

Assemblage de la matrice d'éléments  
fini

Ce TP est à effectuer en python, l'annonce est [ici](#).

# Lecture d'un fichier Gmsh

Le fichier GmshRead.py contient une classe mesh dont les objets sont les données associées à un maillage triangulaire  $\mathbb{P}^1$  en 2  $\mathbb{D}$  construit 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 maillage
- **connect**: tableau de connectivité de taille (Nel, 3)

Pour les remplir, il parcourt un fichier .msh ligne à ligne (`line = f.readline()`) et récupère les données en parseant 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.

## TIP

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 grammaire 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 associée à chacun de ces points est donnée par:

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

! `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 `__init__` la construction en entrée un maillage `Mh` et une fonction, `f`, correspondant au second membre de l'équation. Un `__init__` de la classe contiendra de plus `Ndof`, le nombre de degrés de liberté (correspondant au nombre de noeuds du maillage), une matrice de taille (`Ndof`, `Ndof`), stockée initialement 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 :

```
def __init__(self, pts, wght, psi, derpsi):
    self.pts = pts
    self.wght = wght
    self.psi = psi
    self.derpsi = derpsi
    self.Mh = Mh
    self.Ndof = self.Mh.nnodes
    self.M = dok_matrix((self.Ndof, self.Ndof))
    self.f = f
    self.u = zeros(self.Ndof)

    # Calcul des matrices de base
    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 = self.Mh.connect[ni]
            for nj in range(3):
                jnode = self.Mh.connect[nj]
                dpsi_i = ...
                dpsi_j = ...
                self.M[inode, jnode] += ...
            self.M = self.M.tocsc()
```

3. 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` dès que `inode` correspond à un noeud du bord.
4. 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.
5. Créer une fonction `solve(self)` qui calcule la solution approchée.
6. 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.uffull, linewidth=0.2,
    antialiased=True, cmap=plt.cm.CMRmap)
fig.colorbar(surf, shrink=0.5, aspect=5)

```

7. Tester votre fonction avec un maillage du disque et la fonction  $f = 1$ . Calculer la solution exacte et son gradient.
8. 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'éléments finis  $P_1$ ). Vérifier l'ordre de convergence numérique.
9. Implémenter le support de condition de Neumann non homogène
10. Tester votre code avec conditions de Dirichlet et Neumann homogènes et non homogènes sur une géométrie non triviale de votre choix avec une solution manufacturée dans l'espace et une non-polynomiale en faisant une étude de convergence.

## Comparaison avec Feel++

1. Implémenter les cas précédents (Q10) avec Feel++ en python. Tester et vérifier les résultats des cas précédents, comparer le comportement des normes  $L^2$  et  $H^1$ .
2. Implémenter les cas  $P^2$  et  $P^3$ , Tester et vérifier les résultats des cas précédents, qu'observe-t-on sur l'ordre de convergence en normes  $L^2$  et  $H^1$ . Est-ce le résultat attendu ? A quoi faut-il faire attention ?

## Implementation le cas parabolique avec Feel++

Soit  $\Omega = [0,1]^2$ , rajouter le terme de dérivée en temps  $\frac{\partial u}{\partial t}$ , implémenter un schéma Euler implicite en temps, linéaire et quadratique par morceaux en espace, des conditions de Dirichlet et un second membre donné par les fonctions ci-dessous de telle façon que ces fonctions soient solutions du problème et tester l'erreur  $L^2$  au dernier pas de temps avec les fonctions

1.  $t+x$  sur l'intervalle de temps  $[0,1]$  avec  $\Delta t=0.1$ , Qu'observez-vous concernant l'erreur ?
2.  $\sin(\pi x)\cos(\pi y)\exp(-t)$  sur l'intervalle de temps  $[0,1]$  avec  $\Delta t=0.1$
3.  $\sin(\pi x)\cos(\pi y)\exp(-t)$  sur l'intervalle de temps  $[0,1]$  avec  $\Delta t=0.05$
4. Comparer l'erreur sur les 2 derniers cas.