Bureau d'étude Éléments Finis Mixtes

Haine, Ghislain

10 novembre 2022

Ce bureau d'étude est noté sur compte-rendu, en binôme. Le compte-rendu, **qui ne portera que sur la seconde partie**, devra être composé de 3 à 8 pages et être envoyé **par mail** à `ghislain.haine@isae.fr` en pdf au plus tard le **Vendredi 16 décembre 2022**.

Introduction

GetFEM : c'est quoi ?

GetFEM (`https://getfem.org/`) est une bibliothèque libre et gratuit de résolution d'équations aux dérivées partielles par éléments finis (Français : Développeur historique Yves Renard – INSA de Lyon). Il se compare à de nombreux autres moyens, dans la recherche comme dans l'industrie, mis à disposition des ingénieurs et chercheurs pour leurs calculs.

En effet, si des logiciels tels que MATLAB et Maple sont très utiles pour de premières évaluations, il n'est pas envisageable d'y modéliser tout un problème technique complexe, régit par des équations aux dérivées partielles couplées, en trois dimensions, et dynamiques...

On dispose de plusieurs solutions pour cela, en voici quelques unes :

- FreeFEM++ : un logiciel open-source d'éléments finis (Développeur historique Frédéric Hecht – Labo Jacques Louis Lions, Paris) ;
- XLiFE++ : une bibliothèque C++ étendue d'éléments finis et de résolution intégrale (Développée par l'équipe INRIA P.O.e.m.s et le laboratoire IRMAR de Rennes) ;
- FEniCS / FEniCSx Projects : bibliothèque éléments finis en Python ;
- GetDP : un environnement général de traitement des problèmes discrets (Développeur historique Christophe Geuzaine – Montefiore Institute, University of Liège) ;
- DUNE Numerics : un environnement modulable de résolution des EDP par différentes méthodes ;
- COMSOL : un logiciel de simulation multiphysique (payant).

Pourquoi GetFEM ?

Tout simplement parce qu'il est très simple à prendre en main et dispose de plusieurs API, dont python qui est choisi pour ce BE.

Avant de commencer, il faut :

1. Installer une version récente de GMSH pour gérer le maillage et la visualisation.
2. Installer un environnement conda, typiquement donner par le fichier .yml :

```
name: sxs
channels:
  - conda-forge
dependencies:
  - python=3
  - getfem
  - matplotlib
  - numpy
  - petsc
  - petsc4py
  - spyder
  - slepc
  - slepc4py
```

3. J'ai personnellement choisi 'spyder' comme IDE, mais rien n'est imposé pour le code python bien sûr.

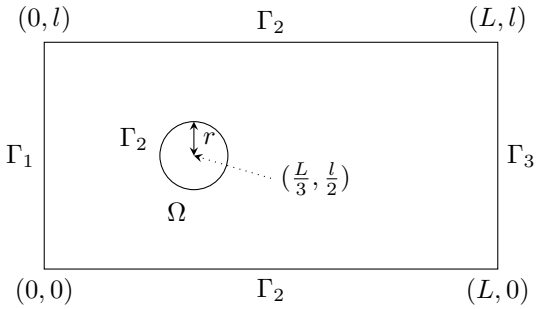
Avant de démarrer, il faut exécuter les commandes `module load` relatives à GMSH et python.

1 Première partie : prise en main de GetFEM

On souhaite résoudre le problème suivant

$$\begin{cases} -\Delta u(x,y) = -1, & \forall (x,y) \in \Omega, \\ u(x,y) = 0, & \forall (x,y) \in \Gamma_1, \\ \dfrac{\partial}{\partial n} u(x,y) = 0, & \forall (x,y) \in \Gamma_2, \\ u(x,y) = -1, & \forall (x,y) \in \Gamma_3, \end{cases} \tag{1}$$

où le domaine Ω est donné par la figure suivante



Le domaine Ω et sa frontière $\partial\Omega = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$.

