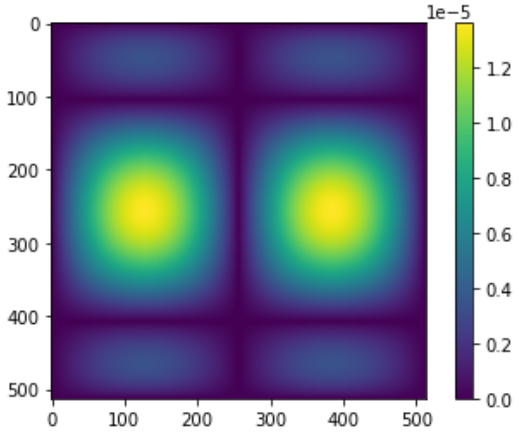

FIGURE 1.26 – Erreur rendue lors de l'exécution de la simulation pour 512×512 volumes avec BICGSTAB / LU.

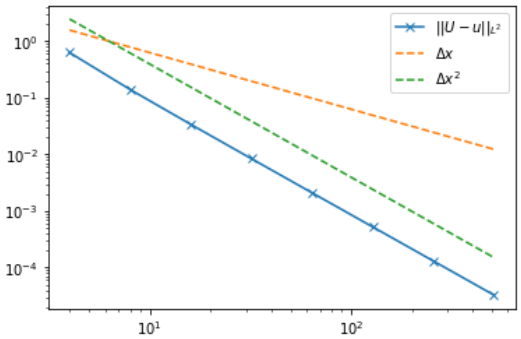

FIGURE 1.27 – Analyse rendue lors de l'exécution de la simulation pour 512×512 volumes avec BICGSTAB / LU.

Expérience numérique avec factorisation ILU

Toujours selon le même protocole, on étudie d'abord l'efficacité de la résolution du système linéaire seul puis dans un second temps on étudie les champs d'erreurs ainsi que l'analyse de la convergence en espace pour chacune des deux méthodes.

```
32.4 s ± 1.28 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.28 – Estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode GMRES et préconditionnement ILU.

```
27.1 s ± 1.09 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FIGURE 1.29 – Estimation de la durée de l'exécution de l'analyse jusqu'à 512×512 volumes par méthode BICGSTAB et préconditionnement ILU.

A nouveau, BICGSTAB se montre meilleur sur la vitesse d'exécution que GMRES. Vient le moment de vérifier les convergences spatiales.

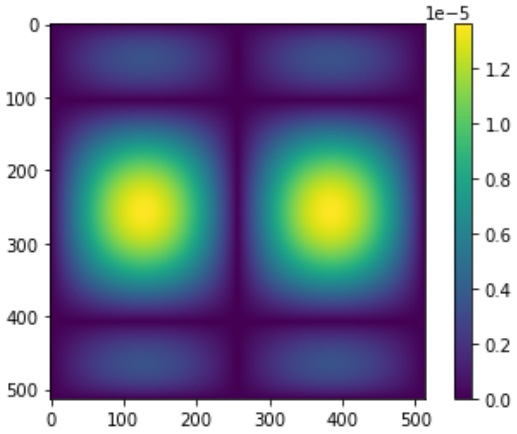

FIGURE 1.30 – Erreur rendue lors de l'exécution de la simulation pour 512×512 volumes avec GMRES / ILU.

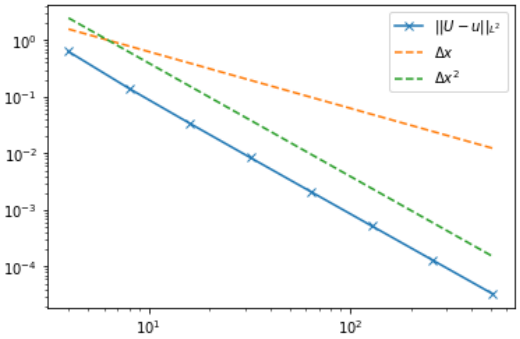

FIGURE 1.31 – Analyse rendue lors de l'exécution de la simulation pour 512×512 volumes avec GMRES / ILU.

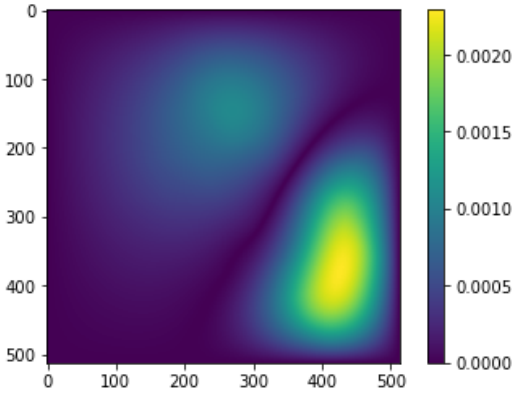

FIGURE 1.32 – Erreur rendue lors de l'exécution de la simulation pour 512×512 volumes avec BICGSTAB / ILU.

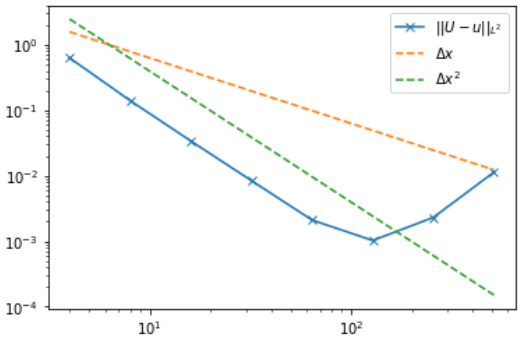

FIGURE 1.33 – Analyse rendue lors de l'exécution de la simulation pour 512×512 volumes avec BICGSTAB / ILU.

Pour une raison que j'ignore, si GMRES parvient parfaitement à obtenir l'ordre 2 en espace attendu, BICGSTAB est mis en échec sur les deux maillages les plus fins. Ainsi, bien qu'il s'exécute plus rapidement que son concurrent GMRES / ILU, le couple BICGSTAB / ILU ne peut convenir pour notre étude.

1.7 Conclusion

Ainsi s'achève le premièr chapitre de ce mémoire, consacré au choix des outils permettant ensuite d'attaquer les études numériques à suivre. Notre choix se portera donc sur le couple GMRES/LU pour la résolution des systèmes linéaires à venir, lesquels seront représentés par des objets creux, pour des raisons de complexité mémoire.

Bien sûr, cette étude est loin d'être complète : on devrait discuter le choix des "métriques" utilisées pour effectuer ce choix. N'a-t-on pas accès au nombre d'itérations plutôt qu'à une commande boîte noire offerte par IPython, grandeur plus objective caractérisant la performance de l'algorithme ? Nous n'avons pas validé le fait que le temps compté est uniquement le temps de résolution et non celui, pourquoi pas prépondérant, d'un appel quelconque n'ayant rien à voir avec les méthodes en elles-même ? Je ne suis pas assez familier de Python pour répondre à toutes ces questions et je présente cela plus comme un exercice de style consistant à montrer qu'il faut y penser que comme une étude aboutie démontrant définitivement que le bon choix est celui annoncé.

Chapitre 2

Analyse & analyse numérique pour le système de Stokes

Ce chapitre est dédié à la compréhension de la physique, de l'analyse puis du développement numérique par les volumes finis des problèmes d'écoulement fluides visqueux. On travaillera en particulier sur le problème de Stokes. La rédaction de l'analyse de cette partie est calquée sur le cours de Yannick Privat et Pascal Frey donné dans le cadre du master de modélisation de Sorbonne-Université lors de l'année 2020 / 2021. Les documents sont disponibles sur la page du cours : https ://www.ljll.math.upmc.fr/frey/nm491.html. La partie numérique se fait à l'aide de la méthode des volumes finis sur maillage marker & cells.

2.1 Modélisation des fluides visqueux

Un fluide est d'abord décrit par la place qu'il occupe dans $\mathbb{R}^d$, on choisira un ouvert connexe borné et régulier, lipschitzien au moins, comme domaine fluide, noté Ω.

L'état d'un fluide formulé en variables eulériennes est décrit par un champ de vitesses $\mathbf{v} : \Omega \to \mathbb{R}^d$ et d'une ou plusieurs grandeurs scalaires : la pression $p : \Omega \to \mathbb{R}$ toujours, et parfois d'autres, selon la phénoméologie du problème. Par exemple, des variables thermodynamiques peuvent être considérées : température, entropie, etc. Dans ce cas le système de Stokes se complète par la définition d'une entropie et le système devient celui de Stokes augmenté de la conservation de cette entropie. Ce mémoire se limite à la description en vitesses et pression.

Les fluides sont caractérisés par une masse volumique $\rho : \Omega \to \mathbb{R}$, exprimée en $kg.m^{-3}$. Dans la suite, nous la considérerons constante. Dans le cas du fluide visqueux, on dispose également d'une viscosité dynamique $\mu : \Omega \to \mathbb{R}$, que nous prendrons constante également.

La dynamique d'un fluide visqueux soumis à une densité de force $\mathbf{f}$ est décrite par le système de Navier-Stokes

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = g$$

est l'équation de continuité, qui s'interprète comme la conservation de la masse et

$$\partial_t(\rho \mathbf{v}) + (\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p - \mu \Delta \mathbf{v} = \rho \mathbf{f} \tag{N-S}$$

décrit la conservation des moments.

2.2 Approximation de Stokes : les faibles nombres de Reynolds

Supposons que $\rho = 1$ et $\mu = 1$. Comme expliqué dans [Cal13], l'équation de Navier-Stokes pour l'écoulement d'un fluide newtonien homogène et incompressible s'écrit, après adimensionnement par ρV_0^2 :

$$\nabla \cdot \mathbf{v} = 0$$
$$\partial_t v(t; \mathbf{x}) + (\mathbf{v} \cdot \nabla)\, \mathbf{v} - \frac{1}{Re} \nabla^2 \mathbf{V} + \nabla p = 0$$

Si on fait tendre le nombre de Reynolds vers 0 alors on trouve à la limite l'équation de Laplace $\nabla^2 \mathbf{v} = 0$, qui est inconsistante avec la condition d'incompressibilité. Par exemple, l'écoulement laminaire incompressible

stationnaire entre deux plans

$$\begin{aligned} \frac{d^2u}{dy^2}(y) &= 0 \\ u(0) &= 0 \\ u(1) &= 0 \end{aligned}$$

donne une solution u identiquement nulle. Ceci résulte d'un mauvais choix pour l'adimensionnement de la pression : l'adimensionnement se fait via le termes d'inertie, négligé dans ce second système, or il est nécessaire de considérer plutôt les forces visqueuses par unité de surface, l'adimensionnement se faisant donc par le facteur $\frac{\mu V_0}{H}$, H étant la hauteur séparant les deux plans.

L'équation de Stokes stationnaire avec conditions de Dirichlet homogène Soit un fluide occupant une région Ω ouverte, connexe, régulière et bornée de $\mathbb{R}^d$ avec $d \in [\![1, 3]\!]$. L'équation de Stokes stationnaire en l'absence de création de masse s'écrit

$$\begin{aligned} \nabla \cdot \mathbf{v} &= 0 \quad && \Omega \\ -\mu \Delta \mathbf{v} + \nabla p &= \rho \mathbf{f} \quad && \Omega \\ u &= 0 \quad && \partial\Omega \end{aligned}$$

où $f : \Omega \to \mathbb{R}^d$. La première est l'équation de conservation de la masse et la seconde l'équation de conservation du moment. Dans le cas où on aurait une fonction source de masse $g : \Omega \to \mathbb{R}$, il faudrait observer une condition de compatibilité supplémentaire. Du fait que

$$\int_\Omega \nabla \cdot \mathbf{v} dx = \int_K g(x) dx$$

le théorème de la divergence donnant l'égalité

$$\int_\Omega (\nabla \cdot \mathbf{v})(x) dx = \int_{\partial\Omega} (\mathbf{v} \cdot \mathbf{n})(\gamma(x)) d\gamma(x)$$

on doit imposer à g la contrainte

$$\int_\Omega g(x) dx = 0$$

Définition d'une solution classique Un couple $\begin{pmatrix} \mathbf{v} \\ p \end{pmatrix}$ est une solution classique du problème de Stokes stationnaire si, et seulement si ce couple appartient à $\left(\mathcal{C}^2(\Omega) \cap \mathcal{C}^0(\bar{\Omega})\right) \times \mathcal{C}^1(\Omega)$

Une solution classique, lorsqu'elle existe, n'est pas unique. En effet, si $\begin{pmatrix} \mathbf{v} \\ p \end{pmatrix}$ est solution, alors pour toute constante $c \in \mathbb{R}$, le couple $\begin{pmatrix} \mathbf{v} \\ p + c \end{pmatrix}$ en est une également. Dans le cas d'une solution classique, il est suffisant de fixer la valeur de p en un point.

Exemples de solutions classiques L'ouvrage [Cal13], chapitre 4 section 2 propose des exemples de solutions exactes pour le problème de Stokes en une dimension.

Par exemple, l'écoulement de Poiseuille plan. Il décrit l'écoulement généré par un gradient horizontal de pression sur un fluide confiné entre deux plans parallèles aux altitudes $y = 0$ et $y = H$. L'écoulement est supposé être laminaire, incompressible, stationnaire et établi. L'hypothèse de laminarité signifique que l'écoulement est parallèle aux parois solides, donc le problème de Stokes se réduit à

$$\mu \partial^2_{y,y} u = \frac{dp}{dy} \tag{P}$$

Cette équation est une E.D.O d'ordre 2, dont les constantes d'intégrations sont déterminées par les vitesses en 0 et H. Si le gradient de pression dans le sens de l'écoulement est noté $\Delta_x p$, constant, alors

$$u : y \mapsto -\frac{H^2}{8\mu} \Delta_x p \left(4\frac{y}{H} - 4\frac{y^2}{H^2}\right)$$

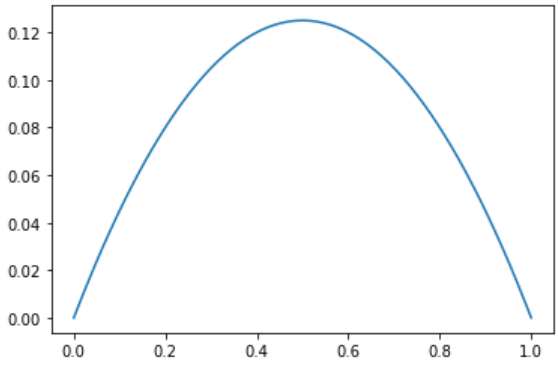

FIGURE 2.1 – En absisse : l'altitude y, en ordonnée, la norme de la vitesse horizontale.

En deux dimensions, on utilisera une variante du triplet de Taylor-Green adapté au système de Stokes. Dans sa version stationnaire, ces fonctions sont données par

$$\begin{aligned} u : x, y &\mapsto sin(x)cos(y) \\ v : x, y &\mapsto -cos(x)sin(y) \\ p &\equiv C \in \mathbb{R} \end{aligned}$$

où u, v sont les deux composantes du vecteur $\mathbf{v}$. Ce triplet vérifie la contrainte $\nabla \cdot \mathbf{v} = 0$ en tout point de Ω. On présente le champ de vecteur pour un tourbillon

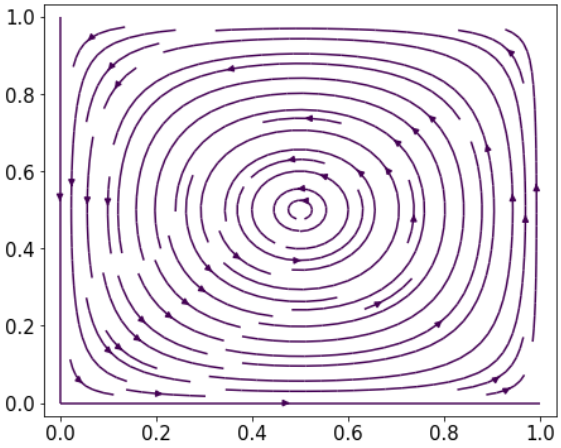

FIGURE 2.2 – Courbes intégrales du champ $\mathbf{v}$ pour un tourbillon

Cadre fonctionnel pour Stokes stationnaire et conditions homogènes en vitesse Soit $\mathcal{D}(\Omega)^d$ l'espace des fonctions $\mathcal{C}_c^\infty(\Omega)^d$ l'espace des fonctions tests. On note $V := H_0^1(\Omega)^d$ muni de sa structure hilbertienne. Notons enfin $Q := \{L_0^2(\Omega) := \{q \in L^2(\Omega), \int_\Omega q(x)dx = 0\}$.

Soit $f \in L^2(\Omega)^d$, soient $\mathbf{v}$, $\mathbf{w} \in V$, multiplions l'équation de conservation des moments et intégrons sur Ω :

$$-\mu \int_\Omega \Delta \mathbf{v} \cdot \mathbf{w} dx + \int_\Omega \nabla p \cdot \mathbf{w} dx = \rho \int_\Omega \mathbf{f} \cdot \mathbf{w} dx$$

La première intégrale vérifie les hypothèses du théorème de la divergence :

$$\int_\Omega \Delta \mathbf{v} \cdot \mathbf{w} dx = \int_\Omega \nabla \mathbf{v} : \nabla \mathbf{w} dx - \int_{\partial\Omega} \partial_{\mathbf{n}} \mathbf{v} \cdot \mathbf{w} d\gamma(x)$$

avec le second terme du membre de droite nul par hypothèse. L'intégrale portant sur le gradient de pression s'intègre également par parties, soit

$$\int_\Omega \nabla p \cdot \mathbf{w} dx = -\int_\Omega p \nabla \cdot \mathbf{w} dx$$

si bien qu'on trouve pour l'équation du moment

$$\mu \int_\Omega \nabla \mathbf{v} : \mathbf{w} dx - \int_\Omega p \nabla \cdot \mathbf{w} = \int_\Omega \mathbf{f} \cdot \mathbf{w} dx$$

L'équation de continuité se traite de la même manière. Choisissons $q \in Q$ et en intégrons sur le domaine Ω, alors une intégration par parties donne l'expression

$$\int_\Omega q \nabla \cdot \mathbf{v} dx = 0$$

Il est possible, grâce à l'identification de $L^2(\Omega)$ avec son dual topologique via le produit scalaire et des propriétés des injections $H_0^1(\Omega)^d \hookrightarrow L^2(\Omega) \equiv L^2(\Omega) \hookrightarrow H^{-1}(\Omega)^d$ de considérer des fonctions $\mathbf{f}$ dans $H^{-1}(\Omega)^d$. Dans ce cas le terme $\int_\Omega \mathbf{f}\mathbf{w} dx$ se lit comme le crochet de dualité $< \mathbf{f}, \mathbf{w} >_{H^{-1}, H_0^1}$.

On introduit les deux formes $a := V \times V \to \mathbb{R}$ et $b : V \times Q \to \mathbb{R}$, définies par les expressions

$$a(\mathbf{v}, \mathbf{w}) := \int_\Omega \nabla \mathbf{v} : \nabla \mathbf{w} dx$$

et

$$b(\mathbf{w}, q) := - \int_\Omega q \nabla \cdot \mathbf{w} dx$$

On introduit enfin $f : V \to \mathbb{R}$,

$$f(\mathbf{w}) := \int_\Omega \mathbf{f} \cdot \mathbf{w} dx$$

Avec ces notations, la formulation variationnelle du problème s'écrit, avec $\mathbf{f} \in H^{-1}(\Omega)^d$,

Trouver $\mathbf{v} \in V, p \in Q$

$$\begin{aligned} a(\mathbf{v}, \mathbf{w}) + b(\mathbf{w}, p) &= f(v) \quad \forall \mathbf{w} \in V \\ b(\mathbf{v}, q) &= 0 \quad \forall q \in Q \end{aligned}$$

On a la proposition suivante

Proposition 2.2.1. *Si f est $L^2(\Omega)$ alors une solution $(\mathbf{v}, p)$ de la formulation variationnelle est solution du problème de Stokes stationnaire.*

La preuve se fait par approximations sur $\mathbf{w}$. Pour tout $\mathbf{w} \in \mathcal{D}(\Omega)$, on a, au sens des distributions

$$< -\Delta \mathbf{v} + \nabla p, \mathbf{w} >_{\mathcal{D}', \mathcal{D}} = \int_\Omega \mathbf{f}\mathbf{w} dx$$

Par densité, pour $\mathbf{w}$ dans V, on choisit une approximation $(\mathbf{w}_n)_n$ de $\mathbf{w}$ pour la norme hilbertienne sur V et on montre alors que

$$-\Delta \mathbf{v} + \nabla p = \mathbf{f}$$

toujours au sens des distributions. Il en va de même pour l'équation de continuité. De plus, les conditions aux limites sont imposées par définition de $V := H_0^1(\Omega)^d$.

