Assemblage de la matrice d'élément fini

Ce TP est à effectuer en python.

Lecture d'un fichier Gmsh

Le fichier GmshRead.py contient une classe mesh dont les objets sont les données associées à un maillage triangulaireP1 en 2D contruit à partir d'un fichier gmsh (v4). Un élément de la classe maillage contient: * `Nnodes` : nombre de noeuds du maillage * `Nodes` : tableau de taille (Nnodes, 2) contenant les coordonnées des noeuds du maillage * `label` : tableau de taille Nnodes contenant des informations sur la location du noeud (noeud du bords / noeuds interne) * `Nel` : nombre d'éléments du mailage * `connect` : tableau de connectivité de taille `(Nel, 3 )`

Pour les remplir, il parcoure un fichier .msh ligne à ligne (`line = f.readline()`) et récupère les données en parsant chaque ligne (`line.split()`).

1. Remplir le tableau `connect`.
2. Ajouter un tableau diam de taille `Nel` qui contient le diamètre de chacun des éléments, ainsi que le paramètre h du maillage.
3. Ajouter un tableau area de taille `Nel` qui contient l'aire de chacun des éléments. [Indication: l'aire d'un parallélogramme est égale au produit vectoriel/ déterminant des deux vecteurs qui l'engendrent]
4. Tester votre code avec le maillage d'un rectangle triangulé régulièrement.
5. Vérifier que l'aire du rectangle est bien la somme des aires des triangles.
6. Vérifier la même chose sur une géométrie différente du rectangle.

Base de l'élément fini P1.

Considérons le triangle de sommets $a_1 = (0,0), a_2 = (1,0)$ et $a_3 = (0,1),$ comme éléments de références. La base d'élément fini associé à chacun de ces points est donnée par:

$$\psi_1(x,y) = 1 - (x+y), \quad \psi_2(x,y) = x, \quad \psi_3(x,y) = y$$

1. Créer une fonction `coord(ld)` qui prend en entrée un tableau de taille 3 contenant les coordonnées barycentriques $\left(\lambda_1, \lambda_2, \lambda_3\right)$ d'un point et renvoie le tableau de taille 2 contenant les coordonnées de ce point:

$$(x,y) = \lambda_1 a_1 + \lambda_2 a_2 + \lambda_3 a_3$$

1. Nous souhaitons créer une fonction `base_psiref()`` qui permettra d'effectuer les quadratures sur l'élément de référence. Cette fonction doit renvoyer les quatre tableaux suivants:
   - `pts`: tableau de taille (7, 3) contenant les coordonnées barycentriques des points de quadrature,
   - `wght`: tableau de taille 7 contenant les poids de quadrature associé,
   - `psi`: tableau de taille (3,7) contenant la valeur des 3 fonctions de bases aux 7 points de quadratures.
   - `derpsi`: tableau de taille (3,7,2) contenant la valeur des gradients des 3 fonctions de bases aux 7 points de quadratures.

Assemblage de la matrice.

1. Créer une `class poisson` qui contient un prend à la contruction en entrée un maillage `Mh` et une fonction, f, correspondant au second membre de l'équation. Un élément de la classe contiendra de plus `Ndof`, le nombre de degré de liberté (correspondant au nombre de noeuds du maillage), une matrice de taille `(Ndof,Ndof)`, stockée intialement au format dok, un tableau `rhs` de taille `Ndof`, contenant le second membre du système linéaire et enfin un tableau u de taille `Ndof` contenant la solution approchée.
2. Ajouter une fonction assemble_matrix(self) qui assemble la matrice. On pourra compléter le code suivant :

```
pts, wght, psi, derpsi = basis_psiref()
for nel in range( Mh.Nel):
    A = ... # noeuds du triangle
    B = ....
    C = ....
    comT = ... #det(T_K) * (nabla (T_K))^{-1} = com(T_{K})^{-1}
    detT = .... #det(T_K)
    for i, wgh in enumerate(wght):
        for ni in range( 3):
            inode = Mh.connect[nel, ni]
            for nj in range(3):
                jnode = Mh.connect[nel, nj]
                dpsi_i = ...
                dpsi_j = ...
                self.M[inode, jnode] += ....
                self.M = self.M.tocsc()
```

1. La matrice précédente est la matrice associée au Laplacien avec condition de Neumann. Pour inclure des conditions de Dirichlet, modifier la matrice de telle sorte que `M[inode,inode] = tgv`, avec `tgv = 1.e6` dès que `inode` correspond à un noeud du bord.
2. Créer sur le même modèle une fonction `rhs( self)` qui assemble le second membre. Pour inclure des conditions de Dirichlet, il faut aussi que `rhs[inode] = 0` dès que `inode` correspond à un noeud du bord.
3. Créer une fonction `solve( self)` qui calcule la solution approchée.
4. Ajouter une fonction `plot_sol( self)` qui permet d'afficher la solution approchée et ajouter un argument plot dans la fonction solve.

```
x = Mh.Nodes[:, 0]
y = Mh.Nodes[:, 1]
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_trisurf(x, y, self.ufull, linewidth=0.2,
antialiased=True, cmap=plt.cm.CMRmap)
fig.colorbar(surf, shrink=0.5, aspect=5)
```

1. Tester votre fonction avec un maillage du disque et la fonction `f = 1`. Calculer la solution exacte et son gradient.
2. Ajouter une fonction `compute( self)` qui calcule l'erreur en norme L^2 et en norme H^1 entre la solution approchée et la solution exacte (projetée sur l'espace d'élément finis P1). Vérifier l'ordre de convergence numérique.

Last updated 2020-03-24 17:33:49 +0100