Mat-7430 : Méthodes numériques avancées pour les EDP
Guide d'utilisation du code Matlab d'éléments finis

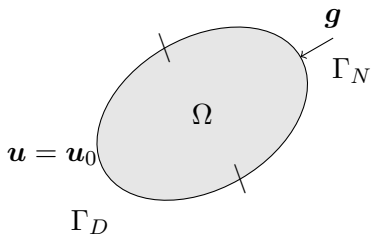
Le présent document est un guide d'utilisation d'un code Matlab qui vise à résoudre des problèmes de type diffusion ou d'élasticité linéaire en 2D ou 3D par la méthode des éléments finis. Le problème aux limites de diffusion s'écrit

$$\begin{cases} -\mathrm{div}(k(x)\nabla u) &= f & \text{dans } \Omega, \text{ (Dirichlet)} \\ u &= g & \text{sur } \Gamma_D, \\ k\dfrac{\partial u}{\partial n} &= h & \text{sur } \Gamma_N, \text{ (Neumann)} \\ k\dfrac{\partial u}{\partial n} + au &= b & \text{sur } \Gamma_R, \text{ (Robin)} \end{cases}$$

où $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_R$. Il est facilement modifiable pour résoudre d'autres types de problèmes aux limites.

Le problème d'élasticité linéaire correspond au problème aux limites

$$\begin{cases} \text{-div } \sigma(\varepsilon(\boldsymbol{u})) &= \boldsymbol{f} & \text{in } \Omega \\ \boldsymbol{u} &= \boldsymbol{u}_0 & \text{on } \Gamma_D \\ \sigma \cdot \boldsymbol{n} &= \boldsymbol{g} & \text{on } \Gamma_N \end{cases}$$



où $\sigma(\varepsilon(\boldsymbol{u})) = 2\eta\varepsilon(\boldsymbol{u}) + \lambda \mathrm{tr}\ \varepsilon(\boldsymbol{u})\ I$ avec les notations

σ	$=$	$(\sigma_{ij})_{3\times 3}$	stress tensor
$\varepsilon(\boldsymbol{u})$	$=$	$(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^t)/2$	linearized deformation tensor
$(\mathrm{div}\ \sigma)_i$	$=$	$\sum_j \frac{\partial \sigma_{ij}}{\partial x_j}$	
$(\nabla \boldsymbol{u})_{ij}$	$=$	$\frac{\partial u_i}{\partial x_j}$	

η and λ sont les coefficients de Lamé qui sont reliés aux modules de Young E et le coefficient de Poisson ν par

$$\eta = \frac{E}{2(1+\nu)}, \qquad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

Dans la pratique, il arrive souvent que les frontières $\Gamma_D, \Gamma_N, \Gamma_R$ soient composées de plusieurs courbes (ou surfaces en 3D). Par exemple, le bord des conditions aux limites de Dirichlet peut s'écrire $\Gamma_D = \bigcup_i \Gamma_{D_i}$. Dans ce cas, la condition aux limites de Dirichlet est remplacée par plusieurs conditions de la forme

$$u = g_i \qquad \text{sur } \Gamma_{D_i}.$$

et de même pour les conditions aux limites de Neumann ou de Robin.

Avant de résoudre un de ces problèmes, vous avez besoin

- d'utiliser obligatoirement le logiciel Gmsh comme mailleur,
- d'un fichier $nomMail$ d'extension $.msh$ qui contient les données du maillage provenant de Gmsh,
- de connaître le nombre d'éléments du maillage,
- les numéros des frontières fournies par Gmsh qui servent à préciser les différents types de conditions aux limites. Ces numéros sont disponibles dans le fichier $nomMail.geo$.

Le fichier probDiffusion.m permet d'exécuter un problème de diffusion autant en 2D que 3D; il suffit de taper probDiffusion à la ligne de commande. De même, le .m fichier probElasticite.m permet d'exécuter un problème délasticité.

Au préalable, il faut fournir le nom du maillage, par exemple

```
nomFichier = 'Maillages2D/carre';
```

tout en précisant le chemin d'accès. Il est important de préciser le nombre d'éléments du maillage, (information disponible à partir de Gmsh) par exemple

```
CondLim.nel = 3482;    % nombre d'elements du maillage P1 (carre)
```

Description du fichier probDiffusion

Les paramètres physiques de l'équation $-\text{div}(k(x)\nabla u) = f$ sont regroupées dans la structure **Param**:

- Param.k = fonction k(x) doit être scalaire,
- Param.f = fonction f.