Formulation comme point selle d'un lagrangien En guise de remarque préliminaire, on note que Q est un espace réflexif, donc que $Q = Q''$. On introduit alors les opérateurs

$$A : V \to V' \qquad < A\mathbf{v}, \mathbf{w} >_{V', V} = a(\mathbf{v}, \mathbf{w})$$

et

$$B : V \to Q' \qquad < B\mathbf{w}, q >_{Q', Q}$$

On admet alors que trouver une solution au problème variationnel est équivalent à trouver $\mathbf{v} \in V$, $p \in Q$ tel que

$$\begin{aligned} A\mathbf{v} + B^T p &= 0 \\ B\mathbf{v} &= 0 \end{aligned}$$

On introduit donc le noyau de l'opérateur B :

$$ker(B) := \{\mathbf{v} \in V, \forall q \in Q, b(\mathbf{v}, q) = 0\}$$

et on introduit le projecteur

$$\pi_A : ker(B) \to ker(B)'$$

défini par $< \pi_A \mathbf{v}, \mathbf{w} >_{V',V} = < A\mathbf{v}, \mathbf{w} >_{V',V}$ pour tout $\mathbf{v} \in ker(B)$.

On cite le théorème

Théorème 2.2.1. *Avec les notations précédentes, sous les hypothèses suivantes*
- *V est réflexif,*
- *Q est réflexif,*
- *$\mathbf{f}$ est un élément de V',*
- *$a \in \mathcal{L}(V \times V; \mathbb{R})$ est bilinéaire,*
- *$b \in \mathcal{L}(V \times Q; \mathbb{R})$ est bilinéaire,*

alors il existe une unique solution au problème variationnel si, et seulement si,

$$\begin{cases} \exists \alpha > 0 : \inf_{\mathbf{v} \in ker(B)} \sup_{\mathbf{w} \in ker(B)} \frac{a(\mathbf{v},\mathbf{w})}{\|\mathbf{v}\|\|\mathbf{w}\|} \geqslant \alpha \\ \forall \mathbf{w} \in ker(B), (\forall \mathbf{v} \in ker(B), a(\mathbf{v}, \mathbf{w}) \Rightarrow \mathbf{w} = 0 \end{cases}$$

et si

$$\exists \beta > 0 : \inf_{q \in Q} \sup_{\mathbf{v} \in V} \frac{b(\mathbf{v}, q)}{\|\mathbf{v}\|_V \|q\|_Q} \geqslant \beta$$

De plus, on a les estimations a priori

$$\|\mathbf{v}\|_V \leqslant \alpha^{-1} \|f\|_{V'}$$
$$\|p\|_Q \leqslant \beta^{-1} \left(1 + \frac{\|a\|}{\alpha}\right) \|f\|_{V'}$$

Ce théorème, issu du cours de messieurs Frey et Privat, trouve en partie sa démonstration dans [BMR04], chapitre 1.

Définition 2.2.1. *Soient V, Q deux espaces de Banach réflexifs, soit $\mathcal{L}(V \times Q; \mathbb{R})$, alors un couple $(\mathbf{v}, p)$ est un point selle du lagrangien $\mathcal{L}$ si*

$$\forall (\mathbf{w}, q) : \mathcal{L}(\mathbf{v}, q) \leqslant \mathcal{L}(\mathbf{v}, p) \leqslant \mathcal{L}(\mathbf{w}, p)$$

Un point selle se caractérise par la proposition

Proposition 2.2.2. *Un couple $(\mathbf{v}, p)$ est un point selle pour $\mathcal{L}$ si, et seulement si,*

$$\inf_{\mathbf{w} \in V} \sup_{q \in Q} \mathcal{L}(\mathbf{w}, q) = \sup_{q \in Q} \mathcal{L}(\mathbf{v}, q) = \inf_{\mathbf{w} \in V} \mathcal{L}(\mathbf{w}, p) = \sup_{q \in Q} \inf_{\mathbf{w} \in V} \mathcal{L}(\mathbf{w}, q)$$

Dans le cas où a est symétrique et définie positive, on dispose du résultat

Proposition 2.2.3. *Si a est symétrique et définie positive alors $(\mathbf{v}, p)$ est solution du problème variationnel si, et seulement si, ce couple est point selle du lagrangien*

$$\mathcal{L}(\mathbf{w}, q) := \frac{1}{2} a(\mathbf{w}, \mathbf{w}) + b(\mathbf{w}, q) - f(\mathbf{w})$$

qui permet de montrer que dès lors que a est symétrique et définie positive, si les conditions inf sup sont toutes deux réalisées, alors
- le problème variationnel admet une unique solution,
- cette solution est l'unique point selle de la forme $\mathcal{L}$ telle que définie dans la proposition précédente,
- la solution vérifie la caractérisation d'un point selle.

On peut enfin conclure cette section par le résultat

Théorème 2.2.2. *Le problème de Stokes stationnaire avec conditions aux limites de Dirichlet homogène en vitesse admet une unique solution et existe un réel strictement positif $c > 0$ tel que, quel que soit $\mathbf{f} \in H^{-1}(\Omega)^d$*

$$\|\mathbf{v}\|_{H^1(\Omega)^d} + \|p\|_{L^2(\Omega)} \leqslant c \|f\|_{H^{-1}}$$

La preuve se fait grâce aux développements précédents avec $B := \nabla\cdot : H_0^1(\Omega)^d \to L_0^2(\Omega)\,;\, ker(B) := V_0 = \{\mathbf{w} \in H_0^1(\Omega)^d, \nabla\cdot\mathbf{w} = 0\}$

Numériquement, l'imposition de la condition de Dirichlet en un point permettant de fixer la constante pour p ne fait pas sens pour une fonction $L^2(\Omega)$ en général. On discrétisera donc la moyenne de p, dont la valeur est notée M :

$$\bar{p} = M \iff \frac{\Delta y \Delta x}{|\Omega|} \sum p_{j,i} = M$$

dans le cas d'un maillage rectangulaire uniforme.

Autre formulation du problème de Stokes Le problème de Stokes peut également se formuler à l'aide du tenseur des contraintes, ou tenseur de Cauchy du fluide. On le note $\sigma := -\mu\left(\nabla\mathbf{v} + \mathbf{v}^T\right) - p\mathrm{Id}_{\mathbb{R}^d}$. Dans ce cas le système de Stokes s'écrit comme un système de lois de conservations adapté à une discrétisation par méthode de volumes finis

$$\begin{aligned} \nabla\cdot\mathbf{v} &= 0 \\ \nabla\sigma(\mathbf{v}, p) &= \mathbf{f} \end{aligned}$$

2.3 Maillage Marker & Cell

Le maillage M.A.C Le maillage markers & cells est une grille décalée en vitesse et en pression. Par convention, on fixe les noeuds de pression aux bords. Un exemple en une dimension avec noeuds de pression aux bords

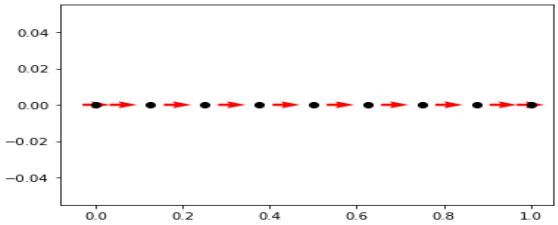

FIGURE 2.3 – En noir, les points de pression, au milieu des flèches rouge les points de vitesse.

En deux dimensions, la cavité $[0,1] \times [0,1]$ discrétisée par un maillage rectangulaire de 8×8 volumes M.A.C est sur la figure suivante.

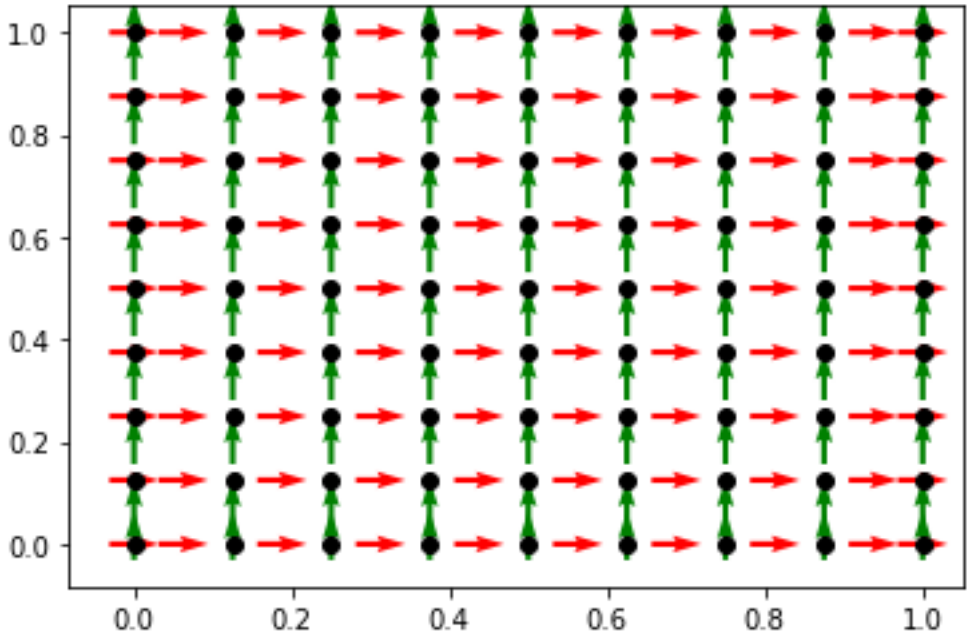

FIGURE 2.4 – En noir, les points de pression, au milieu des flèches vertes, les points de composante verticale de la vitesse et au milieu des flèches rouge les points de composante horizontale de la vitesse.

On dispose de $(n_y+1)(n_x+2)$ points de composante horizontale de vitesse, $(n_y+2)(n_x+1)$ points de composante verticale de vitesse et $(n_y+1)(n_x+1)$ points de pression, leur somme étant le nombre d'inconnues. On les numérote par type et dans l'ordre lexicographique, lisant de gauche à droite, de bas vers le haut. Le vecteur d'état inconnues est posé comme étant

$$X := \begin{pmatrix} U_h \\ V_h \\ P_h \end{pmatrix}$$

Dans la prochaine section, on développe trois codes : deux pour le Laplacien de Dirichlet sur les sous-maillages de vitesse et un Laplacien de Neumann pour le sous-maillage de pression.

Laplacien de Dirichlet sur le maillage de U_h On reprend l'étude du Laplacien avec conditions de Dirichlet sur les bords. Si le schéma à l'intérieur s'écrit de la même manière que celle développée dans le premier chapitre du mémoire, un fait change : les inconnues $U_{h,0,i}$ et $U_{h,n_y,i}$ vérifient directement la condition de Dirichlet, on n'y discrétise pas le Laplacien, en revanche on écrit le Laplacien discret sur les colonnes $U_{h,j,1}$ et U_{h,j,n_x}. Les relations sur les colonnes ne font intervenir que quatre inconnues et imposent une modification du second membre. On présente le code apaté à ce problème

```
from matplotlib import pyplot
import numpy
from numpy import cos, pi, sin
from scipy import sparse
from scipy.sparse.linalg import gmres, splu

# SETUP
u_func  = lambda x, y : sin(x) * cos(y)
Su_func = lambda x, y : 2 * u_func(x, y)

# DISCRETISATION SPATIALE
## Horizontale
xi = 0.
```

```
xf = 2*pi
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 512, 1024])
dx_ = (xf-xi)/nx_

## Verticale
yi = 0.
yf = 2*pi

# INITIALISATION
err = numpy.zeros(nx_.shape)

# ANALYSE DU SCHEMA
for k, nx in enumerate(nx_):

    # Discrétisation spatiale
    dx = dx_[k]
    ny = nx
    print("> Maillage {}x{}".format(ny, nx))
    dy = (yf-yi)/ny
    Ox = numpy.linspace(xi+dx/2, xf-dx/2, nx)
    Ox = numpy.concatenate([[xi], Ox, [xf]], axis=0)
    Oy = numpy.linspace(yi, yf, ny+1)
    X, Y = numpy.meshgrid(Ox, Oy)

    # Construction de l'opérateur discret
    N = 5*(ny-1)*nx + 2*(ny-1) + 2*(nx+2)
    lignes   = numpy.zeros(N)
    colonnes = numpy.zeros(N)
    donnees  = numpy.zeros(N)

    b = (dy*dx) * Su_func(X, Y)
    b[0, :] = u_func(Ox, yi)
    b[-1,:] = u_func(Ox, yf)
    b[:, 0] = u_func(xi, Oy)
    b[:,-1] = u_func(xf, Oy)

    compteur = 0
    for j in range(ny+1):
        for i in range(nx+2):

            if i == 0 or i == nx+1 or j == 0 or j == ny:
                lignes[compteur]   = j * (nx+2) + i
                colonnes[compteur] = j * (nx+2) + i
                donnees[compteur]  = 1
                compteur += 1

            elif i == 1:
                lignes[compteur : compteur+5] = j * (nx+2) + i
                colonnes[compteur]   = j * (nx+2) + i
                colonnes[compteur+1] = j * (nx+2) + i - 1
                colonnes[compteur+2] = j * (nx+2) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+2) + i
                colonnes[compteur+4] = (j+1) * (nx+2) + i
                donnees[compteur]    = 3*(dy/dx) + 2*(dx/dy)
                donnees[compteur+1]  = -2*dy/dx
                donnees[compteur+2]  = -dy/dx
                donnees[compteur+3]  = -dx/dy
                donnees[compteur+4]  = -dx/dy
```

```
                compteur += 5

            elif i == nx:
                lignes[compteur : compteur+5] = j * (nx+2) + i
                colonnes[compteur]   = j * (nx+2) + i
                colonnes[compteur+1] = j * (nx+2) + i - 1
                colonnes[compteur+2] = j * (nx+2) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+2) + i
                colonnes[compteur+4] = (j+1) * (nx+2) + i
                donnees[compteur]    = 3*(dy/dx) + 2*(dx/dy)
                donnees[compteur+1]  = -dy/dx
                donnees[compteur+2]  = -2*dy/dx
                donnees[compteur+3]  = -dx/dy
                donnees[compteur+4]  = -dx/dy
                compteur += 5

            else:
                lignes[compteur : compteur+5] = j * (nx+2) + i
                colonnes[compteur]   = j * (nx+2) + i
                colonnes[compteur+1] = j * (nx+2) + i - 1
                colonnes[compteur+2] = j * (nx+2) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+2) + i
                colonnes[compteur+4] = (j+1) * (nx+2) + i
                donnees[compteur]    = 2 * (dy/dx + dx/dy)
                donnees[compteur+1]  = -dy/dx
                donnees[compteur+2]  = -dy/dx
                donnees[compteur+3]  = -dx/dy
                donnees[compteur+4]  = -dx/dy
                compteur += 5

    A = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+2), (ny+1)*(nx+2))).tocsc(
    Ma = sparse.linalg.LinearOperator(((ny+1)*(nx+2), (ny+1)*(nx+2)), splu(A).solve)

    U = gmres(A, b.flatten(), M=Ma)[0].reshape(ny+1, nx+2)

    u = u_func(X, Y)
    err[k] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)

# FIGURES
pyplot.figure()
pyplot.loglog(nx_, err, 'x-', label=r"$||U-u||_{L^2}$")
pyplot.loglog(nx_, dx_, '--', label=r"$\Delta x$")
pyplot.loglog(nx_, dx_**2, '--', label=r"$\Delta x^2$")
pyplot.legend()
pyplot.show()

pyplot.figure()
pyplot.imshow(numpy.abs(U-u))
pyplot.colorbar()
pyplot.show()
```

avec un champ d'erreurs et la courbe d'erreur de consistance en norme $L^2(\Omega)$

Laplacien de Dirichlet sur le maillage de V_h De même pour le laplacien sur $\mathcal{T}^V$: le schéma au centre est le même, le schéma aux bords verticaux est réduit à $V_{h,j,0} = V^D(x_0, y_{j+1/2})$, $V_{h,j,n_x} = V^D(x_{n_x}, y_{j+1/2})$ et les lignes $j = 1$ et $j = n_y$ font appel à un noeud fantôme et donc à une substitution.

```
from matplotlib import pyplot
import numpy
```

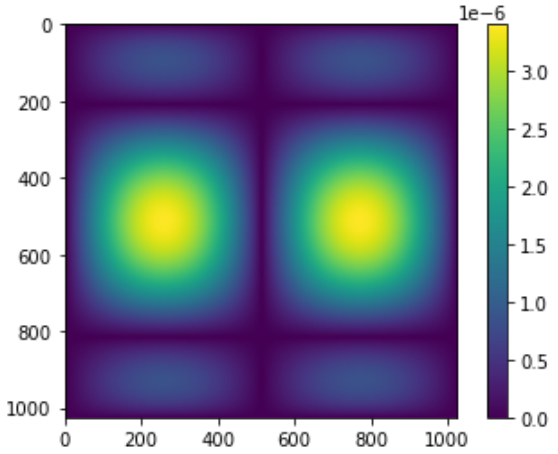

FIGURE 2.5 – Champ d'erreurs pour le Laplacien sur $\mathcal{T}^U$

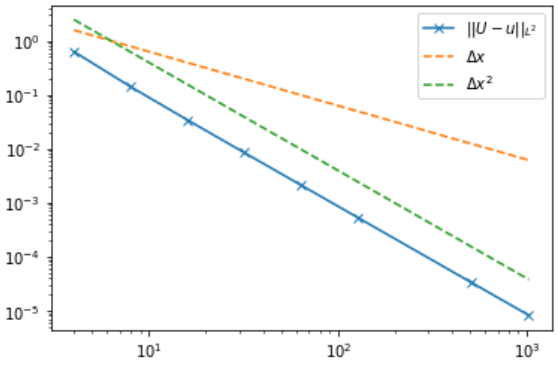

FIGURE 2.6 – Analyse de la norme $L^2(\Omega)$ pour le Laplacien sur $\mathcal{T}^U$

```
from numpy import cos, pi, sin
from scipy import sparse
from scipy.sparse.linalg import gmres, splu

# SETUP
u_func  = lambda x, y : sin(x) * cos(y)
Su_func = lambda x, y : 2*u_func(x, y)

# DISCRETISATION SPATIALE
## Horizontale
xi = 0.
xf = 2*pi
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 256, 512, 1024])
dx_ = (xf-xi)/nx_

## Verticale
yi = 0.
yf = 2*pi

# INITIALISATION
err = numpy.zeros(nx_.shape)

# ANALYSE DU SCHEMA
for k, nx in enumerate(nx_):

    # Discrétisation spatiale
    dx = dx_[k]
    ny = nx
    print("> Maillage {}x{}".format(ny, nx))
    dy = (yf-yi)/ny
    Ox = numpy.linspace(xi, xf, nx+1)
    Oy = numpy.linspace(yi+dy/2, yf-dy/2, ny)
    Oy = numpy.concatenate([[yi], Oy, [yf]], axis=0)
    X, Y = numpy.meshgrid(Ox, Oy)

    # Construction de l'opérateur discret
    N = 5*ny*(nx-1) + 2*(ny+2) + 2*(nx-1)
    lignes   = numpy.zeros(N)
    colonnes = numpy.zeros(N)
    donnees  = numpy.zeros(N)

    b = (dy*dx) * Su_func(X, Y)
    b[0, :] = u_func(Ox, yi)
    b[-1,:] = u_func(Ox, yf)
    b[:, 0] = u_func(xi, Oy)
    b[:,-1] = u_func(xf, Oy)

    compteur = 0
    for j in range(ny+2):
        for i in range(nx+1):

            if i == 0 or i == nx or j == 0 or j == ny+1:
                lignes[compteur]   = j * (nx+1) + i
                colonnes[compteur] = j * (nx+1) + i
                donnees[compteur]  = 1
                compteur += 1

            elif j == 1:
```

```
                lignes[compteur : compteur+5] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                colonnes[compteur+4] = (j+1) * (nx+1) + i
                donnees[compteur]    = 2*(dy/dx) + 3*(dx/dy)
                donnees[compteur+1]  = -dy/dx
                donnees[compteur+2]  = -dy/dx
                donnees[compteur+3]  = -2 * dx/dy
                donnees[compteur+4]  = -dx/dy
                compteur += 5

            elif j == ny:
                lignes[compteur : compteur+5] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                colonnes[compteur+4] = (j+1) * (nx+1) + i
                donnees[compteur]    = 2*(dy/dx) + 3*(dx/dy)
                donnees[compteur+1]  = -dy/dx
                donnees[compteur+2]  = -dy/dx
                donnees[compteur+3]  = -dx/dy
                donnees[compteur+4]  = -2 * dx/dy
                compteur += 5

            else:
                lignes[compteur : compteur+5] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                colonnes[compteur+4] = (j+1) * (nx+1) + i
                donnees[compteur]    = 2 * (dy/dx + dx/dy)
                donnees[compteur+1]  = -dy/dx
                donnees[compteur+2]  = -dy/dx
                donnees[compteur+3]  = -dx/dy
                donnees[compteur+4]  = -dx/dy
                compteur += 5

        A = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+2)*(nx+1), (ny+2)*(nx+1))).tocsc(
        Ma = sparse.linalg.LinearOperator(((ny+1)*(nx+2), (ny+1)*(nx+2)), splu(A).solve)

        U = gmres(A, b.flatten(), M=Ma)[0].reshape(ny+2, nx+1)

        u = u_func(X, Y)
        err[k] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)

    # FIGURES
    pyplot.figure()
    pyplot.loglog(nx_, err, 'x-', label=r"$||U-u||_{L^2}$")
    pyplot.loglog(nx_, dx_, '--', label=r"$\Delta x$")
    pyplot.loglog(nx_, dx_**2, '--', label=r"$\Delta x^2$")
    pyplot.legend()
    pyplot.show()

    pyplot.figure()
```

```
pyplot.imshow(numpy.abs(U-u))
pyplot.colorbar()
pyplot.show()
```