1.1 GMSH et le fichier .geo

GMSH est un mailleur puissant qui se base sur une définition "géométrique" du domaine. On définit d'abord des points, puis des courbes liant ces points, puis des chemins permettant enfin de définir des surfaces (en 2D, en 3D on poursuit la construction des volumes par des surfaces fermées). Il dispose d'une API python, mais nous ne l'utiliserons pas dans ce BE.

Le fichier permettant de mailler le domaine Ω est donné :

```
/*--------------------------*/
//    File Flag.geo
/*--------------------------*/

// Parameters
If (!Exists(h)) h=0.08; EndIf // Mesh size
If (!Exists(L)) L=2.; EndIf // Width
If (!Exists(l)) l=1.; EndIf // Height
If (!Exists(xc)) xc=L/3.; EndIf // x-center
If (!Exists(yc)) yc=l/2.; EndIf // y-center
If (!Exists(R)) R=l/6.; EndIf // Radius

// Points
Point(1) = {xc,yc,0,h};
Point(2) = {xc+R,yc,0,h};
Point(3) = {xc,yc+R,0,h};
Point(4) = {xc-R,yc,0,h};
Point(5) = {xc,yc-R,0,h};
Point(6) = {0,0,0,h};
Point(7) = {L,0,0,h};
Point(8) = {L,l,0,h};
Point(9) = {0,l,0,h};

// Curves (indices to use in GetFEM)
Circle(11) = {2,1,3};
Circle(12) = {3,1,4};
Circle(13) = {4,1,5};
Circle(14) = {5,1,2};
Line(15) = {6,7};
Line(16) = {7,8};
Line(17) = {8,9};
Line(18) = {9,6};

// Paths
Line Loop(21) = {11,12,13,14};
Line Loop(22) = {15,16,17,18};

// Surfaces (indices to use in GetFEM)
Plane Surface(31) = {-21,22};

// Physical Groups (Not needed for GetFEM)
Physical Line("Bottom") = {15};
Physical Line("Right") = {16};
Physical Line("Top") = {17};
Physical Line("Left") = {18};
Physical Line("Hole") = {11,12,13,14};
Physical Surface("Omega") = {31};
```

À faire : Afin de visualiser ce domaine, placez-vous dans le dossier contenant `Flag.geo` avec la console, et tapez `gmsh Flag.geo &`.

À faire : Seule la géométrie apparaît. Pour mailler le domaine (“à la souris”), déplier le module “Mesh”

et cliquez sur “2D”. Un maillage apparaît, le fichier .geo fonctionne. Vous pouvez fermer la fenêtre GMSH.

1.2 Le maillage pour GetFEM

Le principe sera d'appeler GMSH pour mailler et sauver le domaine défini dans le fichier .geo, puis d'importer ce maillage dans l'environnement de GetFEM.

À faire : Dans votre IDE (`spyder &`), ouvrez le fichier `Poisson.py`.

Après l'import des différentes bibliothèques :

```
#%% Initialization

import os
import numpy as np

# import basic modules
import getfem as gf
```

On appelle GMSH puis on importe le maillage obtenu dans un objet GetFEM :

```
#%% Import gmsh file

# External call to gmsh client (a python API exists)
# with (re-)definition of the mesh size (see .geo file)
h = 0.04
os.system('gmsh -2 Flag.geo -setnumber h '+str(h))

m = gf.Mesh('import','gmsh','Flag.msh')

print('')
print('===')

print(m.regions(),'region ids from .geo file (not the Physical ones!)')
print(m.memsize(),'bytes used to store the mesh')

m.display()
```

De ce maillage maintenant au format GetFEM, on définit les régions à l'aide des indices donnés dans le fichier .geo (attention, il NE s'agit PAS des “physical groups” mais bien des entités point, line, surface, etc.) :

```
#%% Prepare boundaries

# Merge the 4 regions for the hole
m.region_merge(11,12)
m.region_merge(11,13)
m.region_merge(11,14)
# Delete the 3 regions now in 11
m.delete_region([12,13,14])
print(m.regions(),'remaining region ids DIFFERENT from .geo file (definition of the "hole" boundary)')
# Name the indices for the sake of readability
BOTTOM_BOUND=15; RIGHT_BOUND=16; TOP_BOUND=17; LEFT_BOUND=18; HOLE_BOUND=11;
print('Bottom id: ', BOTTOM_BOUND, ', Right id: ', RIGHT_BOUND, ', Top id: ', TOP_BOUND,\
      ', Left id: ', LEFT_BOUND, ', Hole id: ', HOLE_BOUND)
# To be complete, we also name the domain
OMEGA=31;
print('Omega id: ', OMEGA)
```

Tout est prêt pour passer au problème variationnel à résoudre.