Les informations relatives au type d'éléments finis sont regroupées dans la structure **TypeEF**:

- TypeEF.triangle = true/false; Maillage 2D en triangle sinon en quad (pas actif en 3D)
- TypeEF.tetra = true/false; Maillage 3D en tetra sinon en hex (pas actif en 2D)
- TypeEF.type = décrit le type d'éléments finis = P2H ou P1 ou P2 ou Q1 ou Q2H ou Q2
- TypeEF.typeGeo = décrit le type de transformation géométrique utilisé: = P1 ou P2 ou Q1 ou Q2
- TypeEF.ordre = l'ordre de l'élément fini
- TypeEF.degGeo = degré de la transformation géométrique (1 ou 2) (élément isoparamétrique)

- TypeEF.nd = 2 ou 3 la dimension du problème
- TypeEF.ndim = 1; le nombre de composantes par noeud (= 1 si diffusion) (= 2 ou 3 si élasticité)
- TypeEF.ordreIntg = 4; le degré d'exactitude du schéma d'intégration (2,3,4 pour triangle) (2,3,5 pour tetra)
- TypeEF.ordreIntgFace = 5;

Toutes les informations relatives aux conditions aux limites sont regroupées dans la structure **CondLim**

- listeDir = liste des indices des frontières Γ_{D_i}, c'est-à-dire les numéros de Gmsh des Physical Line associés aux Γ_{D_i}. On pose CondLim.listeFrontiereDir = listeDir;
- CondLim.fonctFrontiereDir = la liste des fonctions g_i (cell array)
- les fichiers $probDiffusionCondLimites.m$ (cas 2D) et $prob3DDiffusionCondLimites$ contiennent une instruction switch qui permet de choisir un ensemble de conditions aux limites via la variable prob=1,2, etc

```
Exemple:
listeDir = [9, 11]; % indices des frontieres de Dirichlet fournis par Gmsh
f1 =@(x,y,z) cos(x.*y);
f2 =@(x,y,z) 0;
CondLim.fonctFrontiereDir{1} = f1;
CondLim.fonctFrontiereDir{2} = f2;
```

De même pour les conditions aux limites de Neumann (et aussi de Robin), on aura $\Gamma_N = \bigcup_i \Gamma_{N_i}$ pour lequel on applique la condition aux limites de Neumann

$$k\frac{\partial u}{\partial n} = h_i \qquad \text{sur } \Gamma_{N_i}.$$

- listeNeumann = liste des indices des frontières Γ_{N_i}, c'est-à-dire les numéros de Gmsh des Physical Line associés aux Γ_{N_i}. On pose CondLim.listeFrontiereNeumann = listeNeumann;
- Si listeNeumann = [ ], il n'y a pas de condition aux limites de Neumann.
- CondLim.fonctFrontiereNeumann = la liste des fonctions h_i (cell array)

Le code inclut aussi des conditions aux limites de Robin avec la même syntaxe. Pour enlever ces conditions aux limites, il faut poser listeRobin = [ ].

```
Exemple:
listeNeumann = [9:11]; % indices des frontieres de Neumann fournis par Gmsh
f1 =@(x,y,z) 64*(2*y-1);
f2 =@(x,y,z) 64*(2*x-1);
f3 =@(x,y,z) -64*(2*y-1);
CondLim.fonctFrontiereNeumann{1} = f1;
CondLim.fonctFrontiereNeumann{2} = f2;
CondLim.fonctFrontiereNeumann{3} = f3;
```

Pour les conditions de Robin, les fonctions a et b sont déclarées selon la règle:

```
% conditions aux limites de Robin
 a =@(x,y) 1;
 b =@(x,y) 1+x;
 CondLim.fonctFrontiereRobin{indice de la frontiere,1} = a;
 CondLim.fonctFrontiereRobin{indice de la frontiere,2} = b;
```

Voici des remarques importantes sur l'utilisation du code

- Toutes les fonctions associées à un problème donné, conditions aux limites, coefficients, etc doivent être des fonctions vectorisées, par exemple

  ```
  f =@(x,y,z) cos(x.*y)./z;
  ```

- Le fichier du maillage pour les éléments finis d'ordre 2 (P2 ou P2H) n'est pas le même que celui en P1. Il faut utiliser la fonctionnalité: set order 2 de Mesh. Ceci construit une transformation géométrique isoparamétrique d'ordre 2 qui permet de mieux tenir compte de la courbure de la géométrie et ne modifie pas les géométries droites.
- Présentement, seulement le cas isoparamétrique est implanté. On ne peut utiliser une transformation géométrique de degré inférieur que celui de élément fini (P1-P1) ou (P2-P2).
- Il faut faire attention au choix du schéma d'intégration autant pour le domaine que ceux des frontières. En gros, une entité frontière est traitée comme un maillage de dimension inférieure (arête = élément 1D, face = élément 2D). La table de connectivité est fournie par Gmsh.