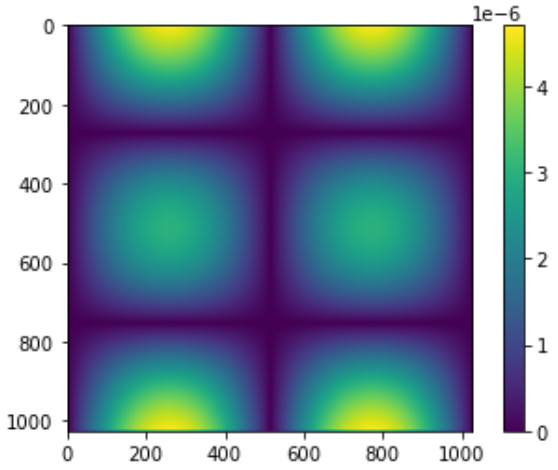

FIGURE 2.7 – Champ d'erreurs pour le Laplacien sur $\mathcal{T}^V$

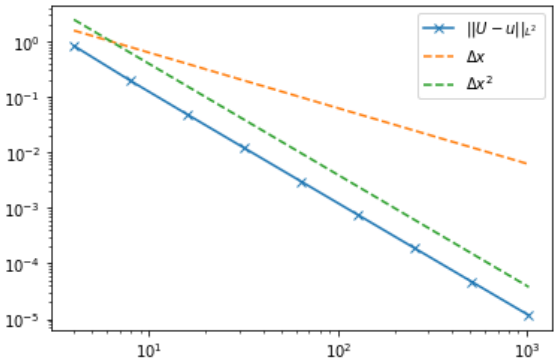

FIGURE 2.8 – Analyse de la norme $L^2(\Omega)$ pour le Laplacien sur $\mathcal{T}^V$

Laplacien de Neumann sur le maillage de pression Cette fois, le problème distingue les noeuds aux coins des noeuds aux bords, des noeuds à l'intérieur. L'équation à l'intérieur est toujours la même, les noeuds aux bords font intervenir quatre points et les noeuds aux coins trois seulement. On approche les conditions de Neumann par une approximation d'ordre 2 de la dérivée de la grandeur P.

```
from matplotlib import pyplot
import numpy
from numpy import cos, pi, sin
from scipy import sparse
from scipy.sparse.linalg import gmres, splu

# SETUP
u_func    = lambda x, y : cos(x) * sin(y)
dudx_func = lambda x, y : -sin(x) * sin(y)
dudy_func = lambda x, y :  cos(x) * cos(y)
Su_func   = lambda x, y : 2 * u_func(x, y)

# DISCRETISATION SPATIALE
## Horizontale
xi = 0.
xf = 2*pi
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 256, 512, 1024])
dx_ = (xf-xi)/nx_

## Verticale
yi = 0.
yf = 2*pi

# INITIALISATION
err = numpy.zeros(nx_.shape)

# ANALYSE DU SCHEMA
for k, nx in enumerate(nx_):

    # Discrétisation spatiale
    dx = dx_[k]
    ny = nx
    print("> Maillage {}x{}".format(ny, nx))
    dy = (yf-yi)/ny
    Ox = numpy.linspace(xi, xf, nx+1)
    Oy = numpy.linspace(yi, yf, ny+1)
    X, Y = numpy.meshgrid(Ox, Oy)

    # Construction de l'opérateur discret
    n = 5*(ny-1)*(nx-1) + 2*4*(ny-1) + 2*4*(nx-1) + 3 * 3 + 1
    lignes   = numpy.zeros(n)
    colonnes = numpy.zeros(n)
    donnees  = numpy.zeros(n)

    b = (dy*dx) * Su_func(X, Y)
    b[0, 0] = u_func(xi, yi)


    compteur = 0
    for j in range(ny+1):
        for i in range(nx+1):

            if i == 0 and j == 0:
                lignes[compteur]   = j * (nx+1) + i
```

```
        colonnes[compteur] = j * (nx+1) + i
        donnees[compteur]  = 1
        compteur += 1

    elif i == 0 and j == ny:
        b[j, i] -= 2*dy*dudx_func(xi, yf)
        b[j, i] += 2*dx*dudy_func(xi, yf)
        lignes[compteur : compteur+3] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i + 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        donnees[compteur]    = 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = - 2 * dy / dx
        donnees[compteur+2]  = - 2 * dx / dy
        compteur += 3

    elif i == nx and j == 0:
        b[j, i] += 2*dy*dudx_func(xf, yi)
        b[j, i] -= 2*dx*dudy_func(xf, yi)
        lignes[compteur : compteur+3] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
        colonnes[compteur+2] = (j+1) * (nx+1) + i
        donnees[compteur]    = 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = - 2 * dy / dx
        donnees[compteur+2]  = - 2 * dx / dy
        compteur += 3

    elif i == nx and j == ny:
        b[j, i] += 2*dy*dudx_func(xf, yf)
        b[j, i] += 2*dx*dudy_func(xf, yf)
        lignes[compteur : compteur+3] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        donnees[compteur]    = 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = - 2 * dy / dx
        donnees[compteur+2]  = - 2 * dx / dy
        compteur += 3

    elif i == 0 and j > 0 and j < ny:
        b[j, i] -= 2 * dy * dudx_func(xi, Oy[j])
        lignes[compteur : compteur+4] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i + 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        colonnes[compteur+3] = (j+1) * (nx+1) + i
        donnees[compteur]    = 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = - 2 * dy/dx
        donnees[compteur+2]  = - dx/dy
        donnees[compteur+3]  = - dx/dy
        compteur += 4

    elif i == nx and j > 0 and j < ny:
        b[j, i] += 2 * dy * dudx_func(xf, Oy[j])
        lignes[compteur : compteur+4] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
```

```
                colonnes[compteur+2] = (j-1) * (nx+1) + i
                colonnes[compteur+3] = (j+1) * (nx+1) + i
                donnees[compteur]    = 2 * (dy/dx + dx/dy)
                donnees[compteur+1]  = - 2 * dy/dx
                donnees[compteur+2]  = - dx/dy
                donnees[compteur+3]  = - dx/dy
                compteur += 4

            elif j == 0 and i > 0 and i < nx:
                b[j, i] -= 2 * dx * dudy_func(Ox[i], yi)
                lignes[compteur : compteur+4] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j+1) * (nx+1) + i
                donnees[compteur]    = 2 * (dy/dx + dx/dy)
                donnees[compteur+1]  = - dy/dx
                donnees[compteur+2]  = - dy/dx
                donnees[compteur+3]  = - 2 * dx/dy
                compteur += 4

            elif j == ny and i > 0 and i < nx:
                b[j, i] += 2 * dx * dudy_func(Ox[i], yf)
                lignes[compteur : compteur+4] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                donnees[compteur]    = 2 * (dy/dx + dx/dy)
                donnees[compteur+1]  = - dy/dx
                donnees[compteur+2]  = - dy/dx
                donnees[compteur+3]  = - 2 * dx/dy
                compteur += 4

            else:
                lignes[compteur : compteur+5] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i - 1
                colonnes[compteur+2] = j * (nx+1) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                colonnes[compteur+4] = (j+1) * (nx+1) + i
                donnees[compteur]    = 2 * (dy/dx + dx/dy)
                donnees[compteur+1]  = -dy/dx
                donnees[compteur+2]  = -dy/dx
                donnees[compteur+3]  = -dx/dy
                donnees[compteur+4]  = -dx/dy
                compteur += 5

        A = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+1), (ny+1)*(nx+1))).tocsc(
        Ma = sparse.linalg.LinearOperator(((ny+1)*(nx+1), (ny+1)*(nx+1)), splu(A).solve)

        U = gmres(A, b.flatten(), M=Ma)[0].reshape(ny+1, nx+1)

        u = u_func(X, Y)
        err[k] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)

    # FIGURES
    pyplot.figure()
```

```
pyplot.loglog(nx_, err, 'x-', label=r"$||U-u||_{L^2}$")
pyplot.loglog(nx_, dx_, '--', label=r"$\Delta x$")
pyplot.loglog(nx_, dx_**2, '--', label=r"$\Delta x^2$")
pyplot.legend()
pyplot.show()

pyplot.figure()
pyplot.imshow(numpy.abs(U-u))
pyplot.colorbar()
pyplot.show()
```

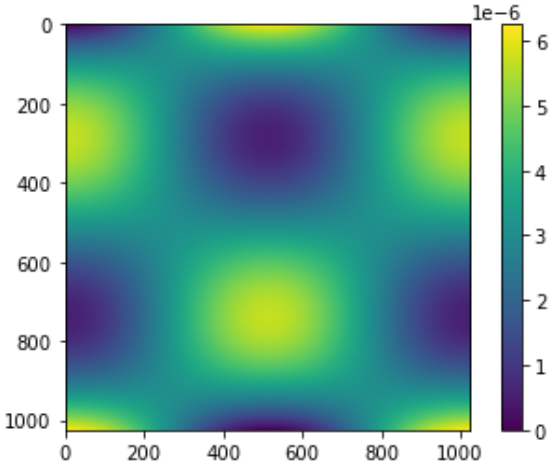

FIGURE 2.9 – Champ d'erreur pour le Laplacien de Neumann, Dirichlet imposé en $(0, 0)$ (haut à gauche de l'image)

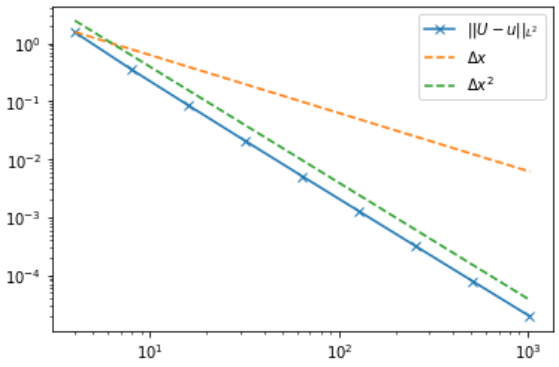

FIGURE 2.10 – Analyse pour le Laplacien de Neumann avec Dirichlet en un point

Discrétisation du système de Stokes On souhaite discrétiser le système de Stokes stationnaire en 2.D posé dans une cavité Ω ouverte, connexe, bornée et lipschitzienne. On introduit le maillage M.A.C $\mathcal{T}$. On suppose que $\mu = 1.$. Soit le système de Stokes

$$\begin{aligned} \partial_x u + \partial_y v &= 0 \\ -\Delta u + \partial_x p &= f^u \\ -\Delta v + \partial_y p &= f^v \end{aligned}$$

La discrétisation en volumes finis de ce système s'effectue en intégrant chaque équation respectivement sur les volumes d'aire $\Delta y \Delta x$ centrés sur des points différents. L'équation de continuité s'intègre sur un volume centré en un point de pression, l'équation de conservation du moment horizontal s'intègre sur un volume centré en $U_{j,i}$ et la conservation du moment vertical s'intègre sur un volume centré en $V_{j,i}$.

Après intégrations par parties et approximation des flux par différences finies d'ordre 1, on trouve pour chaque noeud à l'intérieur :

$$\begin{aligned}
\frac{\Delta y}{\Delta x}\left(U_{j,i+1}-U_{j,i}\right)+\frac{\Delta x}{\Delta y}\left(V_{j+1,i}-V_{j,i}\right)&=0\\
-\frac{\Delta y}{\Delta x}\left(U_{j,i+1}+U_{j,i-1}\right)+-\frac{\Delta x}{\Delta y}\left(U_{j+1,i}+U_{j-1,i}\right)+2\left(\frac{\Delta y}{\Delta x}+\frac{\Delta x}{\Delta y}\right)+&\\
\Delta y\left(P_{j,i}-P_{j,i-1}\right)&=\Delta y\Delta x f^u_{j,i}\\
-\frac{\Delta y}{\Delta x}\left(V_{j,i+1}+V_{j,i-1}\right)+-\frac{\Delta x}{\Delta y}\left(V_{j+1,i}+V_{j-1,i}\right)+2\left(\frac{\Delta y}{\Delta x}+\frac{\Delta x}{\Delta y}\right)+&\\
\Delta x\left(P_{j,i}-P_{j-1,i}\right)&=\Delta y\Delta x f^v_{j,i}
\end{aligned}$$

Comme on l'a vu pour les laplaciens traités précédemment, il convient pour d'éliminer les noeuds inexistants grâce aux conditions aux limites de Dirichlet en vitesse pour la deuxième et l'avant-dernière colonne pour U et la deuxième et avant-dernière ligne pour V. Les points de vitesse sur le bord vérifiant $U_{j,i}=u^D(x_i,y_j)$, $V_{j,i}=v^D(x_j,y_i)$, respectivement sur les sous-maillages U et V - avec des absisses et ordonnées décalées, donc.

Méthode de pénalisation On assemble les inconnues sous la forme

$$X:=\begin{pmatrix}U\\V\\P\end{pmatrix}$$

L'assemblage du système

$$AX=B$$

se fait par blocs

$$A=\begin{pmatrix}A_{uu}&A_{uv}&A_{up}\\A_{vu}&A_{vv}&A_{vp}\\A_{pu}&A_{pv}&A_{pp}\end{pmatrix}$$

Les variables U et V sont découplées donc A_{uv} et A_{vu} sont des blocs nuls. D'autre part, on a les égalités $A_{pu}=A_{up}^T$ et $A_{pv}=A_{vp}^T$.

Si le bloc A_{pp} est laissé nul comme le suggère le système alors le problème est en fait mal posé. La méthode de pénalisation consiste en le choix d'un réel strictement positif ϵ et on propose

$$A_{pp}=\epsilon\mathrm{Id}_{(ny+1)(nx+1)}$$

Dans ce cas le système est résoluble, mais on doit corriger la pression :

$$P=\frac{1}{\epsilon}P$$

Il aurait été utile de vérifier le comportement du schéma en fonction des valeurs de ϵ, chose que je n'ai pas pris le temps de faire. Après quelques échecs d'assemblage et de résolution, il fût décidé que je continue pour passer à la suite et que j'essaie de travailler sur les méthodes de projection scalaire.

2.4 Le système de Stokes dépendant du temps

Système de Stokes dépendant du temps Le problème de Stokes dépendant du temps avec conditions de Dirichlet en vitesse aux bords s'écrit, avec masse volumique et viscosité dynamique constante,

$$\begin{aligned}
\nabla\cdot\mathbf{v}&=0(0,T)\times\Omega\\
\rho\partial_t\mathbf{v}-\mu\Delta\mathbf{v}+\nabla p&=\rho\mathbf{f}(0,T)\times\Omega\\
\mathbf{v}(0,\mathbf{x})&=\mathbf{v}^0(\mathbf{x}\Omega\\
\mathbf{v}_{|\partial\Omega}(t,\mathbf{x})&=0(0,T)\times\partial\Omega
\end{aligned}$$

Cadre fonctionnel pour Stokes dépendant du temps On note

$$H^1_{\int=0}(\Omega) := \{q \in L^2(\Omega), \int_\Omega q(x)dx = 0\}$$

l'espace des fonctions $L^2(\Omega)$ de moyenne nulle. On introduit alors l'espace des fonctions test

$$N(\Omega)^d := \{\mathbf{v} \in \mathcal{C}^\infty_c(\Omega)^d, \nabla \cdot \mathbf{v} = 0\}$$

On note H, respectivement V la fermeture de N dans les espaces $L^2(\Omega)^d$ $H^1_0(\Omega)^d$. Ces deux espaces se caractérisent par

$$H = \{\mathbf{v} \in L^2(\Omega)^d, \nabla \cdot \mathbf{v}, \mathbf{v} \cdot \mathbf{n} = 0\}$$

où $\mathbf{n}$ est la normale sortante de Ω et

$$V = \{\mathbf{v} \in H^1_0(\Omega)^d, \nabla \cdot \mathbf{v} = 0\}$$

L'espace V est dense dans H et l'injection de V dans H est continue. De plus, H s'identifie à son dual. On propose donc

$$V \hookrightarrow H \equiv H' \hookrightarrow V'$$

l'identification étant faite à l'aide du produit scalaire de $L^2(\Omega)^d$.

Formulation contrainte du problème Notons

$$a : H^1_{\int=0} \times H^1_{\int=0} \to \mathbb{R}; \mathbf{u}, \mathbf{v} \mapsto < \nabla \mathbf{u}, \nabla \mathbf{v} >$$

Alors le problème s'écrit, pour tout $\mathbf{f} \in L^2\left((0,T), H^{-1}(\Omega)^d\right)$ et $u_0 \in H$, trouver $\mathbf{v} \in \mathcal{W}(V, V')$ tel que

$$\begin{aligned} < \partial_t \mathbf{v}, \mathbf{w} >_{V',V} + a(\mathbf{v}, \mathbf{w}) =& < \mathbf{f}, \mathbf{w} >_{H^{-1}, H^1_0} p.p(0,T), \forall \mathbf{w} \in V \\ \mathbf{v}(0) =& \mathbf{v}^0 \end{aligned}$$

avec $\mathcal{W}(V,V') := \{\mathbf{v} : (0,T) \to V, \mathbf{v} \in L^2((0,T),V), \partial_t \mathbf{v} \in L^2((0,T),V')\}$ Sous cette forme, on peut montrer que, quel que soit $T > 0$, il existe une unique solution au problème. On peut s'inspirer de la preuve de l'existence globale des solutions de Leray présentée dans le chapitre 3 du cours d'Anne-Laure DALIBARD, première partie (disponible ici), seul le premier lemme est ici concerné.

Une formulation mixte On introduit une nouvelle forme, linéaire, $b : \mathcal{L}(H^1_0(\Omega)^d \times L^2_0(\Omega); \mathbb{R}); \mathbf{w}, q \mapsto (-\nabla \cdot \mathbf{w}, q)_{L^2(\Omega)}$ et alors résoudre le problème de Stokes dépendant du temps est équivalent à trouver $\mathbf{v} \in \mathcal{W}(H^1_0(\Omega)^d, H^{-1}(\Omega)^d)$ et $p \in L^2((0,T), L^2_0(\Omega)$ tels que

$$\begin{aligned} (\partial_t \mathbf{v}, \mathbf{w})_{L^2} + b(\mathbf{v}, p) + a(\mathbf{v}, \mathbf{w}) =& (\mathbf{f}, \mathbf{w})_{L^2} & p.p(0,T), \forall \mathbf{w} \in H^1_0(\Omega)^d \\ \nabla \cdot \mathbf{v} q =& 0 & \forall q \in L^2_0(\Omega) \mathbf{v}(0) = \mathbf{v}^0 \end{aligned}$$

Quel que soit T strictement positif, il existe une unique solution du problème.

