Université Paris-Dauphine
Département MIDO

M2, EDP-MAD.

T.P. 5 : Bases réduites

Dans ce T.P., on implémente l'exemple donné dans le cours sur la température dans une plaque hétérogène en régime permanent.

1 Code "Offline" : calcul par éléments finis

On rappelle l'exemple. On considère un domaine $\Omega = [0,1]^2$, et le problème variationnel consistant à trouver u tel que

$$a(u,v;\mu) = f(v),\ \forall v \in H^1(\Omega),\ v_{|[0,1]\times\{1\}} = 0$$

avec

- $a(u,v;\mu) = \int_\Omega \kappa(\mu)\nabla u \nabla v$,
- $f(v) = \int_0^1 v(x,0)dx$,
- la fonction $\kappa(\mu)$ est constante par morceaux. Dans ce T.P., on décompose Ω en 9 parties Ω_i identiques, numérotées de $i=1$ à $i=9$ (par exemple $\Omega_1 = [0,1/3]^2$, $\Omega_2 = [0,1/3]\times[1/3,2/3]$,etc). On note μ_i les valeurs correspondantes de k sur ces domaines.

On rappelle que cette formulation variationnelle correspond à des conditions aux bords de type Neuman homogène sur $\{0\}\times[0,1]$ et $\{1\}\times[0,1]$, une condition de Neuman non-homogène $\partial_n v = 1$ sur $[0,1]\times\{0\}$ et Dirichlet homogène sur $[0,1]\times\{1\}$.

Coder en FreeFem++ cet exemple. On commencera par la ligne

```
include "mu.idp"
```

où le fichier "mu.idp" contient les valeurs des paramètres μ_i

```
real mu1=1,mu2=2,mu3=3,mu4=4,mu5=5,mu6=6,mu7=7,mu8=8,mu9=9;
```

2 De FreeFem++ vers Octave et vis-et-versa

Pour pouvoir faire facilement un code Base réduite, on peut utiliser Octave. Un interfacage entre FreeFem et Octave s'impose donc pour transférer les matrices et vecteurs de l'un vers l'autre.

2.1 Transfert de fichiers de FreeFem vers Octave

Le logiciel FreeFem++ résoud en fait un système du type

$$A(\mu)U(\mu) = F.$$

En fin de code FreeFem++, ajouter des lignes permettant d'écrire des fichiers "FFA.m", "FFU.m" et "FFF.m" où sont stockées (sous forme "sparse" pour ce qui concerne A) les matrices et vecteurs A, U et F obtenus avec μ. Pour avancer plus vite, voici les lignes pour obtenir "FFA.m" :

```
varf  a(u,v)=int2d(th)(kappa*(dx(u)*dx(v)+dy(u)*dy(v)));
matrix mass=a(vh,vh) ;
int nCh=vh.ndof ;
int[int] I(1),J(1);
real[int] A(1);
[I,J,A]=mass;
ofstream gnu("FFA.m");
gnuA.precision(20);
gnuA << "A=sparse(" << nCh << "," << nCh << ");" <<  endl;
for (int k=0;k<I.n;k++)
{ gnuA << "A(" << I[k]+1 << "," << J[k]+1 << ")=" << A[k] << ";" << endl;}
```

et "FFF.m" :

```
varf l(unused,v) = int1d(th,Aa)(v) + on(3,unused = 0);
vh F;
F[] = l(0,vh);
ofstream gnuF("FFF.m");
gnuF.precision(20);
gnuF << "F=zeros(" << nCh << ",1);" <<  endl;
for (int k=0;k<nCh;k++)
{ gnuF << "F(" << k << ")=" << F[][k] << ";" << endl;};
```

À l'aide de ces lignes, on produit un fichier lisible par Octave.

2.2 Calculs en FreeFemm depuis Octave

Lors du calcul des "snapshots", on aura besoin soumettre un vecteur de paramètres μ à FreeFem, puis de lancer FreeFem depuis Octave.

1. Faire une sous-fonction "writemu.m" qui, étant donné μ, sauve sur le disque un fichier "mu.idp". Pour avancer plus vite, voici les lignes permettant d'écrire le fichier "mu.idp"

```
Mu=sprintf('real mu1=%i,mu2=%i,mu3=%i,mu4=%i,mu5=%i,mu6=%i,mu7=%i,mu8=%i,mu9=%i;',mu);
fid=fopen("mu.idp",'w');
fprintf(fid, Mu);
fclose(fid);
```

2. On souhaite ensuite lancer des calculs depuis Octave. La commande est simplement :

```
system('FreeFem++ TPBR.edp')
```

3 Code "Online" : méthode des bases réduites

Nous sommes maintenant prêts pour écrire un code Octave de calcul par base réduite. On considère deux ensembles de points P_{train} et P_{test} discrétisant l'espace des paramètres $P = \{\mu\} = [0,1]^9$. La première va être utilisée pour construire la base par un algorithme glouton ("greedy"), et la deuxième, composée de points tirés aléatoirement, sera utilisée pour tester la méthode. Voici les étapes, avec un test à la fin :

1. Justifier pourquoi le problème est affine-décomposable. Extraire les matrices $A_1, \cdots, A_9$ permettant cette décomposition.
2. Extraire également la matrice du produit scalaire $H_0^1(\Omega)$.
3. Coder le calcul de l'estimateur d'erreur a posteriori vu en cours $\Delta_u = \frac{1}{\alpha(\mu)}\|r(\mu)\|_{H_0^1(\Omega)}$. On le codera de manière à ce que le calcul soit décomposé offline-online et on vérifira qu'il s'annule en cas d'erreur nulle.
4. Coder l'algorithme glouton basé sur l'estimateur a posteriori. On choisira un ensemble P_{train} random petit pour les tests de debuggage.
5. Tester la base construite sur l'ensemble de paramètres P_{test}. On augmentera la taille de la base incrémentalement pour observer la vitesse de convergence, et on l'orthonormalisera au fur-et-à-mesure de sa construction.