Voici un exemple du fichier probDiffusion.m

```
% script Matlab pour resoudre probleme EF
% maillage fourni par Gmsh


clear all;
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% information sur les parametres physiques
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Param.k=@(x,y,z) 1; % constante de diffusion
%Param.f = @(x,y,z) (x.^2 + y.^2).*cos(x.*y);  % terme source
Param.f = @(x,y,z) 0; % terme source
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Informations sur le probleme d'elements finis
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% parametres
nomFichier = 'Maillages2D/cercleP2';
%nomFichier = 'Maillages3D/cube';

CondLim.nel =  688;    % nombre d'elements du maillage P1 (carre)
%CondLim.nel = 240 ;    % nombre d'elements du maillage P2 (cubeP2)


% type l'element fini
TypeEF.triangle = true; % false pour quad
TypeEF.tetra = false; % false pour hex
TypeEF.type ='P2H';  % P2H ou P1 ou P2 ou Q1 ou Q2H ou Q2
TypeEF.typeGeo ='P2';  %  P1 ou P2 ou Q1 ou Q2
TypeEF.ordre = 2; % ordre de l'element fini
TypeEF.degGeo = 2; % degre de la transformation geometrique (isoparametrique)
TypeEF.nd = 2;   % dimension = 2 ou 3
TypeEF.ndim = 1;   % nombre de composantes par noeud
TypeEF.ordreIntg = 4;   % ordre du schema d'integration (2,3,4 pour tri) (2,3,5 pour tet)
TypeEF.ordreIntgFace = 5;   % ordre du schema d'integration (2*nptg - 1)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Conditions aux limites du probleme
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% conditions aux limites de Dirichlet
 listeDir = [8:9]; % indices des frontieres de Dirichlet fournies par Gmsh
 n = length(listeDir);
 CondLim.listeFrontiereDir = listeDir;
 CondLim.listeNoeudFrontiereDir = cell(n,1);
 CondLim.connectFrontiereDir = cell(n,1);
 CondLim.fonctFrontiereDir = cell(n,1);

 % conditions aux limites de Neumann
 listeNeumann = []; % indices des frontieres de Neumann fournies par Gmsh
 nn = length(listeNeumann);
 CondLim.listeFrontiereNeumann = listeNeumann;
 CondLim.connectFrontiereNeumann = cell(nn,1);
 CondLim.fonctFrontiereNeumann = cell(nn,1);

 % conditions aux limites de Robin
 listeRobin = []; % indices des frontieres de Robin fournies par Gmsh
 nnn = length(listeRobin);
 CondLim.listeFrontiereRobin = listeRobin;
 CondLim.connectFrontiereRobin = cell(nnn,1);
 CondLim.fonctFrontiereRobin = cell(nnn,2);

 % script pour les conditions aux limites

switch TypeEF.nd

    case 2 % probleme 2D

prob = 1;
probDiffusion_CondLimites;

    case 3 % probleme 3D
prob = 1;
probDiffusion3D_CondLimites;
end
```

```
% resolution

[Mail, CondLim, FctBases]= problemeEF(nomFichier, TypeEF, CondLim, Param);
%[Mail, CondLim, FctBases]= problemeSteklovVP(nomFichier, TypeEF, CondLim, Param);
 %[Mail, CondLim, FctBases]= probleme2DValeursPropres(nomFichier, TypeEF, CondLim, Param);
```

Description du fichier probElasticite

Le fichier est identique au fichier probDiffusion sauf pour la description des paramètres physiques de l'équation -div $\sigma(\varepsilon(\boldsymbol{u})) = \boldsymbol{f}$. L'information se trouve dans la structure **Param**:

- Param.eta est la constante de Lamé $\eta = E/(2(1+\mu))$,
- Param.lambda est la constante de Lamé $\lambda = 2\eta\mu/(1-2\mu)$,
- La force volumique est donnée par l'expression Param.f qui est un cell à 3 composantes

```
Param.f{1} = @(x,y,z) 0; % terme source composante x
Param.f{2} = @(x,y,z) 0; % terme source composante y
Param.f{3} = @(x,y,z) 0; % terme source composante z
```