2.5 Analyse numérique pour le problème de Stokes dépendant du temps

Exemple de l'équation de réaction-diffusion avec Dirichlet aux bords On développe un code adapté à la résolution de

$$\begin{aligned} \partial_t u - \Delta u &= S_u \\ u_{|\partial\Omega} &= u^D \\ u(0,\cdot) &= u^0 \end{aligned}$$

On se sert de la fonction analytique suivante

$$u(t,x,y) = sin(x)cos(y)e^{-2t}$$

. Alors l'équation est satisfaite pour le second membre $S_u \equiv 0$.

```
from matplotlib import pyplot
import numpy
from numpy import cos, exp, pi, sin
import scipy
from scipy import sparse
from scipy.sparse import linalg

# SETUP
u_func = lambda t, x, y : sin(x) * cos(y) * exp(-2*t)
Su_func = lambda t, x, y : numpy.zeros(x.shape)

# DISCRETISATION TEMPORELLE
ti = 0.
nt = 1000
dt = .001
tf = ti + nt*dt
T  = tf-ti

# DISCRETISATION SPATIALE
## Horizontale
xi = 0.
xf = 2*pi
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 256])
dx_ = (xf-xi)/nx_

## Verticale
yi = 0.
yf = 2*pi

# INITIALISATION
err = numpy.zeros(nx_.shape)

# ITERATION SPATIALE
for k, nx in enumerate(nx_):

    # Discrétisation spatiale
    dx = dx_[k]
    Ox = numpy.linspace(xi, xf, nx+1)
    ny = nx
    dy = (yf-yi)/ny
    Oy = numpy.linspace(yi, yf, ny+1)
    X, Y = numpy.meshgrid(Ox, Oy)

    # Construction de l'opérateur discret
    n = 5 * (ny*nx) + 2*(ny+1) + 2*(nx+1)
    lignes   = numpy.zeros(n)
    colonnes = numpy.zeros(n)
    donnees  = numpy.zeros(n)
    compteur = 0

    for j in range(ny+1):
        for i in range(nx+1):

            if i == 0 or i == nx or j == 0 or j == ny:
                lignes[compteur]   = j * (nx+1) + i
                colonnes[compteur] = j * (nx+1) + i
                donnees[compteur]  = 1
                compteur += 1
```

```
            else:
                lignes[compteur : compteur+5] = j * (nx+1) + i
                colonnes[compteur]   = j * (nx+1) + i
                colonnes[compteur+1] = j * (nx+1) + i + 1
                colonnes[compteur+2] = j * (nx+1) + i - 1
                colonnes[compteur+3] = (j-1) * (nx+1) + i
                colonnes[compteur+4] = (j+1) * (nx+1) + i
                donnees[compteur] = 2 * (dy/dx + dx/dy) + dx*dy/dt
                donnees[compteur+1] = -dy/dx
                donnees[compteur+2] = -dy/dx
                donnees[compteur+3] = -dx/dy
                donnees[compteur+4] = -dx/dy
                compteur += 5

    A = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+1), (ny+1)*(nx+1))).tocsc(
    Ma = linalg.LinearOperator(((ny+1)*(nx+1), (ny+1)*(nx+1)), linalg.splu(A).solve)

    # Initialisation
    U = u_func(0, X, Y)

    # Itération temporelle
    for n in range(nt+1):
        # info
        if n%10 == 0: print("> grille {}x{}, itération {}/{}".format(ny, nx, n, nt))

        # date courante
        t = ti + (n+1)*dt

        # construction du second membre
        b = dy*dx * Su_func(t, X, Y) + dy*dx/dt * U
        b[0, :] = u_func(t, Ox, yi)
        b[-1,:] = u_func(t, Oy, yf)
        b[:, 0] = u_func(t, xi, Oy)
        b[:,-1] = u_func(t, xf, Oy)

        # Résolution
        U = linalg.gmres(A, b.flatten(), M=Ma)[0].reshape((ny+1, nx+1))

    u = u_func(tf, X, Y)
    err[k] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)

# FIGURES
pyplot.suptitle(r"$\epsilon := ||U_h(t_f, \cdot, \cdot) - u_h(t_f, \cdot, \cdot)||$ ; $\Delta t$={},
pyplot.loglog(nx_, err, 'x-', label="$\ln(\epsilon)$")
pyplot.loglog(nx_, dx_, label="$\Delta x$")
pyplot.loglog(nx_, dx_**2, label="$\Delta x^2$")
pyplot.legend()
pyplot.xlabel(r"$N_x$")
pyplot.show()

pyplot.figure()
pyplot.imshow(numpy.abs(U-u))
pyplot.colorbar()
pyplot.show()
```

On fait tourner la simulation sur trois couples $(\Delta t, n_t)$.

On trouve bien que l'erreur rencontre un plancher autour de Δt, c'est la conséquence du premier ordre en temps du schéma d'Euler implicite. D'autres méthodes peuvent être employées, les méthodes

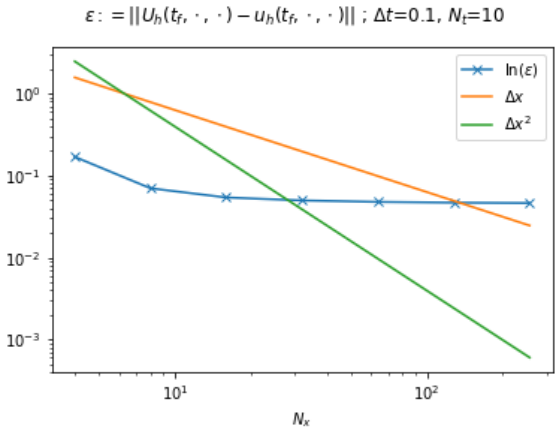

FIGURE 2.11

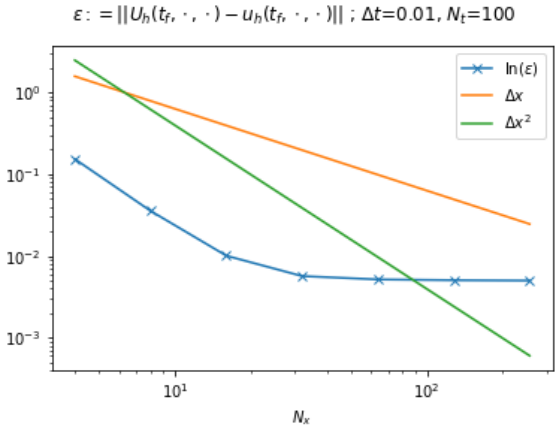

FIGURE 2.12

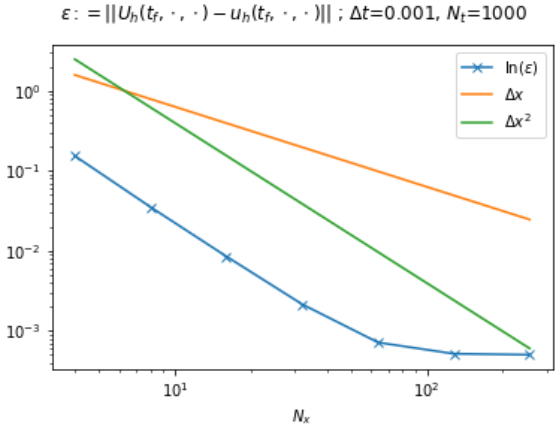

FIGURE 2.13

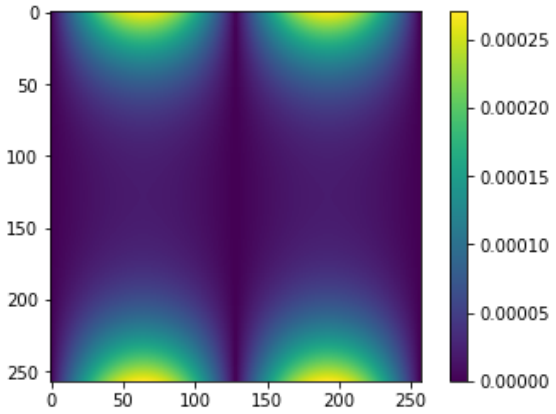

FIGURE 2.14 – Champ d'erreur au temps final pour $\Delta t = 10^{-3}$

explicites imposant une condition de Courant–Friedrichs–Lewy pour assurer leur stabilité. Dans ce genre de méthode, on prescrit cette condition et on calcule Δt en fonction de Δx et de la C.F.L. Une bonne pratiqe est d'implémenter la discrétisation spatiale en premier lieu.

Une méthode de projection Les méthodes de projections pour les fluides incompressibles ont été synthétisées dans l'article [GMS06], section 3.1. Leur intérêt réside dans l'obtention d'une suite d'équations elliptiques indépendantes, là où les variables vitesses et pression sont couplées dans le système de Stokes.

La décomposition de Helmholtz de l'espace $L^2(\Omega)^d$ s'écrit avec

$$H := \{\mathbf{v} \in L^2(\Omega)^d, \nabla \cdot \mathbf{v} = 0, \mathbf{v} \cdot \mathbf{n}_{|\partial\Omega} = 0\}$$

et

$$G := \{\mathbf{v} \in L^2(\Omega)^d, \mathbf{v} = \nabla\phi, \phi \in H^1(\Omega)/\mathbb{R}\}$$

Pour chaque $\mathbf{v}$, il existe un unique couple $\mathbf{v}_\phi, \mathbf{v}_\psi$ tel que $\mathbf{v} = \mathbf{v}_\phi + \mathbf{v}_\psi$ et les vecteurs du membre de droite sont orthogonaux.

Dans la décomposition précédente, $\mathbf{v}_\phi = \nabla\phi \in G$ et $\mathbf{v}_\psi = \nabla \times \psi \in H$. Le domaine étant connexe, il est simplement connexe et donc $\nabla\psi = 0$.

Pour un potentiel scalaire ϕ, on a le problème de Poisson donné avec conditions de Neumann au bord

$$\begin{aligned}\Delta\phi &= \nabla\mathbf{v} \\ \nabla\phi \cdot \mathbf{n}_{|\partial\Omega} &= \mathbf{v} \cdot \mathbf{n}_{|\partial\Omega}\end{aligned}$$

et alors $\mathbf{v}_\phi = \nabla\phi$ et $\mathbf{v}_\psi := \mathbf{v} - \mathbf{v} - \nabla\phi$

L'algotithme commence par un étape de prédiction, on résoud

$$\begin{aligned}\frac{\tilde{\mathbf{v}} - \mathbf{v}^{(n)}}{\Delta t} - \mu\Delta\tilde{\mathbf{v}}^{(n+1)} + \nabla p^{(n)} &= \mathbf{f}^{(n+1)} \\ \tilde{\mathbf{v}}^{(n+1)} &= \mathbf{v}^D\end{aligned}$$

L'algorithme se poursuit par l'étape de projection, on résoud cette fois

$$\begin{aligned}\nabla \cdot \nabla\phi^{(n+1)} &= \nabla \cdot \tilde{\mathbf{v}}^{(n+1)} \\ \nabla\phi^{(n+1)} \cdot \mathbf{n}_{|\partial\Omega} &= 0\end{aligned}$$

et il se termine par la correction de la vitesse, on calcule

$$\begin{aligned}\mathbf{v}^{(n+1)} &= \tilde{\mathbf{v}}^{(n+1)} - \Delta t\nabla\phi^{(n+1)} \\ \Phi^{(n+1)} &= P^{(n+1)} - P^{(n)}\end{aligned}$$

On dispose du théorème

Théorème 2.5.1. *Soit $(\mathbf{v}, p)$ une solution du problème de Stokes dépendant du temps, et lisse. Alors la solution $(\mathbf{v}_h, p_h)$ de l'algorithme proposé vérifie*

$$\begin{aligned}\|\mathbf{v}_{\Delta t} - \mathbf{v}_{h,\Delta t}\|_{l^\infty(L^2(\Omega)} + \|\mathbf{v}_{\Delta t} - \tilde{\mathbf{v}}_{h,\Delta t} &\leqslant c(\mathbf{u}, p, T)\Delta t \\ \|p_{\Delta t} - p_{h,\Delta t}\|_{l^\infty(L^2(\Omega)} + \|\mathbf{v}_{\Delta t} - \tilde{\mathbf{v}}_{h,\Delta t}\|_{l^\infty(H^1(\Omega)} &\leqslant c(\mathbf{v}, p, T)\sqrt{\Delta t}\end{aligned}$$

Implémentation de projection scalaire pour Stokes 2.D On étudie l'évolution d'un tourbillon de Taylor-Green initialisé à $t = 0$.

```
from matplotlib import pyplot
import numpy
from numpy import cos, exp, pi, sin
from scipy import sparse
from scipy.sparse.linalg import gmres, splu

# PHASE LIQUIDE
rho = 1.
mu  = 1.

# SETUP
u_func  = lambda t, x, y :  sin(x)*cos(y)*exp(-2*t)
v_func  = lambda t, x, y : -cos(x)*sin(y)*exp(-2*t)
p_func  = lambda t, x, y :  numpy.zeros(x.shape)
dudx_func = lambda t, x, y :  cos(x)*cos(y)*exp(-2*t)
dvdy_func = lambda t, x, y : -cos(x)*cos(y)*exp(-2*t)
dpdx_func = lambda t, x, y :  numpy.zeros(x.shape)
dpdy_func = lambda t, x, y :  numpy.zeros(x.shape)
Su_func = lambda t, x, y :  numpy.zeros(x.shape)
Sv_func = lambda t, x, y :  numpy.zeros(x.shape)

# DISCRETISATION TEMPORELLE
ti = 0.
nt = 1000
dt = .001
tf = ti + nt*dt

# DISCRETISATION SPATIALE
xi  = 0.
xf  = 2*pi
nx_ = numpy.array([4, 8, 16, 32, 64, 128, 256])
dx_ = numpy.zeros(nx_.shape)

# INITIALISATION
## err[k, 0] : erreur sur la variable U pour nx_[k]**2 volumes
## err[k, 1] : erreur sur la variable V pour nx_[k]**2 volumes
## err[k, 2] : erreur sur la variable P pour nx_[k]**2 volumes
## err[k, 3] : erreur sur la variable D pour nx_[k]**2 volumes ; D pour div
err = numpy.zeros((nx_.shape[0], 4))

# ANALYSE
for k, nx in enumerate(nx_):

    # Discrétisation spatiale
    ## horizontale
    dx = (xf-xi)/nx
    dx_[k] = dx
    x_c = numpy.concatenate([[xi], numpy.linspace(xi+dx/2, xf-dx/2, nx), [xf]], axis=0)
    x_f = numpy.linspace(xi, xf, nx+1)
    ## verticale
    yi = xi
    yf = xf
    ny = nx
    dy = (yf-yi)/ny
    y_c = numpy.concatenate([[yi], numpy.linspace(yi+dx/2, yf-dx/2, ny), [yf]], axis=0)
    y_f = numpy.linspace(yi, yf, ny+1)
```

```
## 2.D
Xu, Yu = numpy.meshgrid(x_c, y_f)
Xv, Yv = numpy.meshgrid(x_f, y_c)
Xp, Yp = numpy.meshgrid(x_f, y_f)

# Initialisation
U = u_func(ti, Xu, Yu)
V = v_func(ti, Xv, Yv)
P = numpy.zeros(Xp.shape)
## construction de la divergence initiale
D = (U[:,1:]-U[:,:-1])/dx + (V[1:,:]-V[:-1,:])/dy
D[0, 1:-1]  = (U[0,2:-1]-U[0,1:-2])/dx + 2/dy*(V[1,1:-1]-V[0,1:-1])
D[-1, 1:-1] = (U[-1,2:-1]-U[-1,1:-2])/dx + 2/dy*(V[-1,1:-1]-V[-2,1:-1])
D[1:-1, 0]  = 2/dx*(U[1:-1,1]-U[1:-1,0]) + (V[2:-1,0]-V[1:-2,0])/dy
D[1:-1,-1]  = 2/dx*(U[1:-1,-1]-U[1:-1,-2]) + (V[2:-1,-1]-V[1:-2,-1])/dy
D[0, 0]  = 2/dx*(U[0,1]-U[0,0]) + 2/dy*(V[1,0]-V[0,0])
D[-1, 0] = 2/dx*(U[0,1]-U[0,0]) + 2/dy*(V[-1,0]-V[-2,0])
D[-1,-1] = 2/dx*(U[0,-1]-U[0,-2]) + 2/dy*(V[-1,0]-V[-2,0])
D[0,-1]  = 2/dx*(U[0,-1]-U[0,-2]) + 2/dy*(V[1,0]-V[0,0])

if nx == nx_[-1]:
    ## contrôle de la U initiale
    pyplot.imshow(U)
    pyplot.colorbar()
    pyplot.suptitle(r"$U({}, \cdot, \cdot)$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".format(ti, ny
    pyplot.show()
    ## contrôle de la vitesse V initiale
    pyplot.imshow(V)
    pyplot.colorbar()
    pyplot.suptitle(r"$V({}, \cdot, \cdot)$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".format(ti, ny
    pyplot.show()
    ## contrôle de la divergence initiale
    pyplot.imshow(P)
    pyplot.colorbar()
    pyplot.suptitle(r"$P({}, \cdot, \cdot)$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".format(ti, ny
    pyplot.show()
    ## contrôle de la divergence initiale
    pyplot.imshow(numpy.abs(D))
    pyplot.colorbar()
    pyplot.suptitle(r"$|div(u, v)({}, \cdot, \cdot)|$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".form
    pyplot.show()

# Construction des opérateurs creux
## prédicteur de la vitesse U
N = 5*(ny-1)*nx + 2*(ny-1) + 2*(nx+2)
lignes   = numpy.zeros(N)
colonnes = numpy.zeros(N)
donnees  = numpy.zeros(N)

compteur = 0
for j in range(ny+1):
    for i in range(nx+2):

        if i == 0 or i == nx+1 or j == 0 or j == ny:
            lignes[compteur]   = j * (nx+2) + i
            colonnes[compteur] = j * (nx+2) + i
            donnees[compteur]  = 1
            compteur += 1
```

```
        elif i == 1:
            lignes[compteur : compteur+5] = j * (nx+2) + i
            colonnes[compteur]   = j * (nx+2) + i
            colonnes[compteur+1] = j * (nx+2) + i - 1
            colonnes[compteur+2] = j * (nx+2) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+2) + i
            colonnes[compteur+4] = (j+1) * (nx+2) + i
            donnees[compteur]    = 3*(dy/dx) + 2*(dx/dy) + (dy*dx)/dt
            donnees[compteur+1]  = -2*dy/dx
            donnees[compteur+2]  = -dy/dx
            donnees[compteur+3]  = -dx/dy
            donnees[compteur+4]  = -dx/dy
            compteur += 5

        elif i == nx:
            lignes[compteur : compteur+5] = j * (nx+2) + i
            colonnes[compteur]   = j * (nx+2) + i
            colonnes[compteur+1] = j * (nx+2) + i - 1
            colonnes[compteur+2] = j * (nx+2) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+2) + i
            colonnes[compteur+4] = (j+1) * (nx+2) + i
            donnees[compteur]    = 3*(dy/dx) + 2*(dx/dy) + (dy*dx)/dt
            donnees[compteur+1]  = -dy/dx
            donnees[compteur+2]  = -2*dy/dx
            donnees[compteur+3]  = -dx/dy
            donnees[compteur+4]  = -dx/dy
            compteur += 5

        else:
            lignes[compteur : compteur+5] = j * (nx+2) + i
            colonnes[compteur]   = j * (nx+2) + i
            colonnes[compteur+1] = j * (nx+2) + i - 1
            colonnes[compteur+2] = j * (nx+2) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+2) + i
            colonnes[compteur+4] = (j+1) * (nx+2) + i
            donnees[compteur]    = 2 * (dy/dx + dx/dy) + (dy*dx)/dt
            donnees[compteur+1]  = -dy/dx
            donnees[compteur+2]  = -dy/dx
            donnees[compteur+3]  = -dx/dy
            donnees[compteur+4]  = -dx/dy
            compteur += 5

Auu = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+2), (ny+1)*(nx+2))).tocs
Mau = sparse.linalg.LinearOperator(((ny+1)*(nx+2), (ny+1)*(nx+2)), splu(Auu).solve)

# Construction des opérateurs creux et des seconds membres
## prédicteur de la vitesse V
N = 5*ny*(nx-1) + 2*(ny+2) + 2*(nx-1)
lignes   = numpy.zeros(N)
colonnes = numpy.zeros(N)
donnees  = numpy.zeros(N)

compteur = 0
for j in range(ny+2):
    for i in range(nx+1):

        if i == 0 or i == nx or j == 0 or j == ny+1:
```

```
            lignes[compteur]   = j * (nx+1) + i
            colonnes[compteur] = j * (nx+1) + i
            donnees[compteur]  = 1
            compteur += 1

        elif j == 1:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = 2*(dy/dx) + 3*(dx/dy) + (dy*dx)/dt
            donnees[compteur+1]  = -dy/dx
            donnees[compteur+2]  = -dy/dx
            donnees[compteur+3]  = -2 * dx/dy
            donnees[compteur+4]  = -dx/dy
            compteur += 5

        elif j == ny:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = 2*(dy/dx) + 3*(dx/dy) + (dy*dx)/dt
            donnees[compteur+1]  = -dy/dx
            donnees[compteur+2]  = -dy/dx
            donnees[compteur+3]  = -dx/dy
            donnees[compteur+4]  = -2 * dx/dy
            compteur += 5

        else:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = 2 * (dy/dx + dx/dy) + (dy*dx)/dt
            donnees[compteur+1]  = -dy/dx
            donnees[compteur+2]  = -dy/dx
            donnees[compteur+3]  = -dx/dy
            donnees[compteur+4]  = -dx/dy
            compteur += 5

Avv = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+2)*(nx+1), (ny+2)*(nx+1))).tocs
Mav = sparse.linalg.LinearOperator(((ny+2)*(nx+1), (ny+2)*(nx+1)), splu(Avv).solve)

# Construction des opérateurs creux
## Poisson pour la pression
N = 5*(ny-1)*(nx-1) + 2*4*(ny-1) + 2*4*(nx-1) + 3 * 3 + 1
lignes   = numpy.zeros(N)
colonnes = numpy.zeros(N)
donnees  = numpy.zeros(N)

compteur = 0
```

```
for j in range(ny+1):
    for i in range(nx+1):

        if i == 0 and j == 0:
            lignes[compteur]   = j * (nx+1) + i
            colonnes[compteur] = j * (nx+1) + i
            donnees[compteur]  = 1
            compteur += 1

        elif i == 0 and j == ny:
            lignes[compteur : compteur+3] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i + 1
            colonnes[compteur+2] = (j-1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = 2 * dy / dx
            donnees[compteur+2]  = 2 * dx / dy
            compteur += 3

        elif i == nx and j == 0:
            lignes[compteur : compteur+3] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = 2 * dy / dx
            donnees[compteur+2]  = 2 * dx / dy
            compteur += 3

        elif i == nx and j == ny:
            lignes[compteur : compteur+3] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = (j-1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = 2 * dy / dx
            donnees[compteur+2]  = 2 * dx / dy
            compteur += 3

        elif i == 0 and j > 0 and j < ny:
            lignes[compteur : compteur+4] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i + 1
            colonnes[compteur+2] = (j-1) * (nx+1) + i
            colonnes[compteur+3] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = 2 * dy/dx
            donnees[compteur+2]  = dx/dy
            donnees[compteur+3]  = dx/dy
            compteur += 4

        elif i == nx and j > 0 and j < ny:
            lignes[compteur : compteur+4] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = (j-1) * (nx+1) + i
            colonnes[compteur+3] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
```

```
            donnees[compteur+1]  = 2 * dy/dx
            donnees[compteur+2]  = dx/dy
            donnees[compteur+3]  = dx/dy
            compteur += 4

        elif j == 0 and i > 0 and i < nx:
            lignes[compteur : compteur+4] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = dy/dx
            donnees[compteur+2]  = dy/dx
            donnees[compteur+3]  = 2 * dx/dy
            compteur += 4

        elif j == ny and i > 0 and i < nx:
            lignes[compteur : compteur+4] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = dy/dx
            donnees[compteur+2]  = dy/dx
            donnees[compteur+3]  = 2 * dx/dy
            compteur += 4

        else:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = dy/dx
            donnees[compteur+2]  = dy/dx
            donnees[compteur+3]  = dx/dy
            donnees[compteur+4]  = dx/dy
            compteur += 5

App = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+1), (ny+1)*(nx+1))).tocs
Map = sparse.linalg.LinearOperator(((ny+1)*(nx+1), (ny+1)*(nx+1)), splu(App).solve)

# Itération temporelle
for n in range(nt):

    # Info
    if n % 10 == 0: print("> {}x{} volumes, itération {}/{}".format(ny, nx, n, nt))

    # Date
    t = ti + (n+1)*dt

    # Prédiction de la vitesse U
    ## second membre
    bu = (dx*dy) * Su_func(t, Xu, Yu) + (dx*dy)/dt * U
```

```
        bu[0, :] = u_func(t, x_c, yi)
        bu[-1,:] = u_func(t, x_c, yf)
        bu[:, 0] = u_func(t, xi, y_f)
        bu[:,-1] = u_func(t, xf, y_f)

        ## résolution
        U = gmres(Auu, bu.flatten(), M=Mau)[0].reshape((ny+1, nx+2))

        # Prédiction de la vitesse V
        ## second membre
        bv = (dx*dy) * Sv_func(t, Xv, Yv) + (dx*dy)/dt * V
        bv[0, :] = v_func(t, x_f, yi)
        bv[-1,:] = v_func(t, x_f, yf)
        bv[:, 0] = v_func(t, xi, y_c)
        bv[:,-1] = v_func(t, xf, y_c)

        ## résolution
        V = gmres(Avv, bv.flatten(), M=Mav)[0].reshape((ny+2, nx+1))

        # Construction de la divergence
        D = (U[:,1:]-U[:,:-1])/dx + (V[1:,:]-V[:-1,:])/dy
        D[0, 1:-1]  = (U[0,2:-1]-U[0,1:-2])/dx + 2/dy*(V[1,1:-1]-V[0,1:-1])
        D[-1, 1:-1] = (U[-1,2:-1]-U[-1,1:-2])/dx + 2/dy*(V[-1,1:-1]-V[-2,1:-1])
        D[1:-1, 0]  = 2/dx*(U[1:-1,1]-U[1:-1,0]) + (V[2:-1,0]-V[1:-2,0])/dy
        D[1:-1,-1]  = 2/dx*(U[1:-1,-1]-U[1:-1,-2]) + (V[2:-1,-1]-V[1:-2,-1])/dy
        D[0, 0]  = 2/dx*(U[0,1]-U[0,0]) + 2/dy*(V[1,0]-V[0,0])
        D[-1, 0] = 2/dx*(U[0,1]-U[0,0]) + 2/dy*(V[-1,0]-V[-2,0])
        D[-1,-1] = 2/dx*(U[0,-1]-U[0,-2]) + 2/dy*(V[-1,0]-V[-2,0])
        D[0,-1]  = 2/dx*(U[0,-1]-U[0,-2]) + 2/dy*(V[1,0]-V[0,0])

        # Projection, résolution de Phi
        Phi = gmres(App, (dx*dy)*D.flatten(), M=Map)[0].reshape((ny+1, nx+1))

        # Correction des vitesses U, V et de la pression P
        U[1:-1, 1:-1] = U[1:-1, 1:-1] - dt * (Phi[1:-1, 1:]-Phi[1:-1, :-1])/dx
        V[1:-1, 1:-1] = V[1:-1, 1:-1] - dt * (Phi[1:, 1:-1]-Phi[:-1, 1:-1])/dy
        P = P + dt * Phi

    # Solution analytiques
    u = u_func(t, Xu, Yu)
    v = v_func(t, Xv, Yv)
    p = p_func(t, Xp, Yp)
    d = dudx_func(t, Xp, Yp) + dvdy_func(t, Xp, Yp)

    # Calcul des erreurs
    err[k, 0] = numpy.sqrt(dy*dx) * numpy.linalg.norm(U-u, ord=2)
    err[k, 1] = numpy.sqrt(dy*dx) * numpy.linalg.norm(V-v, ord=2)
    err[k, 2] = numpy.sqrt(dy*dx) * numpy.linalg.norm(P-p, ord=2)
    err[k, 3] = numpy.sqrt(dy*dx) * numpy.linalg.norm(D-d, ord=2)

    # Champs d'erreur
    if nx == nx_[-1]:
        ## Champs d'erreur par variable
        pyplot.imshow(numpy.abs(U-u))
        pyplot.colorbar()
        pyplot.suptitle(r"$U({}, \cdot, \cdot)$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".format(t, ny,
        pyplot.show()
```

```
        pyplot.imshow(numpy.abs(V-v))
        pyplot.colorbar()
        pyplot.suptitle(r"$V({}, \cdot, \cdot)$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".format(t, ny,
        pyplot.show()

        pyplot.imshow(numpy.abs(P-p))
        pyplot.colorbar()
        pyplot.suptitle(r"$P({}, \cdot, \cdot)$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".format(t, ny,
        pyplot.show()

        pyplot.imshow(numpy.abs(D))
        pyplot.colorbar()
        pyplot.suptitle(r"$|div(u, v)({}, \cdot, \cdot)|$ ; {}x{} vol ; $n_t$={}, $\Delta t$={}".form
        pyplot.show()

# ANALYSE
pyplot.figure()
pyplot.loglog(nx_, err[:, 0], 'x-', label=r"$ln(\epsilon_u)$")
pyplot.loglog(nx_, err[:, 1], 'x-', label=r"$ln(\epsilon_v)$")
pyplot.loglog(nx_, err[:, 2], 'x-', label=r"$ln(\epsilon_p)$")
pyplot.loglog(nx_, err[:, 3], 'x-', label=r"$ln(\epsilon_d)$")
pyplot.loglog(nx_, dx_, '--', label=r"$\Delta x$")
pyplot.loglog(nx_, dx_**2, '--', label=r"$\Delta x^2$")
pyplot.legend()
pyplot.suptitle(r"$\epsilon_X := || X({}, \cdot, \cdot) - x({}, \cdot, \cdot) ||$ ; dt={}, $n_t$={}"
pyplot.xlabel(r"$ln(N_x)$")
pyplot.show()
```