1.3 La définition du modèle

GetFEM fonctionne par association d'objets. À un maillage, on associe des éléments finis et une méthode d'intégration. Aux éléments finis, on associe un modèle constitué d'éléments appelés `brick` (plus ou moins identifiables aux termes de la formulation variationnelle), lesquelles sont associées à la méthode d'intégration.

D'abord, on définit un objet "élément fini" et un objet "méthode d'intégration" associés au maillage :

```
#%% Finite element and integration method

# For the field u
mfu = gf.MeshFem(m,1) # The FE space is on mesh m, for a scalar field
mfu.set_fem(gf.Fem('FEM_PK(2,2)')) # PK-Lagrange, continuous, on simplex of dim. 2, order 2
# mfu.set_classical_fem(2) # Do = PK-Lagrange, continuous, on simplex of dim. 2, order 2

# An exact integration associated to the mesh
mim = gf.MeshIm(m, gf.Integ('IM_TRIANGLE(4)')) # integration pol. of order 4 is exact
```

On définit ensuite le modèle et ces différentes variables :

```
#%% Model definition

# Data (may vary in x and y)
f = mfu.eval('-1.') # RHS
gn = mfu.eval('0.') # Neumann
gd_l = mfu.eval('0.') # Dirichlet
gd_r = mfu.eval('-1.') # Dirichlet

# Model, variables
md = gf.Model('real') # Can handle complex fields
md.add_fem_variable('u', mfu) # Name the variable to compute

# Data initialization
md.add_initialized_fem_data('f', mfu, f)
md.add_initialized_fem_data('gn', mfu, gn)
md.add_initialized_fem_data('gd_l', mfu, gd_l)
md.add_initialized_fem_data('gd_r', mfu, gd_r)
```

Enfin, on définit chaque terme de la formulation variationnelle à l'aide de `brick` dédiées, ainsi que les conditions essentielles :

```
#%% Bricks

#md.add_Laplacian_brick(mim, 'u') # Laplacian brick
md.add_linear_term(mim, 'Grad_u:Grad_Test_u') # Laplacian variational formulation

md.add_source_term_brick(mim, 'u', 'f')
md.add_source_term_brick(mim, 'u', 'gn', BOTTOM_BOUND)
md.add_source_term_brick(mim, 'u', 'gn', TOP_BOUND)
md.add_source_term_brick(mim, 'u', 'gn', HOLE_BOUND)

# Essential conditions with simplification (other ways for Dirichlet imposition exist)
md.add_Dirichlet_condition_with_simplification('u', LEFT_BOUND, 'gd_l')
```

```
md.add_Dirichlet_condition_with_simplification('u', RIGHT_BOUND, 'gd_r')

md.brick_list() # Get some infos on the model now defined
```

Notons que des bricks élémentaires sont déjà disponibles au sein de GetFEM : le laplacien peut donc directement se définir via `add_Laplacian_brick`. Cela s'avère particulièrement utile dans certains cas comme nous le verrons dans l'exemple de l'élasticité.