De plus, il y a les fichiers *probElasticiteCondLimites.m* (cas 2D) et *probElasticite3DCondLimites.m* qui contiennent une instruction switch qui permet de choisir un ensemble de conditions aux limites via la variable prob=1,2, etc

Description du code

Tout le code se trouve dans la fonction problemeEF. En gros les étapes du code sont

- Lecture du maillage et lecture des conditions aux limites proveant du fichier de maillage

```
ndim = TypeEF.ndim; % nombre de composantes par noeud
nd = TypeEF.nd;   % dimension 2 ou 3
[Mail,CondLim] = lireMaillageGmsh(nomFichier,CondLim,TypeEF);
```

- Evaluation des fonctions de bases (et des dérivées) aux noeuds de Gauss

```
 switch nd
    case 2
FctBases = evalueFctBasesElemRef2D(TypeEF);
    case 3
FctBases = evalueFctBasesElemRef3D(TypeEF);
end
```

- Initialisation de la solution aux frontières de type Dirichlet

```
 switch ndim
    case 1
        u = initSolutionEntite(Mail,FctBases, CondLim,nnoeud);
    case 2
        u = initSolutionVect2DEntite(Mail, FctBases, CondLim, nnoeud);
    case 3
        u = initSolutionVect3DEntite(Mail, FctBases, CondLim, nnoeud);
end
```

- Assemblage du système linéaire $AU = B$ sans les conditions aux limites

```
[A, B] = assembSysteme(Mail, FctBases, CondLim, u, param);
```

- Prise en compte des conditions aux limites de Neumann (de même pour Robin)

```
 for i=1:length(CondLim.listeFrontiereNeumann)

switch ndim
    case 1
 fonctNeumann=CondLim.fonctFrontiereNeumann{i};
    case 2
fonctNeumann = cell(1,2);
fonctNeumann{1}=CondLim.fonctFrontiereNeumann{i,1};
fonctNeumann{2}=CondLim.fonctFrontiereNeumann{i,2};
    case 3
fonctNeumann = cell(1,3);
fonctNeumann{1}=CondLim.fonctFrontiereNeumann{i,1};
fonctNeumann{2}=CondLim.fonctFrontiereNeumann{i,2};
fonctNeumann{3}=CondLim.fonctFrontiereNeumann{i,3};
end
BNeumann = appliqueCondLimNeumann(Mail, FctBases, CondLim.connectFrontiereNeumann{i},  ndim,
    fonctNeumann );
B = B + BNeumann;

end
```

- On réduit la taille du système linéaire afin de tenir compte des valeurs connues (conditions de Dirichlet)

```
 NoeudsLibre = 1:nnoeud;
 I=CondLim.listeNoeudDir;  % pour le P1 et P2

 switch ndim
```

```
    case 2
      I = [2*I-1 , 2*I];
    case 3
      I = [3*I-2 , 3*I-1, 3*I];
end

 NoeudsLibre(I) = [];
 B(I) = [];
 A=A(NoeudsLibre,NoeudsLibre);
```

- On résoud le système linéaire en correction $A\delta U = B - AU$ et mise à jour de la solution $U = U + \delta U$

```
du = A\B;
u(NoeudsLibre) = u(NoeudsLibre) + du;
```

- En option, le code permet le calcul de l'erreur par rapport à une solution exacte connue qui est déclarée dans les fichiers solutionExacte2D.m et solutionExacte3D pour les problèmes scalaire et dans solutionExacteVect2D.m et solutionExacteVect3D.m pour les problèmes vectorielles.

```
switch ndim
    case 1
        switch nd
    case 2
    [ normeH1, normeL2 ] = evalueNormeSolution2D( Mail,  FctBases, u );
    case 3
    [ normeH1, normeL2 ] = evalueNormeSolution3D( Mail,  FctBases, u );
        end

    case 2
    [ normeH1, normeL2 ] = evalueNormeSolutionVect2D( Mail,  FctBases, u );
    case 3
    [ normeH1, normeL2 ] = evalueNormeSolutionVect3D( Mail,  FctBases, u );
end
```

- Stockage de la solution et visualisation à l'aide de Gmsh.

```
ExportGmsh(Mail,u, nomFichier, TypeEF);
```