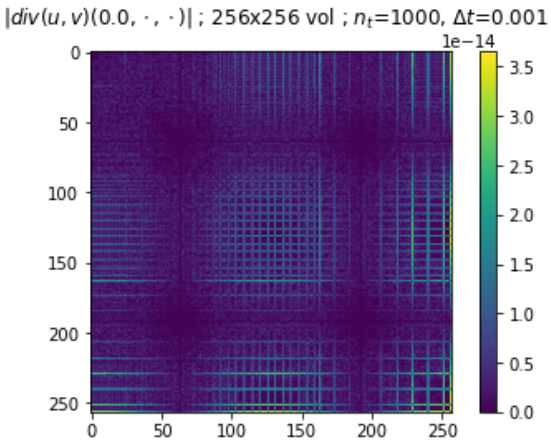

FIGURE 2.15 – Validation de la méthode de reconstruction de la divergence sur les points de pression à partir des vitesses initiales

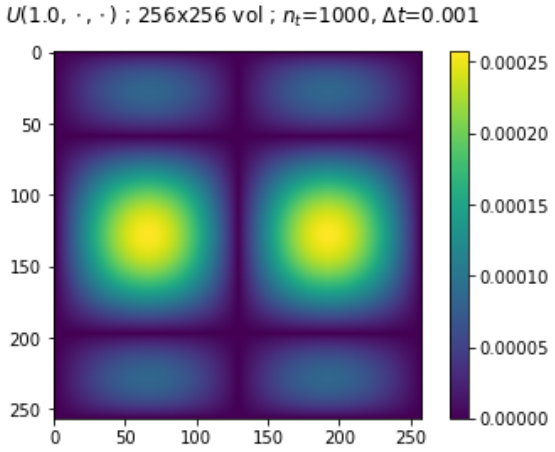

FIGURE 2.16 – Champ d'erreur de U_h au temps final

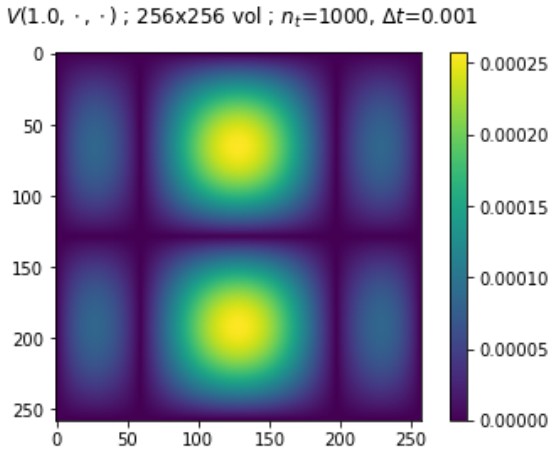

FIGURE 2.17 – Champ d'erreur de V_h au temps final

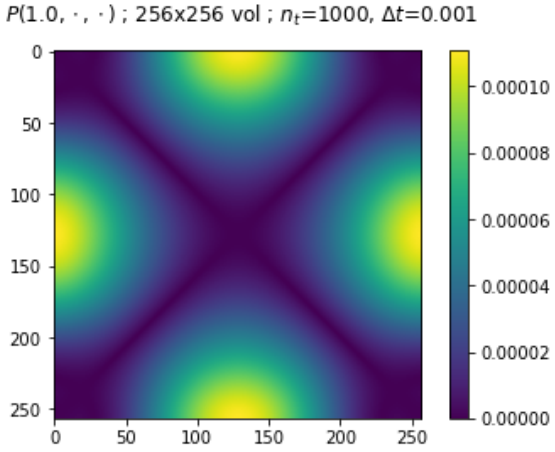

FIGURE 2.18 – Champ d'erreur de la pression au temps final

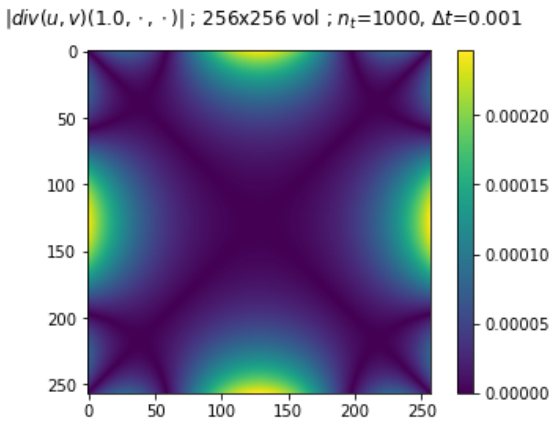

FIGURE 2.19 – Champ d'erreur de la divergence au temps final

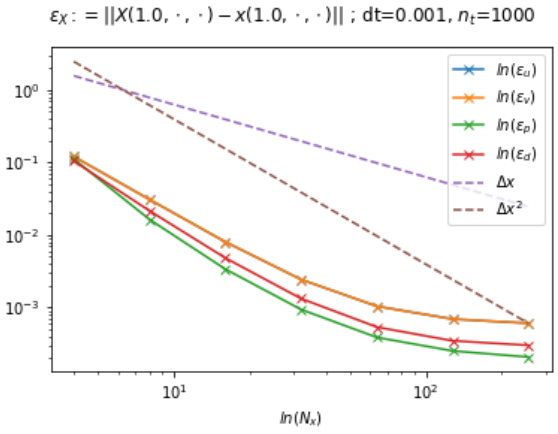

FIGURE 2.20 – Analyse de la méthode S.I.P pour Stokes. Les deux courbes de vitesses sont confondues

2.6 Conclusion

Ce chapitre a été l'occasion de présenter les concepts de la mécanique des fluides, et ses méthodes d'analyse pour le problème de Stokes. On a un très bref aperçu de la variétés de méthodes existantes pour traiter numériquement des questions d'écoulement visqueux. Après avoir étudié le caractère bien posé des équations, on a introduit un maillage décalé et discrétisé ce problème en volumes finis avant de se pencher sur les algorithmes permettant d'approcher une solution classique en deux dimensions du problème continu. Si ce chapitre occupe beaucoup de place dans le mémoire, il a exigé plus de temps encore durant mon stage.

Chapitre 3

Deux modèles d'écoulement en milieu poreux : Darcy et Brinkman

Les écoulements en milieu poreux sont modélisés par différentes lois. On choisit ici d'explorer les cas Darcy, première loi empirique obtenue et son extension à un terme visqueux, connu sous le nom de système de Brinkman. On étudie numériquement la dégénerescence du système de Brinkman en système de Darcy ; plus précisément, on pose la question de la robustesse du code face aux valeurs de la porosité du milieu.

3.1 Description d'un milieu poreux

Dans le cadre de ce mémoire, on caractérise un milieu poreux par sa porosité volumique, la dimension caractéristique de ses pores et par sa perméabilité associée. Il existe plusieurs types de milieux, selon la forme et l'organisation des pores en réseau et nous passerons ces points sous silence.

Volume d'un élément représentatif élémentaire Le milieu poreux est un substrat solide présentant des trous, trous qui seront occupés par une ou plusieurs phases fluides. On définit en premier lieu la taille d'un volume élémentaire représentatif, de taille l, contenant plusieurs pores et des volumes fluides. On note alors $V := l^2$ pour des cellules carrées.

La porosité volumique On note ϕ la porosité volumique d'un milieu poreux. Il est défini par le rapport entre les volumes de fluides sur volume d'un élément représentatif élémentaire.

$$\phi := \frac{V^f}{V}$$

La taille des pores Les milieux diffèrent par la taille caractéristique des pores, par exemple, au sein d'un volume élémentaire représentatif, on peut associer la moyenne des longueurs de chaque élément solide ; on la note en général d_p ou d_f selon la forme des éléments solides.

La perméabilité du milieu Enfin, on définit la perméabilité d'un milieu comme étant un tenseur dont les propriétés dépendent du milieu : il peut être réduit à un scalaire lorsque le milieu est isotrope, diagonal lorsque le milieu est orthotrope, ou plus généralement

$$\mathbf{K} := \begin{pmatrix} K_{xx} & K_{xy} \\ Kyx & K_{yy} \end{pmatrix}$$

pour un milieu en deux dimensions en général. Le tenseur de perméabilité est fonction de la porosité ϕ. La littérature fait essentiellement référence à deux lois de corrélation pour la perméabilité, on a pour les milieux granulaires dont le diamètre moyen d'une fibre est noté d_p

$$K(\phi) \sim \frac{d_p^2 \phi^3}{180(1-\phi)^2} \tag{KC}$$

Cette loi est connue sous le nom de loi de Kozeny-Carman, introduite par Kozeny en 1927 et affinée par Carman en 1939. Une autre correspondant aux milieux fibreux, dont les fibres sont de taille d_f, organisés en réseau carré de cylindres parallèles et soumis à un écoulement parallèle, connue sous le nom de Happel-Langmuir (introduction en 1942 par Langmuir et étudiée par Happel en 1959)

$$K(\phi) \sim \frac{d_f^2}{16(1-\phi)} \left(-\ln(1-\phi) - \frac{3}{2} + 2(1-\phi) - \frac{(1-\phi)^2}{2} \right) \tag{HL}$$

Pour un réseau fibreux en deux dimensions, on dispose par exemple de la loi de Happel dont l'expression est

$$K(\phi) \sim \frac{d_f^2}{32(1-\phi)} \left(-\ln(1-\phi) - \frac{1-(1-\phi)^2}{1+(1-\phi)^2} \right) \tag{HL}$$

étudiée en 1959 toujours pas Happel.

Autres grandeurs Si je n'ai présenté que les caractéristiques et lois utilisées ici, on trouvera d'autre nombres et lois servant à décrire la physique des écoulements en milieux poreux. La tortuosité par exemple, indice de la complexité du réseau de pores, les chaleur volumique et conductivité thermique pour étudier la thermodynamique, etc. On renvoie pour cela à [Cal13].

3.2 Dynamique des écoulements en milieu poreux

Système de Darcy-Brinkman On note toujours Ω un ouvert connexe borné lipschitzien de $\mathbb{R}^d$. L'équation décrivant un écoulement stationnaire d'un fluide newtonien incompressible dans un milieu poreux s'écrit sous la forme

$$\begin{aligned} \nabla \cdot \mathbf{v} = &\quad 0 \\ -\tilde{\mu}\Delta \mathbf{v} + \mu \mathbf{K}^{-1} \cdot \mathbf{v} + \nabla p = &\quad \rho \mathbf{f} \end{aligned} \tag{B}$$

Ce genre de problèmes faisant apparaître un terme d'ordre 1 en vitesse est appelé problème de Stokes généralisé. On remarque que $\mu \mathbf{K}^{-1} > 0$ et on note ce coefficient α. L'analyse du problème continu va à nouveau suivre le cours mentionné dans le chapitre précédent (cours de messieurs Frey et Privat).

Formulation variationnelle On note $V := H_0^1(\Omega)^d$ et $Q := L_0^2(\Omega)$, l'espace des fonctions de carré intégrable de moyenne nulle sur le domaine Ω. La formulation variationnelle du système de Brinkman s'écrit

Trouver $\mathbf{v} \in V$, $p \in Q$,

$$\begin{aligned} \int_\Omega (\alpha \mathbf{v}\mathbf{w} + \nabla \mathbf{v} : \nabla \mathbf{w}) dx - \int_\Omega p \nabla \cdot \mathbf{v} dx &= \int_\Omega \mathbf{f}\mathbf{w} dx \\ \int_\Omega q \nabla \cdot \mathbf{v} dx &= 0 \end{aligned}$$

$\forall \mathbf{w} \in V, q \in Q$. On définit les formes

$$a : V \times V \to \mathbb{R} \qquad \mathbf{v}, \mathbf{w} \mapsto \int_\Omega (\alpha \mathbf{v}\mathbf{w} + \nabla \mathbf{v} : \nabla \mathbf{w}) dx$$

$$b : V \times Q \to \mathbb{R}, \qquad \mathbf{w}, q \mapsto -\int_\Omega q \nabla \cdot \mathbf{w}$$

et enfin

$$f : V \to \mathbb{R}, \qquad \mathbf{w} \mapsto \int_\Omega \mathbf{f} \cdot \mathbf{w} dx$$

Le problème devient

Trouver $(\mathbf{v}, p) \in V \times Q$:

$$\begin{aligned} a(\mathbf{v}, \mathbf{w}) + b(\mathbf{w}, p) &= \mathbf{f}(\mathbf{w}) \quad \forall \mathbf{w} \in V \\ b(\mathbf{v}, q) &= 0 \quad \forall q \in Q \end{aligned}$$

Théorème 3.2.1. *Un couple* $(\mathbf{v}, p)$ *est solution du problème de Brinkman si, et seulement si, il est un point selle pour le lagrangien*

$$\mathcal{L}(\mathbf{w}, q) := \frac{1}{2}a(\mathbf{w}, \mathbf{w}) + b(\mathbf{w}, q) - f(\mathbf{w})$$

donc si, et seulement si,

$$\mathcal{L}(\mathbf{v}, p) = \min_{\mathbf{w}\in V} \max_{q\in Q} \mathcal{L}(\mathbf{w}, q)$$

3.3 Expériences numériques pour le système de Brinkman dépendant du temps en deux dimensions

Obtention du système linéaire En notant $\mathbf{v} := \begin{pmatrix} u \\ v \end{pmatrix}$, le système de Brinkman dépendant du temps. En supposant que $\rho = 1$ et $\mu = 1$. Soit $\mathbf{K} = K\mathrm{Id}_{\mathbb{R}^2}$. Le système de Brinkman en 2.D s'écrit

$$\begin{aligned}
\partial_x u + \partial_y v &= 0 \\
\frac{\rho}{\phi}\partial_t u - \frac{\mu}{\phi}\Delta u + \frac{\mu}{K}u + \partial_x p &= f^u \\
\frac{\rho}{\phi}\partial_t v - \frac{\mu}{\phi}\Delta v + \frac{1}{K}v + \partial_y p &= f^v \\
u_{|\partial\Omega} &= u^D \\
v_{|\partial\Omega} &= v^D
\end{aligned}$$

On utilise, comme pour le problème de Stokes, une méthode de projection scalaire sur maillage M.A.C. Le problème se construit à l'aide d'un tripler de Green-Taylor dépendant du temps, on considère les fonctions

$$\begin{aligned}
u(t,x,y) &= sin(x)cos(y)e^{-2t} \\
v(t,x,y) &= -\,cos(x)sin(y)e^{-2t} \\
p(t,x,y) &= 0
\end{aligned}$$

Dans ce cas le vecteur second membre s'écrit

$$\begin{aligned}
f^u(t,x,y) &= \frac{1}{K}u(t,x,y) \\
f^v(t,x,y) &= \frac{1}{K}v(t,x,y)
\end{aligned}$$

Le système linéaire de l'étape prédiction est donné par

$$\begin{aligned}
\frac{\Delta y}{\Delta x}\left(\tilde{U}_{j,i+1} + \tilde{U}_{j,i-1}\right) + \frac{\Delta x}{\Delta y}\left(\tilde{U}_{j-1,i} + \tilde{U}_{j-1,i}\right) + \left[2\left(\frac{\Delta y}{\Delta x} + \frac{\Delta x}{\Delta y}\right) + (\Delta y \Delta x)\left(\frac{1}{\Delta t} + \frac{1}{K}\right)\right]\tilde{U}_{j,i} = \\
\Delta y \Delta x f^u_{j,i} + \frac{\Delta y \Delta x}{\Delta t}U_{j,i} - (\Delta y \Delta x)(P_{j,i} - P_{j,i-1}) \\
\frac{\Delta y}{\Delta x}\left(\tilde{V}_{j,i+1} + \tilde{V}_{j,i-1}\right) + \frac{\Delta x}{\Delta y}\left(\tilde{V}_{j-1,i} + \tilde{V}_{j-1,i}\right) + \left[2\left(\frac{\Delta y}{\Delta x} + \frac{\Delta x}{\Delta y}\right) + (\Delta y \Delta x)\left(\frac{1}{\Delta t} + \frac{1}{K}\right)\right]\tilde{V}_{j,i} = \\
\Delta y \Delta x f^v_{j,i} + \frac{\Delta y \Delta x}{\Delta t}V_{j,i} - (\Delta y \Delta x)(P_{j,i} - P_{j-1,i})
\end{aligned}$$

Ensuite on reconstruit la divergence sur le sous-maillage $\mathcal{T}^P$, avant d'implémenter le laplacien de Neumann moins un coin, qui aura la condition $\Phi_{0,0} = 0$.