Il reste maintenant à résoudre le modèle :

```
#%% Solve with the simplest way

md.solve()
u = md.variable('u') # extraction of the computed solution
mfu.export_to_pos('Laplacian.pos',mfu,u,'u') # save as gmsh .pos file for visualization
```

Dans la suite, nous souhaiterons étudier les différents type de résolution proposés dans le poly pour le problème de Stokes. On a donc besoin d'accéder aux matrice et vecteur de la "formulation algébrique", puis de résoudre le système :

```
#%% Solve with algebra

A = md.tangent_matrix()
A.to_csc()
b = md.rhs()

u_mumps = np.transpose(gf.linsolve('mumps',A,b))
mfu.export_to_pos('Laplacian_mumps.pos',mfu,u_mumps,'u (mumps)')

error = u-u_mumps
print('L-sup error w.r.t GetFEM solution: ',np.max(np.abs(error)))
```

À faire : Lancez le fichier `Poisson.py`. Dans la console, lancez `gmsh Laplacian.pos &` pour visualiser la solution (donnée par GetFEM). Pour améliorer la vue, ouvrez le menu `Tools` et sélectionnez `Options`. Choisissez `View [1]` à gauche puis `Transfo` dans les onglets à droite et mettez 1 dans la case `Raise` correspondant à la coordonnée z. Vous pouvez maintenant observer la déformation de la membrane en faisant varier l'angle de vue en 3D à l'aide de la souris (vous pouvez fermer le menu `Options`).

1.4 Le problème de l'élasticité linéaire

On s'intéresse maintenant au problème, déjà traité dans un précédent BE, de l'élasticité linéaire.

$$\begin{cases} -\text{div}\left[\sigma(u(x,y))\right] = \begin{pmatrix}0\\0\end{pmatrix}, & \forall (x,y) \in \Omega, \\ u(x,y) = \begin{pmatrix}0\\0\end{pmatrix}, & \forall (x,y) \in \Gamma_1, \\ \dfrac{\partial}{\partial n} u(x,y) = \begin{pmatrix}0\\0\end{pmatrix}, & \forall (x,y) \in \Gamma_2, \\ \dfrac{\partial}{\partial n} u(x,y) = \begin{pmatrix}1\\0\end{pmatrix}, & \forall (x,y) \in \Gamma_3, \end{cases} \tag{2}$$

où σ est le tenseur des contraintes (il est symétrique), qui dépend du déplacement u par l'intermédi-aire d'une loi de comportement (ici la loi de Hooke). Il s'écrit de manière générale

$$\sigma_{ij} = \sum_{k,l} C_{ijkl} \mathcal{E}_{kl}(u),$$

où C est le tenseur des compliances élastiques (il est également symétrique) et $\mathcal{E}$ est le tenseur de déformation (il est aussi symétrique). Dans le cas linéarisé, ce dernier s'écrit

$$\mathcal{E}_{ij} = \frac{1}{2}\left(\frac{\partial}{\partial x_j}u_i + \frac{\partial}{\partial x_i}u_j\right).$$

On considère ici le cas d'un matériau homogène isotrope : C ne dépend plus que de deux coefficients. On choisit de travailler avec le formalisme faisant intervenir le module d'Young E et le coefficient de Poisson ν. Nous donnerons plus bas les formules de la loi de Hooke et du champ de contraintes de Von Mises (elles diffèrent en 2D et 3D).

La formulation variationnelle associée au problème (2) est : trouver $u \in V$ tel que pour tout $v \in V$, on ait $a(u,v) = l(v)$ où

$$a(u,v) = \int_\Omega \sum_{i,j} \sigma_{ij}(u)\mathcal{E}_{ij}(v),$$

et

$$l(v) = \int_\Omega f \cdot v + \int_{\Gamma_2} g_n \cdot v.$$

En deux dimensions, la loi de Hooke s'écrit

$$\begin{cases} \sigma_{xx} = \dfrac{E}{1-\nu^2}\left(\mathcal{E}_{xx} + \nu\mathcal{E}_{yy}\right) \\ \sigma_{yy} = \dfrac{E}{1-\nu^2}\left(\nu\mathcal{E}_{xx} + \mathcal{E}_{yy}\right) \\ \sigma_{xy} = \sigma_{yx} = \dfrac{E}{1+\nu}\mathcal{E}_{xy}. \end{cases}$$

Le champ de contraintes de Von Mises à représenter est quant à lui donné par

$$\sigma_{VM} = \left(\sigma_{xx}^2 + \sigma_{yy}^2 + 3\sigma_{xy}^2 - \sigma_{xx}\sigma_{yy}\right)^{\frac{1}{2}}.$$

À faire : À l'aide de l'aide en ligne de GetFEM, définissez les bricks correspondant au problème.