Enfin, on peut corriger l'état via l'étape de correction :

$$\begin{aligned}
P^{(n+1)} &= \Phi + P^{(n)} \\
U^{(n+1)} &= \tilde{U}^{(n+1)} - \Delta t \nabla_x \Phi \\
V^{(n+1)} &= \tilde{V}^{(n+1)} - \Delta t \nabla_y \Phi
\end{aligned}$$

On assemble le système pour les solutions de Taylor-Green.

Tenseur de perméabilité sphérique On suppose dans cette simulation que $\mathbf{K} := K\mathrm{Id}_{\mathbb{R}^2}$. On étudie la convergence $L^2(\Omega)$ pour plusieurs discrétisations temporelles.

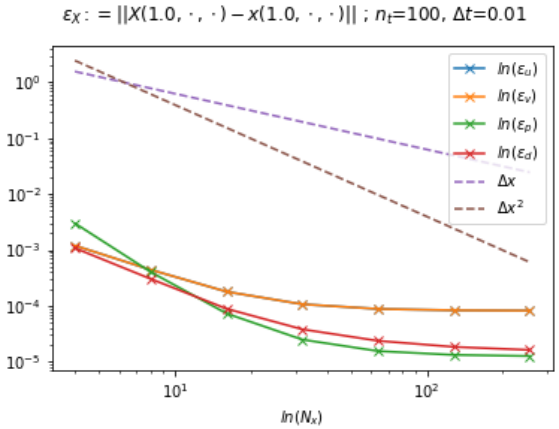

FIGURE 3.1 – Analyse de la méthode S.I.P jusqu'à 256×256 volumes.

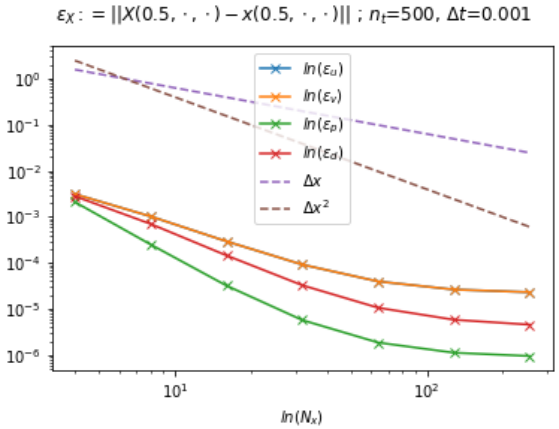

FIGURE 3.2 – Analyse de la méthode S.I.P jusqu'à 256×256 volumes.

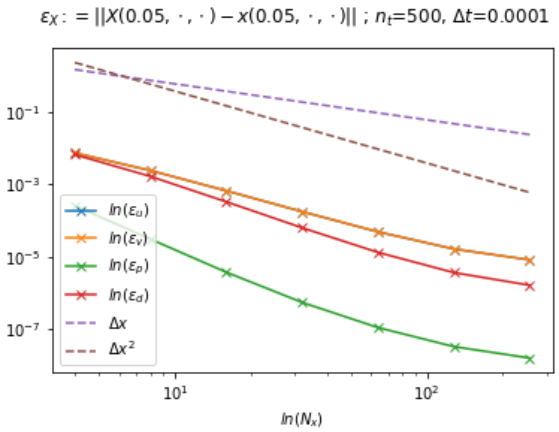

FIGURE 3.3 – Analyse de la méthode S.I.P jusqu'à 256 × 256 volumes.

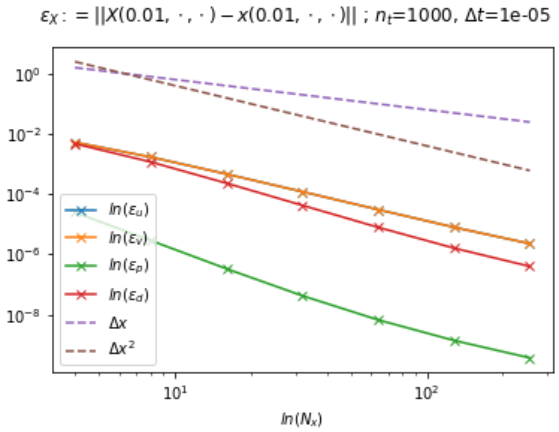

FIGURE 3.4 – Analyse de la méthode S.I.P jusqu'à 256 × 256 volumes.

Tenseur de perméabilité orthotrope On suppose cette fois que le tenseur $\mathbf{K}$ est diagonal de valeurs propres différentes. On prend un rapport de 1 à 5 et on constate que les erreurs des composantes de vitesse décoïncident, en norme $L^2(\Omega)$.

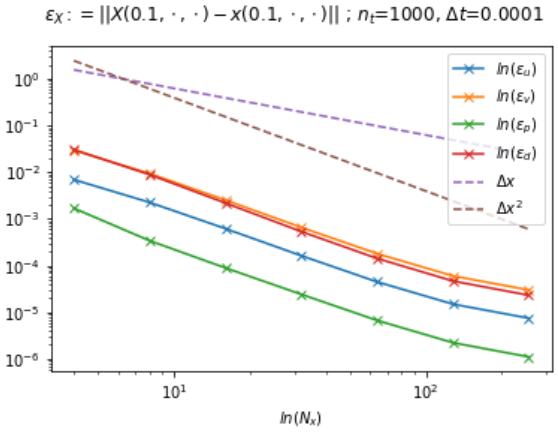

FIGURE 3.5 – Simulation de Brinkman - Taylor - Green avec $\mathbf{K} := \begin{pmatrix} 0.01 & 0. \\ 0. & 0.05 \end{pmatrix}$; analyse jusqu'à 256×256 volumes

3.4 Étude numérique de la dégénerescence en système de Darcy

Le système de Brinkman adimensioné fait apparaître le nombre de Darcy

$$Da := \frac{K}{L^2}$$

où K est la hauteur de la cavité. On définit en général le nombre de Brinkman comme témoin du rapport entre le terme visqueux et le terme de Dacy.

$$Br := \frac{Da(\phi)}{\phi}$$

Estimation du nombre de brinkman en fonction de la porosité

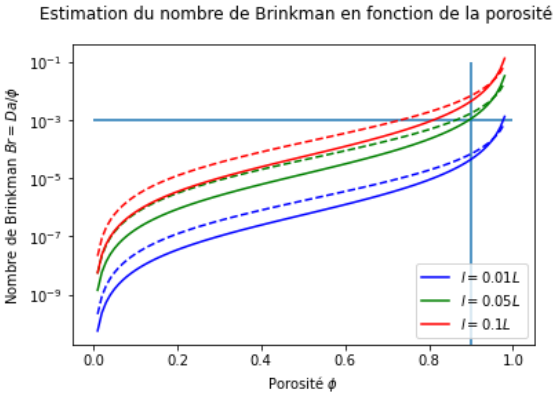

FIGURE 3.6 – En pointillé : loi de Happel, en traits pleins : loi de Kozeny-Carman.

On voit que le rapport dépasse 10^{-3} pour les porosités supérieures à 0.9 (courbe verte, cas Kozeny-Carman). Dans ce même cas, il est supérieur à 10^{-2} pour les porosités supérieurs à 0.95. Cela donne une idée de la plage de valeurs sur laquelle le terme de Brinkman (visqueux) est négligeable devant le terme de Darcy, le terme d'inertie.

Robustesse du code Une question intéressante est alors d'obtenir une idée de l'évolution de l'erreur, à discrétisation constante, en fonction de la porosité ϕ. On déroule l'algorithme à discrétisation temporelle fixée, pour différents maillages entre 32×32 et 512×512, on obtient les courbes

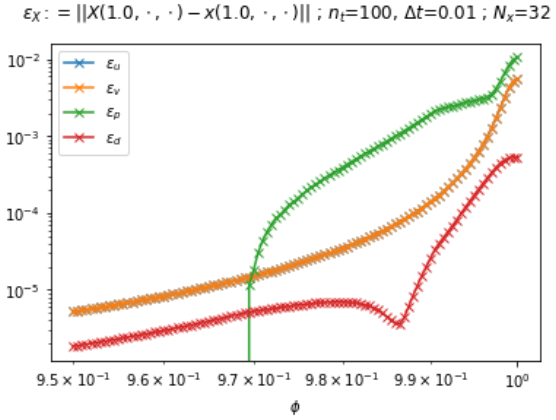

FIGURE 3.7 – Zoom sur la plage $\phi > 0.95$

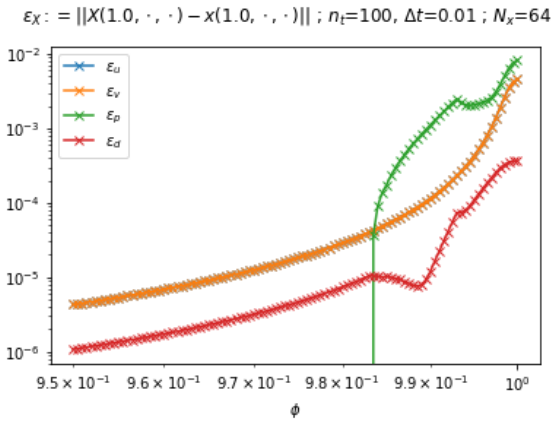

FIGURE 3.8 – Zoom sur la plage $\phi > 0.95$

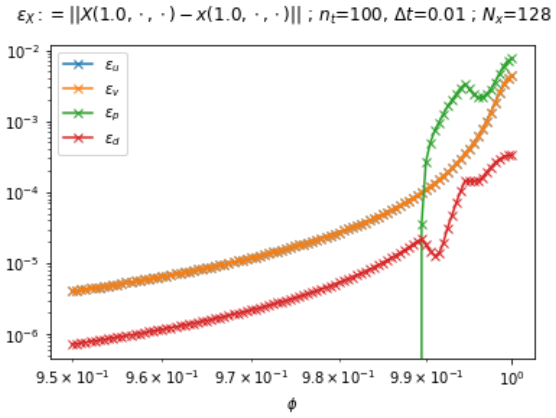

FIGURE 3.9 – Zoom sur la plage $\phi > 0.95$

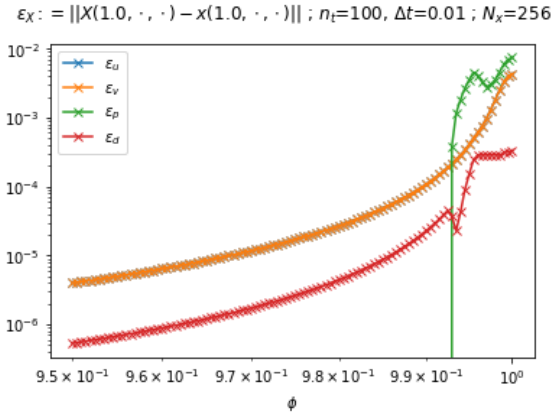

FIGURE 3.10 – Zoom sur la plage $\phi > 0.95$

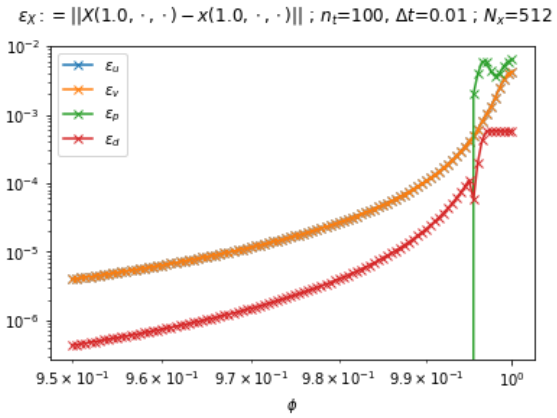

FIGURE 3.11 – Zoom sur la plage $\phi > 0.95$

Quel que soit le maillage, il n'y a pas d'événement marquant avant les valeurs de porosité supérieures à 0.95. En témoin de la tendance, les deux courbes qui montrent la croissance sur les plages $\phi \in [.1, .9]$ puis $\phi \in [.9, .94]$:

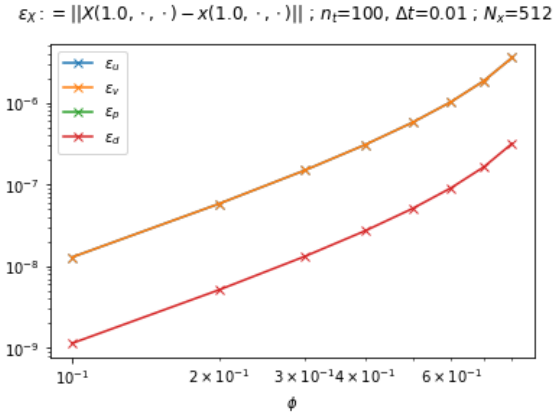

FIGURE 3.12 – Zoom sur la plage $\phi \in [.1, .9]$

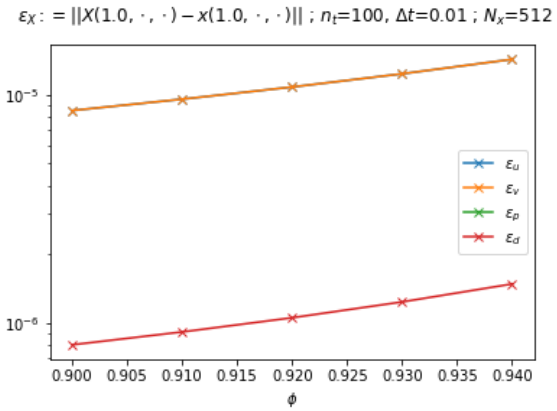

FIGURE 3.13 – Zoom sur la plage $\phi \in [.9, .95]$

3.5 Conclusion

Dans ce chapitre, nous avons appris quelques éléments de modélisation des écoulements en milieux poreux, on a validé un code pour la simulation de ces écoulements et nous avons testé numériquement la limite de l'algorithme pour les grandes valeurs de ϕ. On a également pu étudier la prépondérance du terme d'inertie, de Darcy, sur le terme visqueux, de Brinkman. Á ce stade du mémoire, nous disposons donc d'un outil pour attaquer le problème de l'interface entre deux régions : une fluide, et une poreuse, séparées par une interface.

Chapitre 4

Superposition d'une couche fluide sur une couche poreuse

Dans ce chapitre, on essaie d'étudier les nouvelles conditions de saut pour les problèmes d'écoulement fluide avec interface avec un milieu poreux. La modélisation, en particulier la dérivation des conditions de saut et leur comparaison avec d'autres jeux de conditions ont été traités dans [AGO17]. L'analyse du système couplé se trouve par exemple dans [Ang18].

4.1 Mise en situation

Soit Ω un ouvert connexe borné de $\mathbb{R}^d$ avec $d \leqslant 3$. On suppose que sa frontière $\Gamma := \partial\Omega$ est lipschitzienne et on note ν le vecteur unitaire sortant de ce domaine. Le domaine Ω est divisé en deux composantes connexes disjointes Ω_f, région fluide, Ω_p, région poreuse, séparées par une frontière lipschitzienne commune $\Sigma \subset \mathbb{R}^{d-1}$. L'hyperplan Σ est muni d'une base orthonormée $\{\tau_j\}$, $j \leqslant d-1$ et d'un vecteur normal $\mathbf{n}$ dont on fixe le sens comme étant de la région poreuse vers la région fluide. La situation se résume par le dessin, issu de [Ang21], figure 1 :

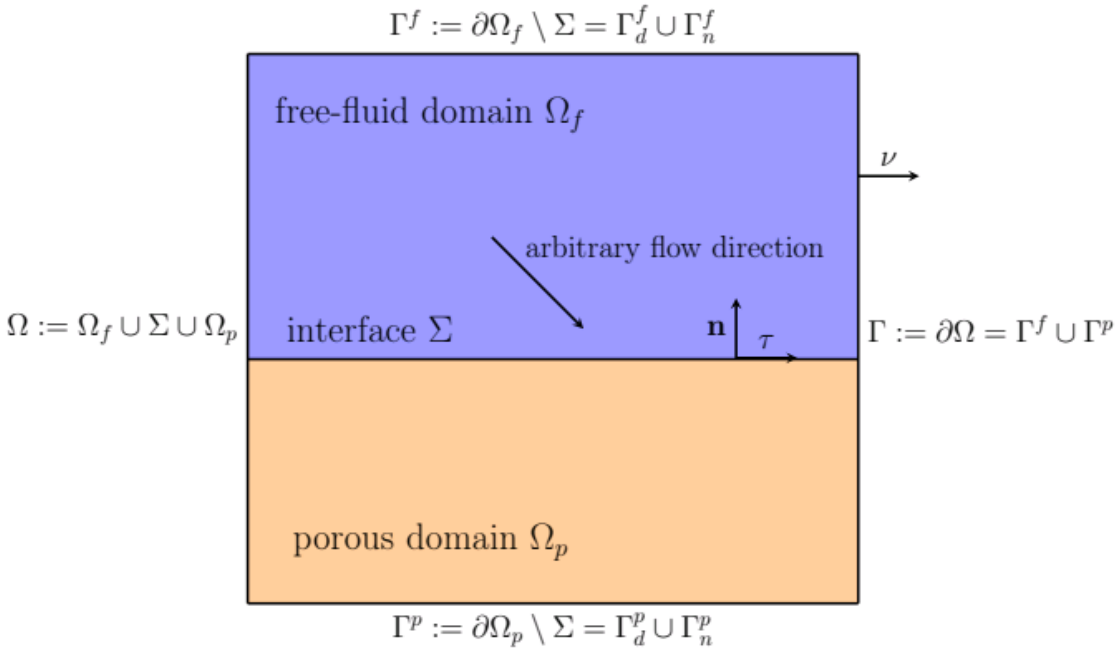

FIGURE 4.1 – Situation ; Γ_d^p et Γ_d^f sont munis de conditions de Dirichlet en vitesse et Γ_n^p ou Γ_n^f sont munies de conditions de Neumann, en vitesse également.

Les auteurs introduisent alors le cadre fonctionnel. On notera $H^s(\Omega) := W^{s,2}(\Omega)$, muni des normes

usuelles

$$\|u\|_{s,\Omega} := \int_\Omega (1+|\xi|^2)^s |\hat{u}(\xi)|^2 d\xi$$

On note $< \cdot, \cdot >_{-1,\Omega}$ le crochet de dualité entre $H^{-1}(\Omega)$ et $H_0^1(\Omega)$. On introduit enfin les espaces de Hilbert réels

$$\mathbf{H}_{div}(\Omega) := \{u \in L^2(\Omega); \nabla \cdot u \in L^2(\Omega)\}$$

muni de la norme de graphe définie par

$$\|u\|^2_{\mathbf{H}_{div}} := \|u\|^2_{0,\Omega} + \|\nabla \cdot u\|^2_{0,\Omega}$$

et

$$L^2_0(\Omega) := \{q \in L^2(\Omega); \int_\Omega q(x)dx = 0\}$$

muni de la norme usuelle $\|\cdot\|_{0,\Omega}$.

Le cadre fonctionnel propre à Σ est ensuite introduit. Il est question des espaces $\tilde{H}^{1/2}(\Sigma)$ l'espace des traces des fonctions $H^1_{0,\partial\Omega_f\setminus\Sigma}$ ou $H^1_{0,\partial\Omega_p\setminus\Sigma}$, nulles sur les bords $\partial\Omega_f\setminus\Sigma$ ou $\partial\Omega_p\setminus\Sigma$. Ils notent de plus $< \cdot, \cdot >_{-1/2,\Sigma}$ le crochet de dualité entre $\tilde{H}^{-1/2}(\Sigma)$ et $\tilde{H}^{1/2}(\Sigma)$. L'espace $\tilde{H}^{-1/2}(\Sigma)$ étant un espace de distributions sur Σ. Les deux espaces $H^{1/2}(\Sigma)$ et $\tilde{H}^{1/2}(\Sigma)$ sont équipés des normes

$$|u|^2_{H^{1/2}(\Sigma)} := \int_\Sigma \int_\Sigma \frac{|u(\mathbf{x})-u(\mathbf{y}|^2}{|\mathbf{y}-\mathbf{x}|^d} d\mathbf{x}d\mathbf{y} \quad ; \quad \|u\|^2_{H^{1/2}(\Sigma)} := \|u\|^2_{0,\Sigma} + |\nabla u|^2_{H^{1/2}(\Sigma)}$$

$$|u|^2_{\tilde{H}^{1/2}(\Sigma)} := |u|^2_{H^{1/2}(\Sigma)} + \int_\Sigma \frac{|u(\mathbf{x})|^2}{d(\mathbf{x},\partial\Sigma)} d\mathbf{x} \quad ; \quad \|u\|^2_{\tilde{H}^{1/2}(\Sigma)} := \|u\|^2_{0,\Sigma} + |u|^2_{\tilde{H}^{1/2}(\Sigma)}$$

où $d(\mathbf{x},\partial\Sigma) := \inf\{d(\mathbf{x},\mathbf{y}); \mathbf{y} \in \partial\Sigma\}$. L'écriture des normes est licite du fait qu'on a les injections

$$H^{1/2}(\Sigma) \hookrightarrow L^2(\Sigma)$$

et

$$\tilde{H}^{1/2}(\Sigma) \hookrightarrow L^2(\Sigma)$$

En guise de propriétés, sont cités le fait que l'injection de $\tilde{H}^{1/2}(\Sigma) \hookrightarrow H^{1/2}(\Sigma)$ est continue du fait que la distance de $d(\mathbf{x},\partial\Sigma)$ est $W^{1,\infty}(\Sigma)$ mais l'équivalence entre les normes n'est vaie que si Σ est une courbe fermée lorsque $d=2$ ou plus généralement une surface fermée. Enfin, les auteurs proposent la propriété suivante pour les prolongements par 0 de fonctions $\tilde{u} \in \tilde{H}^{1/2}(\Sigma)$ à tout $\partial\Omega_f$ (respectivement $\partial\Omega_p$), noté $u \in H^{1/2}(\partial\Omega_f)$:

$$\exists c(\Sigma,\partial\Omega_f) > 0 \quad \forall \tilde{u} \in \tilde{H}^{1/2}(\Sigma) \qquad \|u\|^{1/2}_H(\Sigma) \leqslant c(\Sigma,\partial\Omega_f)\|u\|_{\tilde{H}^{1/2}(\Sigma)}$$

Modèle pour le milieu fluide Ω_f Dans la région purement fluide, on considère le modèle de Stokes à masse volumique et viscosité constantes, qui correspond à un écoulement incompressible d'un fluide visqueux à faible nombre de Reynolds

$$\begin{aligned} \nabla \cdot \mathbf{v} &= 0 \quad (0,T)\times\Omega_f \\ \rho\partial_t \mathbf{v} - \mu\nabla\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T\right) + \nabla p &= \rho\mathbf{f^f} \quad (0,T)\times\Omega_f \end{aligned}$$

Modèle pour le milieu poreux On regarde dans le milieu poreux le problème de Brinkman pour un fluide visqueux incompressible s'écoulant à faible nombre de Reynolds.

$$\begin{aligned} \nabla \cdot \mathbf{v} &= 0 \quad (0,T)\times\Omega_p \\ \tfrac{\rho}{\phi}\partial_t \mathbf{v} - \tfrac{\mu}{\phi}\nabla\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T\right) + \mu\mathbf{K}^{-1} + \nabla p &= \rho\mathbf{f^p} \quad (0,T)\times\Omega_p \end{aligned}$$

Tenseur de stress du problème Pour la région fluide, on introduit le tenseur de Cauchy du milieu

$$\sigma^f(\mathbf{v},p) := \mu\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T\right) - p\mathrm{Id}_{\mathbb{R}^d}$$

et dans la région poreuse le tenseur

$$\sigma^p(\mathbf{v},p) := \frac{\mu}{\phi}\left(\nabla\mathbf{v} + \mathbf{v}^T)\right) - p\mathrm{Id}_{\mathbb{R}^d}$$

Existence et unicité avec viscosité variable Pour l'analyse du système dans son cas le plus général, on se référera à [Ang18].

4.2 Expérience numérique en 2.D

La seule expérience à laquelle je me serai prêté est la suivante : essayer de simuler le tourbillon de Taylor-Green, avec un saut nul sur la vitesse comme sur la pression. Des solutions de Taylor-Green, on explicite le problème avec les deux sauts nuls. L'idée étant de tester plusieurs altitudes pour la surface Σ, comme présentée dans [Ang21]. Au cours du stage, je n'aurai eu le temps que d'effectuer une simulation, qui est présentée ici.

On implémente le système couplé à partir des solutions de Taylor-Green. Le saut en pression comme en vitesse est nul. Le champ de divergence est nul en tout temps et en tout point ; on suppose que l'interface porte une densité de force nulle. Le système, pour Taylor-Green, s'écrit, avec tenseur de perméabilité réduit à un scalaire :

$$\begin{aligned}
\partial_x u + \partial_y &= 0 && \Omega \\
-\partial_t u - \Delta u + \partial_x p &= 0 && \Omega^f \\
-\frac{1}{\phi}\partial_t u - \frac{1}{\phi}\Delta u + \frac{1}{K}u &= \frac{1}{K}u && \Omega^p \\
-\partial_t v - \Delta v + \partial_y p &= 0 && \Omega^f \\
-\frac{1}{\phi}\partial_t v - \frac{1}{\phi}\Delta v + \frac{1}{K}v + \partial_y p &= \frac{1}{K}v && \Omega^p
\end{aligned}$$

Fixant Σ au milieu du domaine, sur une ligne de pression / vitesse horizontale, l'idée était d'assembler le système par région. On note n_y (un nombre pair) le nombre de volumes sur la dimension verticale, on compte donc $n_y/2$ volumes dans la région fluide et $n_y/2$ dans la région poreuse.

```
# Assemblage du prédicteur sur la vitesse U
N = 5*(ny-1)*nx + 2*(ny-1) + 2*(nx+2)
lignes   = numpy.zeros(N)
colonnes = numpy.zeros(N)
donnees  = numpy.zeros(N)

compteur = 0
    for j in range(ny+1):
        for i in range(nx+2):

            # Si U[j,i] est au bord, appliquer la condition de Dirichlet
            if i == 0 or i == nx+1 or j == 0 or j == ny:
                lignes[compteur]   = j * (nx+2) + i
                colonnes[compteur] = j * (nx+2) + i
                donnees[compteur]  = 1
                compteur += 1

            # U[j,1] dans le milieu poreux (Brinkman)
            elif i == 1 and j < ny//2:
                lignes[compteur : compteur+5] = j * (nx+2) + i
                colonnes[compteur]   = j * (nx+2) + i
                colonnes[compteur+1] = j * (nx+2) + i - 1
                colonnes[compteur+2] = j * (nx+2) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+2) + i
                colonnes[compteur+4] = (j+1) * (nx+2) + i
                donnees[compteur]    = mu/phi*(3*(dy/dx) + 2*(dx/dy)) + (dy*dx)/dt + (dy*dx)*mu/Kp
                donnees[compteur+1]  = mu/phi*(-2*dy/dx)
                donnees[compteur+2]  = mu/phi*(-dy/dx)
                donnees[compteur+3]  = mu/phi*(-dx/dy)
                donnees[compteur+4]  = mu/phi*(-dx/dy)
                compteur += 5
```

```
# U[j,1] dans le milieu fluide (Stokes)
elif i == 1 and j > ny//2-1:
    lignes[compteur : compteur+5] = j * (nx+2) + i
    colonnes[compteur]   = j * (nx+2) + i
    colonnes[compteur+1] = j * (nx+2) + i - 1
    colonnes[compteur+2] = j * (nx+2) + i + 1
    colonnes[compteur+3] = (j-1) * (nx+2) + i
    colonnes[compteur+4] = (j+1) * (nx+2) + i
    donnees[compteur]    = mu*(3*(dy/dx) + 2*(dx/dy)) + + (dy*dx)/dt
    donnees[compteur+1]  = mu*(-2*dy/dx)
    donnees[compteur+2]  = mu*(-dy/dx)
    donnees[compteur+3]  = mu*(-dx/dy)
    donnees[compteur+4]  = mu*(-dx/dy)
    compteur += 5

# U[j,nx] dans le milieu poreux (Brinkman)
elif i == nx and j < ny//2:
    lignes[compteur : compteur+5] = j * (nx+2) + i
    colonnes[compteur]   = j * (nx+2) + i
    colonnes[compteur+1] = j * (nx+2) + i - 1
    colonnes[compteur+2] = j * (nx+2) + i + 1
    colonnes[compteur+3] = (j-1) * (nx+2) + i
    colonnes[compteur+4] = (j+1) * (nx+2) + i
    donnees[compteur]    = mu/phi*(3*(dy/dx) + 2*(dx/dy)) + (dy*dx)/dt + (dy*dx)*mu/Kp
    donnees[compteur+1]  = mu/phi*(-dy/dx)
    donnees[compteur+2]  = mu/phi*(-2*dy/dx)
    donnees[compteur+3]  = mu/phi*(-dx/dy)
    donnees[compteur+4]  = mu/phi*(-dx/dy)
    compteur += 5

# U[j,nx] dans le milieu fluide (Stokes)
elif i == nx and j > ny//2-1:
    lignes[compteur : compteur+5] = j * (nx+2) + i
    colonnes[compteur]   = j * (nx+2) + i
    colonnes[compteur+1] = j * (nx+2) + i - 1
    colonnes[compteur+2] = j * (nx+2) + i + 1
    colonnes[compteur+3] = (j-1) * (nx+2) + i
    colonnes[compteur+4] = (j+1) * (nx+2) + i
    donnees[compteur]    = mu*(3*(dy/dx) + 2*(dx/dy)) + (dy*dx)/dt
    donnees[compteur+1]  = mu*(-dy/dx)
    donnees[compteur+2]  = mu*(-2*dy/dx)
    donnees[compteur+3]  = mu*(-dx/dy)
    donnees[compteur+4]  = mu*(-dx/dy)
    compteur += 5

# U[j,i] à l'intérieur de la région poreuse (Brinkman)
elif j < ny//2:
    lignes[compteur : compteur+5] = j * (nx+2) + i
    colonnes[compteur]   = j * (nx+2) + i
    colonnes[compteur+1] = j * (nx+2) + i - 1
    colonnes[compteur+2] = j * (nx+2) + i + 1
    colonnes[compteur+3] = (j-1) * (nx+2) + i
    colonnes[compteur+4] = (j+1) * (nx+2) + i
    donnees[compteur]    = mu/phi*(2 * (dy/dx + dx/dy)) + (dy*dx)/dt + (dy*dx)*mu/Kp
    donnees[compteur+1]  = mu/phi*(-dy/dx)
    donnees[compteur+2]  = mu/phi*(-dy/dx)
    donnees[compteur+3]  = mu/phi*(-dx/dy)
```

```
                donnees[compteur+4]  = mu/phi*(-dx/dy)
                compteur += 5

            # U[j,i] à l'intérieur de la région fluide (Stokes)
            else:
                lignes[compteur : compteur+5] = j * (nx+2) + i
                colonnes[compteur]   = j * (nx+2) + i
                colonnes[compteur+1] = j * (nx+2) + i - 1
                colonnes[compteur+2] = j * (nx+2) + i + 1
                colonnes[compteur+3] = (j-1) * (nx+2) + i
                colonnes[compteur+4] = (j+1) * (nx+2) + i
                donnees[compteur]    = mu*(2 * (dy/dx + dx/dy)) + (dy*dx)/dt
                donnees[compteur+1]  = mu*(-dy/dx)
                donnees[compteur+2]  = mu*(-dy/dx)
                donnees[compteur+3]  = mu*(-dx/dy)
                donnees[compteur+4]  = mu*(-dx/dy)
                compteur += 5

Auu = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+1)*(nx+2), (ny+1)*(nx+2))).tocsc()
Mu = sparse.linalg.LinearOperator(((ny+1)*(nx+2), (ny+1)*(nx+2)), splu(Auu).solve)

# Assemblage du prédicteur sur la vitesse V
N = 5*ny*(nx-1) + 2*(ny+2) + 2*(nx-1)
lignes   = numpy.zeros(N)
colonnes = numpy.zeros(N)
donnees  = numpy.zeros(N)

compteur = 0
for j in range(ny+2):
    for i in range(nx+1):

        # Si V[j,i] est sur le bord du domaine
        if i == 0 or i == nx or j == 0 or j == ny+1:
            lignes[compteur]   = j * (nx+1) + i
            colonnes[compteur] = j * (nx+1) + i
            donnees[compteur]  = 1
            compteur += 1

        # Si V[j,i] est sur la deuxième ligne (Brinkman)
        elif j == 1:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = mu/phi*(2*(dy/dx) + 3*(dx/dy)) + (dy*dx)/dt + (dy*dx)*mu/Kp
            donnees[compteur+1]  = mu/phi*(-dy/dx)
            donnees[compteur+2]  = mu/phi*(-dy/dx)
            donnees[compteur+3]  = mu/phi*(-2 * dx/dy)
            donnees[compteur+4]  = mu/phi*(-dx/dy)
            compteur += 5

        # Si V[j,i] est sur l'avant-dernière ligne (Stokes)
        elif j == ny:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
```

```
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = mu*(2*(dy/dx) + 3*(dx/dy)) + (dy*dx)/dt
            donnees[compteur+1]  = mu*(-dy/dx)
            donnees[compteur+2]  = mu*(-dy/dx)
            donnees[compteur+3]  = mu*(-dx/dy)
            donnees[compteur+4]  = mu*(-2 * dx/dy)
            compteur += 5

        # Si V[j,i] est à l'intérieur de la région poreuse (Brinkman)
        elif j < ny//2:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = mu/phi*(2 * (dy/dx + dx/dy)) + (dy*dx)/dt + (dy*dx)*mu/Kp
            donnees[compteur+1]  = mu/phi*(-dy/dx)
            donnees[compteur+2]  = mu/phi*(-dy/dx)
            donnees[compteur+3]  = mu/phi*(-dx/dy)
            donnees[compteur+4]  = mu/phi*(-dx/dy)
            compteur += 5

        # Si V[j,i] est à l'intérieur de la région fluide (Stokes)
        elif j > ny//2-1:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = mu*(2 * (dy/dx + dx/dy)) + (dy*dx)/dt
            donnees[compteur+1]  = mu*(-dy/dx)
            donnees[compteur+2]  = mu*(-dy/dx)
            donnees[compteur+3]  = mu*(-dx/dy)
            donnees[compteur+4]  = mu*(-dx/dy)
            compteur += 5

Avv = sparse.coo_matrix((donnees, (lignes, colonnes)), shape=((ny+2)*(nx+1), (ny+2)*(nx+1))).tocsc()
Mv = sparse.linalg.LinearOperator(((ny+2)*(nx+1), (ny+2)*(nx+1)), splu(Avv).solve)

# Assemblage du Laplacien de Neumann pour P
N = 5*(ny-1)*(nx-1) + 2*4*(ny-1) + 2*4*(nx-1) + 3 * 3 + 1
lignes   = numpy.zeros(N)
colonnes = numpy.zeros(N)
donnees  = numpy.zeros(N)

compteur = 0
for j in range(ny+1):
        for i in range(nx+1):

            if i == 0 and j == 0:
                lignes[compteur]   = j * (nx+1) + i
                colonnes[compteur] = j * (nx+1) + i
                donnees[compteur]  = 1
                compteur += 1

            elif i == 0 and j == ny:
```

```
        lignes[compteur : compteur+3] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i + 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        donnees[compteur]    = - 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = 2 * dy / dx
        donnees[compteur+2]  = 2 * dx / dy
        compteur += 3

    elif i == nx and j == 0:
        lignes[compteur : compteur+3] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
        colonnes[compteur+2] = (j+1) * (nx+1) + i
        donnees[compteur]    = - 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = 2 * dy / dx
        donnees[compteur+2]  = 2 * dx / dy
        compteur += 3

    elif i == nx and j == ny:
        lignes[compteur : compteur+3] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        donnees[compteur]    = - 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = 2 * dy / dx
        donnees[compteur+2]  = 2 * dx / dy
        compteur += 3

    elif i == 0 and j > 0 and j < ny:
        lignes[compteur : compteur+4] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i + 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        colonnes[compteur+3] = (j+1) * (nx+1) + i
        donnees[compteur]    = - 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = 2 * dy/dx
        donnees[compteur+2]  = dx/dy
        donnees[compteur+3]  = dx/dy
        compteur += 4

    elif i == nx and j > 0 and j < ny:
        lignes[compteur : compteur+4] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
        colonnes[compteur+2] = (j-1) * (nx+1) + i
        colonnes[compteur+3] = (j+1) * (nx+1) + i
        donnees[compteur]    = - 2 * (dy/dx + dx/dy)
        donnees[compteur+1]  = 2 * dy/dx
        donnees[compteur+2]  = dx/dy
        donnees[compteur+3]  = dx/dy
        compteur += 4

    elif j == 0 and i > 0 and i < nx:
        lignes[compteur : compteur+4] = j * (nx+1) + i
        colonnes[compteur]   = j * (nx+1) + i
        colonnes[compteur+1] = j * (nx+1) + i - 1
        colonnes[compteur+2] = j * (nx+1) + i + 1
```

```
            colonnes[compteur+3] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = dy/dx
            donnees[compteur+2]  = dy/dx
            donnees[compteur+3]  = 2 * dx/dy
            compteur += 4

        elif j == ny and i > 0 and i < nx:
            lignes[compteur : compteur+4] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = dy/dx
            donnees[compteur+2]  = dy/dx
            donnees[compteur+3]  = 2 * dx/dy
            compteur += 4

        else:
            lignes[compteur : compteur+5] = j * (nx+1) + i
            colonnes[compteur]   = j * (nx+1) + i
            colonnes[compteur+1] = j * (nx+1) + i - 1
            colonnes[compteur+2] = j * (nx+1) + i + 1
            colonnes[compteur+3] = (j-1) * (nx+1) + i
            colonnes[compteur+4] = (j+1) * (nx+1) + i
            donnees[compteur]    = - 2 * (dy/dx + dx/dy)
            donnees[compteur+1]  = dy/dx
            donnees[compteur+2]  = dy/dx
            donnees[compteur+3]  = dx/dy
            donnees[compteur+4]  = dx/dy
            compteur += 5

App = sparse.coo_matrix((dt*donnees, (lignes, colonnes)), shape=((ny+1)*(nx+1), (ny+1)*(nx+1))).tocsc
Mp = sparse.linalg.LinearOperator(((ny+1)*(nx+1), (ny+1)*(nx+1)), splu(App).solve)
```

Les résultats de la simulation effectuée avec ces opérateurs discrets sont ci-dessous. On regarde le maillage 512×512 et plusieurs valeurs de porosité.

On commence par regarder les champs d'erreur sur la variable U pour plusieurs valeurs de la porosité.

On présente enfin les graphes d'analyse.

Bien que les courbes soient cassées, et présentant des valeurs bien trop grandes pour valider le code utilisé, les courbes de vitesse et de pression ont une tendance à l'ordre deux. Le problème réside probablement dans la composition du problème modèle et dans les choix tels que l'emplacement de la surface Σ.