À faire : Ouvrez le fichier `Elasticity.pos` à l'aide de GMSH.

À faire : Pour améliorer la vue, ouvrez le menu `Tools` et sélectionnez `Options`. Choisissez `View [0]` et `View [2]` (en maintenant la touche Ctrl du clavier) à gauche puis `Transfo` dans les onglets à droite. On souhaite appliquez la déformation au maillage et au champ de contrainte de Von Mises. Cochez `Use general transformation expressions` et sélectionnez `View [1]` comme `Data source`. Ajustez ensuite le facteur d'échelle en mettant 100 à `Factor`. Vous pouvez maintenant observer la déformation du domaine (vous pouvez fermer le menu `Options`). Vous pouvez décocher u à gauche pour retirer le champs de vecteurs.

2 Deuxième partie : résolution du problème de Stokes

On s'intéresse au problème de *Stokes stationnaire* en 2 dimensions. **L'objectif principal est d'analyser les différences entre les résolutions proposées en cours sur le problème algébrique final, en fonction des différents paramètres.**

- On supposera une condition d'adhérence aux parois et autour du "trou" : ceci ce traduit par une condition de Dirichlet homogène sur le champ de vitesse sur Γ_2.
- On supposera un profil quadratique de type Poiseuille en entrée : il s'agit d'une condition de Dirichlet non-nulle sur Γ_1.

- On supposera un flux “aspirant” en sortie, se traduisant par une condition de Fourier-Robin (de type “impédance”) sur Γ_3 : $p = \nu\ \overrightarrow{\boldsymbol{u}} \cdot \overrightarrow{\boldsymbol{n}}$. On notera que cette condition permet de fixer la constante à définir pour le champs de pression.

- Enfin, on supposera qu'il n'y a aucune charge extérieure agissant sur le système.

Les équations d'intérêt à discrétiser sont alors proches de celles vues en cours (seules les conditions aux limites diffèrent), *i.e.*

$$\begin{cases} -\nu\Delta\overrightarrow{\boldsymbol{u}}(\overrightarrow{\boldsymbol{x}}) + \nabla p(\overrightarrow{\boldsymbol{x}}) = \begin{pmatrix}0\\0\end{pmatrix}, & \forall\ \overrightarrow{\boldsymbol{x}} \in \Omega, \\ \operatorname{div}\overrightarrow{\boldsymbol{u}}(\overrightarrow{\boldsymbol{x}}) = 0, & \forall\ \overrightarrow{\boldsymbol{x}} \in \Omega, \\ \overrightarrow{\boldsymbol{u}}(\overrightarrow{\boldsymbol{x}}) = \begin{pmatrix}4y(1-y)\\0\end{pmatrix}, & \forall\ \overrightarrow{\boldsymbol{x}} \in \Gamma_1, \\ \overrightarrow{\boldsymbol{u}}(\overrightarrow{\boldsymbol{x}}) = \begin{pmatrix}0\\0\end{pmatrix}, & \forall\ \overrightarrow{\boldsymbol{x}} \in \Gamma_2, \\ p(\overrightarrow{\boldsymbol{x}}) = \nu\overrightarrow{\boldsymbol{u}} \cdot \overrightarrow{\boldsymbol{n}}, & \forall\ \overrightarrow{\boldsymbol{x}} \in \Gamma_3, \end{cases} \tag{3}$$

où $\partial_n \overrightarrow{\boldsymbol{u}}$ désigne le vecteur des dérivées normales des composantes de $\overrightarrow{\boldsymbol{u}}$.

À faire : Complétez le fichier `Stokes.py`.

À faire : Étudiez l'influence du paramètre h du maillage sur le temps de résolution, le conditionnement, la charge mémoire et la valeur de la divergence du champs de vitesse.

À faire : Étudiez l'influence du paramètre ε dans la méthode de pénalisation.

À faire : Étudiez l'influence du paramètre γ dans la méthode de dualité-pénalisation.