4.3 Conclusion

Il n'y a pas grand chose à conclure de cette partie si ce n'est que si l'intention était là, le résultat est décevant. Bien sûr, avec plus de temps et de résultat, on aurait voulu poursuivre cette étude en proposant d'autres lois d'une part et un raffinement pour les grandes valeurs de porosité ϕ. En ce qui concerne l'emplacement de Σ, il est probable que l'imposer sur une ligne de points de pression est peu astucieux, surtout en vue d'évolutions avec traitement d'un saut non-nul. En effet, la discrétisation de l'opérateur fait intervenir deux valeurs P^p et P^f, laquelle doit être imposée sur la ligne en question ?

Autant de questions qui concluent mon mémoire.

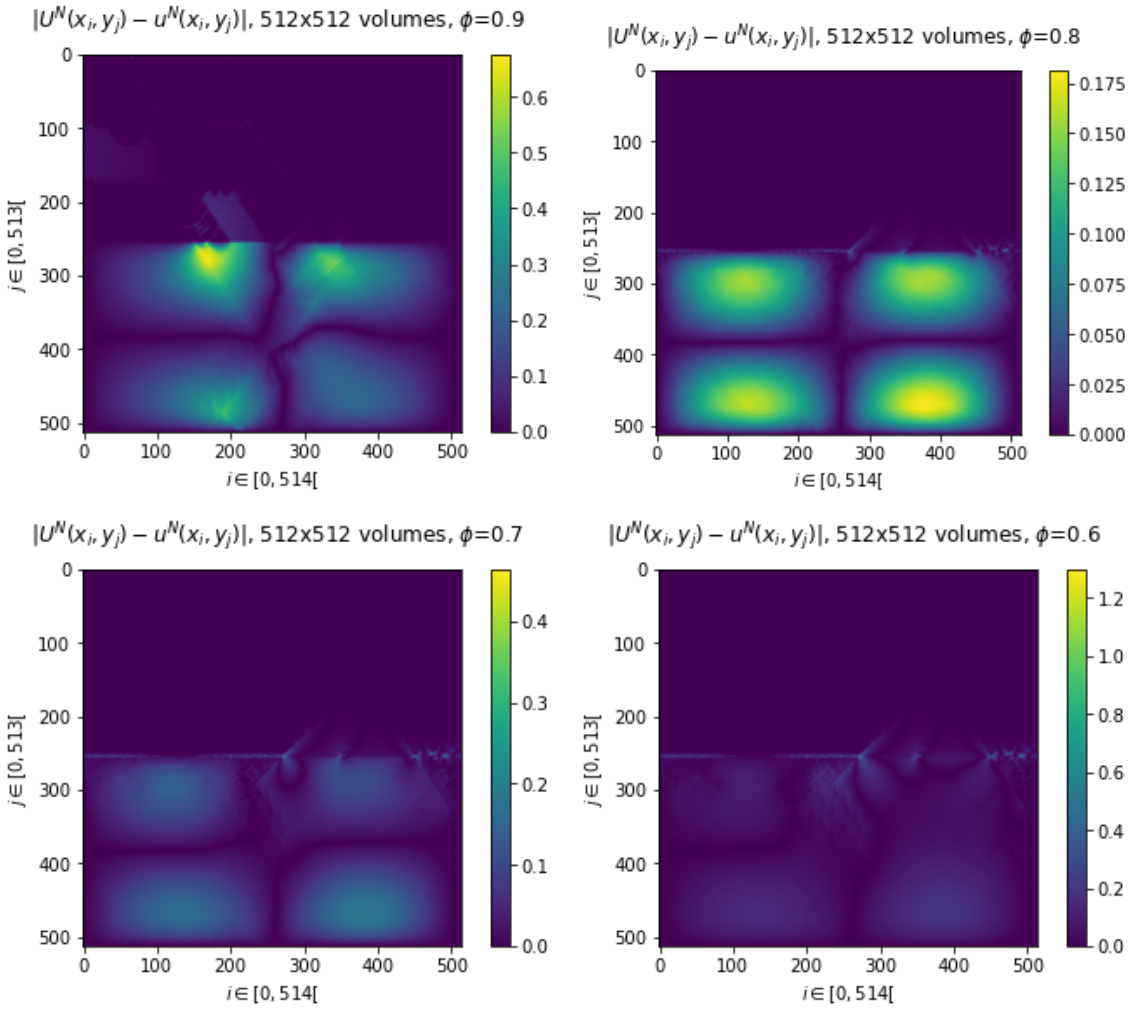

FIGURE 4.2 – Champ d'erreur pour la variable U_h

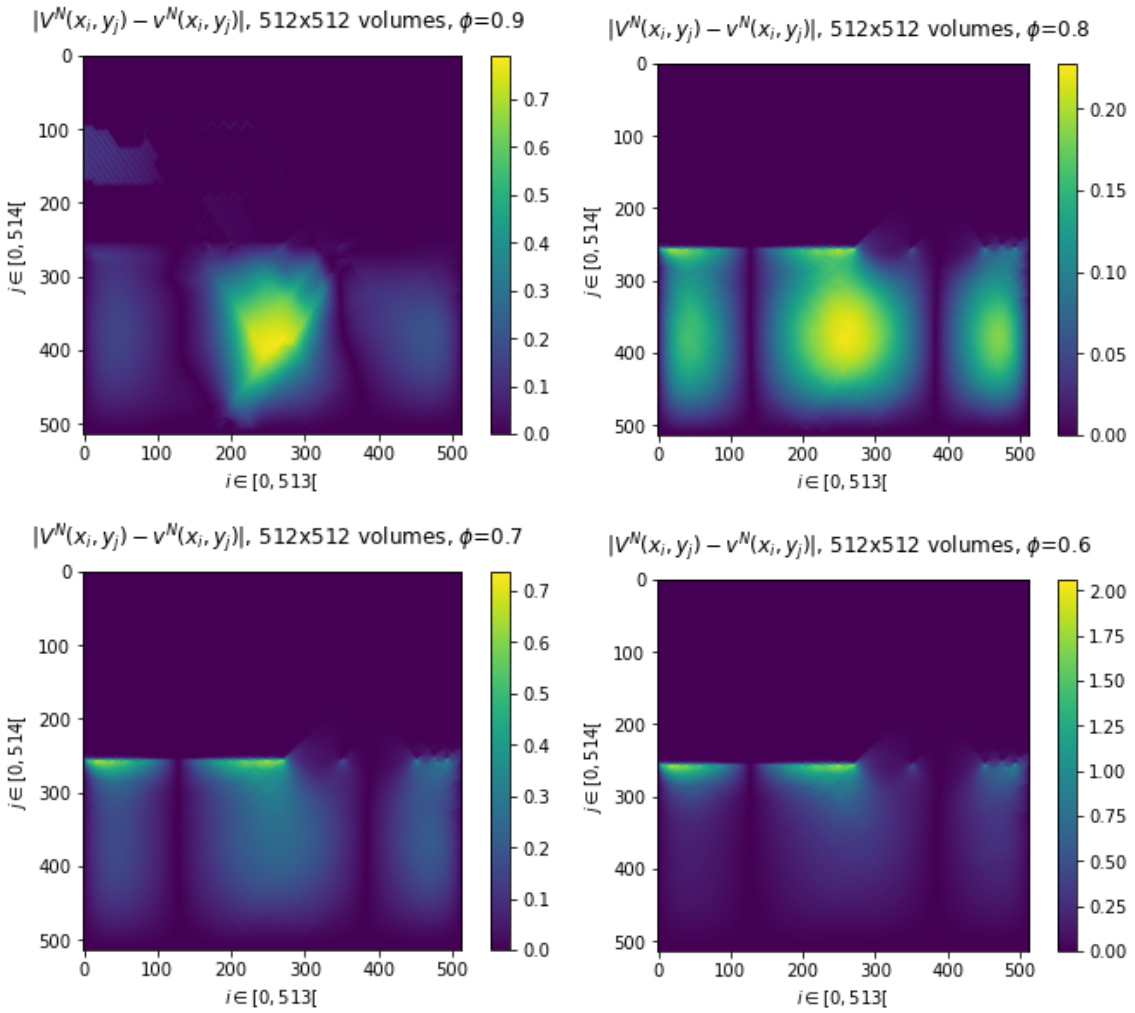

FIGURE 4.3 – Champ d'erreur pour la variable V_h

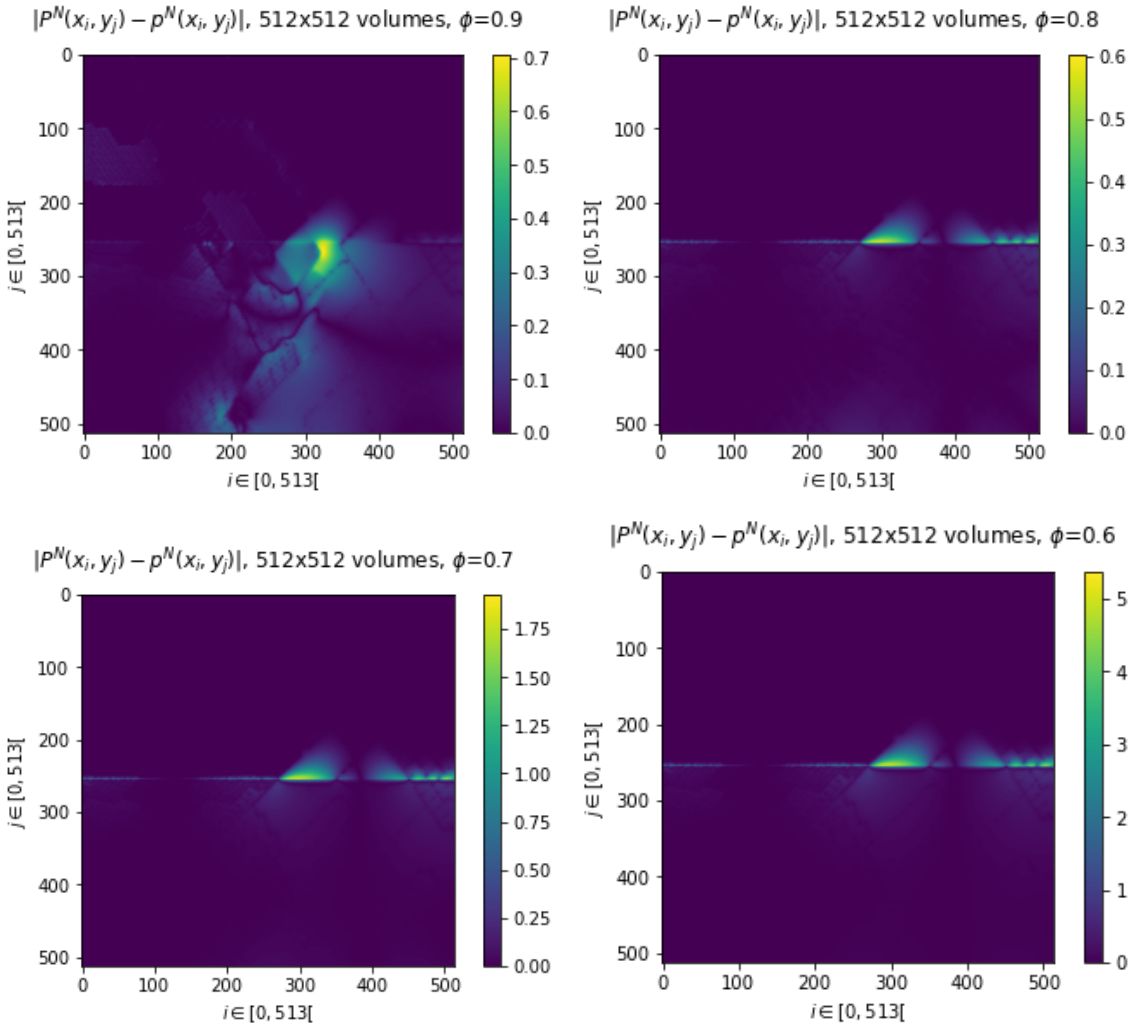

FIGURE 4.4 – Champ d'erreur pour la variable P_h

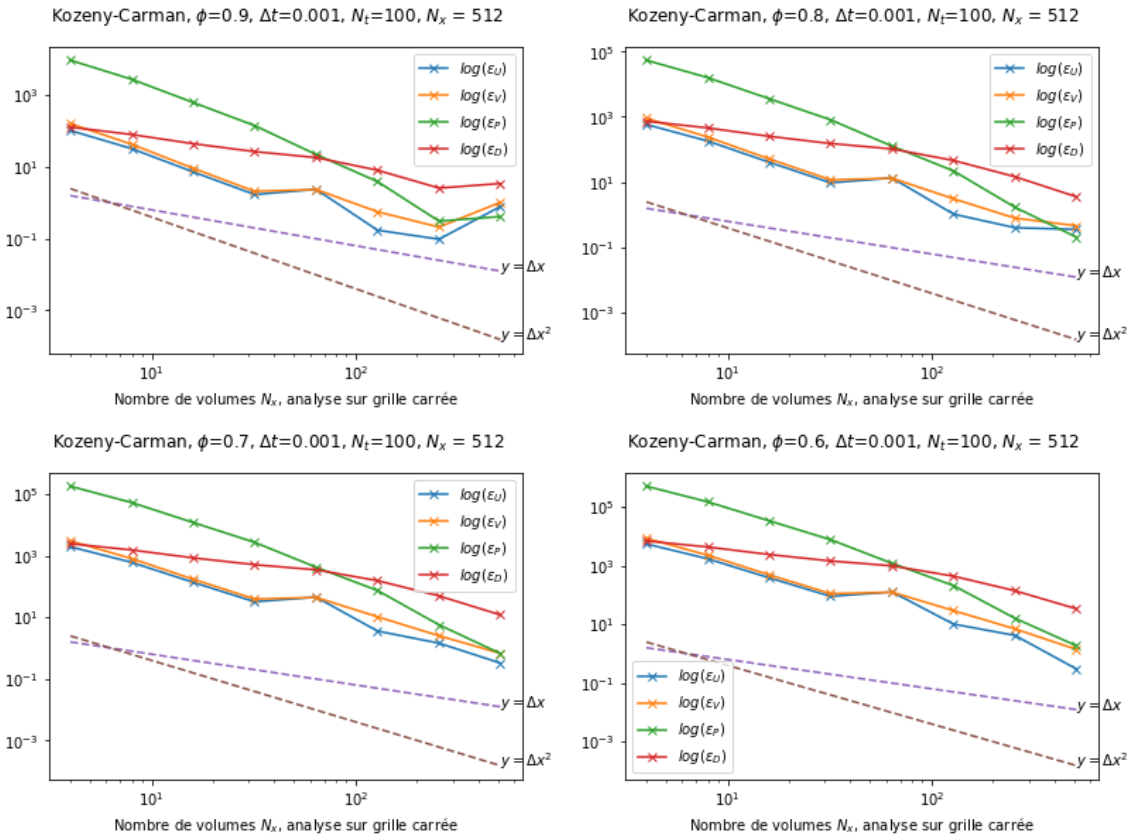

FIGURE 4.5 – Analyse des graphes

Conclusion

En cinq mois de stage, j'ai exploré différents aspects du travail de l'ingénieur en analyse numérique. Pour commencer, le choix des outils et des technologies permettant d'effectuer le calcul souhaité, ici la résolution de systèmes linéaires associés aux systèmes de type Stokes. Je me suis ouvert à une méthode de discrétisation des lois de conservations, les volumes finis, et aux concepts qui y sont associés : les objets que sont les maillages, et leur caractère admissible ou non ; les choix à effectuer lors de la discrétisation et éventuellement leur impact sur la simulation, je pense au choix de la représentation de la grandeur étudiée, valeur moyenne, valeur ponctuelle, ou aure, les choix de discrétisation des flux : une fois obtenue l'intégrale surfacique, il faut à nouveau approcher cette intégrale, plutôt que de discrétiser par le schéma centré d'ordre 1, nous aurions pu regarder une discrétisation de ces intégrales par éléments finis, etc. Une fois le maillage choisi et le problème discrétisé, il s'agit de montrer l'existence et l'unicité d'une solution à ce nouveau problème. Enfin, il s'agit d'étudier la stabilité des flux, par l'estimation point par point d'abord de l'erreur entre une solution classique et une solution numérique, calculée par le schéma, puis son estimation en norme. Il ne faut pas oublier de se pencher sur la consistance de l'opérateur discret, comprendre si son inverse existe et reste borné pour de petits pas de maillages. Alors on peut établir la convergence et, si une solution analytique est connue, implémenter le schéma et valider le code produit en analysant l'erreur en fonction du pas de maillage pour différentes normes discrètes. Bien que certains faits et résultats sont très classiques, les méthodes de volumes finis sont un domaine d'étude en soi que j'espère pouvoir creuser dans la suite de ma vie professionnelle.

Ensuite est venu le temps d'apprendre la mécanique des fluides, et de consolider mes connaissances en analyse. J'ai probablement passé trop de temps sur ces points mais qui me paraissent pourtant fondamentaux du point de vue la méthode. Il me semble que l'établissement du caractère bien posé d'un problème est incontournable avant de passer à la simulation. Cette partie sur le problème de Stokes m'a permis de fixer les étapes de l'analyse d'un problème et de mobiliser les arguments d'analyse fonctionnelle étudiés dans les différents cours que j'ai pu suivre cette année afin de comprendre le développement de cette analyse. Du reste, j'ai appris au cours de ce stage la condition inf sup et les conditions de compatibilités pour le problème de Stokes. C'est ce point qui fait que ce n'est pas, pour moi, simplement de la redite mais un véritable pas en avant dans mes connaissances.

La suite du mémoire a été l'occasion de dérouler les méthodes et d'asseoir les découvertes vues dans les deux premières parties. Le système de Brinkman en particulier apparaît comme étant un système de Stokes généralisé, avec un terme d'inertie, appelé terme de Darcy dans le cas des milieux poreux. On peut alors montrer l'existence et l'unicité d'une solution par une méthode lagrangienne, utiliser les résultats d'optimisation dans les espaces de Banach pour en rédiger les preuves, et se rapporter sans retenue aux ouvrages contenant les résultats nécessaires.

Enfin, l'étude de la superposition des deux couches, fluides et poreuses, n'aura été qu'effleurée, sur les deux dernières semaines. Les mêmes qui ont vu naître ce mémoire. Je n'y ai pas passé assez de temps pour dire quoi que ce soit d'intelligent à ce sujet. Ici de nouvelles difficultés apparaissent avec la notion de saut, ce qui implique l'apparition de nouveaux espaces fonctionnels donc un nouveau cadre pour discuter l'existence et l'unicité, de nouvelles difficultés dans le choix du maillage, puis une nouvelle discrétisation du problème.

Bibliographie

[Bré83] Haïm Brézis. *Analyse fonctionnelle : théorie et applications*. Collection Mathématiques appliquées pour la maîtrise. Masson, 1983. isbn : 9782225771989. url : https://books.google.fr/books?id=7SXvAAAAMAAJ.

[EGH00] Robert Eymard, Thierry Gallouët et Raphaèle Herbin. « Finite Volume Methods ». In : *Solution of Equation in Rn (Part 3), Techniques of Scientific Computing (Part 3)*. Sous la dir. de J. L. Lions et Philippe Ciarlet. T. 7. Handbook of Numerical Analysis. Elsevier, 2000, p. 713-1020. doi : 10.1016/S1570-8659(00)07005-8. url : https://hal.archives-ouvertes.fr/hal-02100732.

[BMR04] C. Bernardi, Y. Maday et F. Rapetti. *Discrétisations variationnelles de problèmes aux limites elliptiques*. Mathématiques et Applications. Springer, 2004. isbn : 9783540213697. url : https://books.google.fr/books?id=BLFYqcgkCocC.

[GMS06] Jean-Luc Guermond, Peter Dimitrov Minev et Jie Shen. « An overview of projection methods for incompressible flows ». In : *Computer Methods in Applied Mechanics and Engineering* 195 (2006), p. 6011-6045.

[Cal13] Jean-Paul Caltagirone. *Physiquement des écoulements continus*. Springer, 2013. isbn : 9783642395093.

[AGO17] Philippe Angot, Benoît Goyeau et J. Alberto Ochoa-Tapia. « Asymptotic modeling of transport phenomena at the interface between a fluid and a porous layer : Jump conditions ». In : *Physical Review E* 95.6 (juin 2017). doi : 10.1103/PhysRevE.95.063302. url : https://hal.archives-ouvertes.fr/hal-01583856.

[Ang18] Philippe Angot. « WELL-POSED STOKES/BRINKMAN AND STOKES/DARCY COUPLING REVISITED WITH NEW JUMP INTERFACE CONDITIONS ». In : *ESAIM : Mathematical Modelling and Numerical Analysis* 52.5 (2018). Accepted on November 24, 2017, p. 1875-1911. doi : 10.1051/m2an/2017060. url : https://hal.archives-ouvertes.fr/hal-01635289.

[Ang21] Philippe Angot. « Solvability of the variable-viscosity fluid-porous flows coupled with an optimal stress jump interface condition ». working paper or preprint. Fév. 2021. url : https://hal.archives-ouvertes.fr/hal-03172378.