Journée Utilisateur LMGC90v2

TP 1 module poroMAILx - compression confinée

* Dominique Ambard

* Laboratoire de Mécanique et Génie Civil, UMR CNRS 5508, Université Montpellier II.
dominique.ambard@univ-montp2.fr

24 mai 2012

Plan

Modèle de Biot - Poroélasticité linéaire saturé

- Définition des variables d'état : σ, ε, p, ϕ.
$$dG = \sigma d\varepsilon + p d\phi.$$
- Relation de comportement couplé :
$$\sigma = \overline{\overline{K}}\varepsilon - \overline{b}p.$$
$$\phi = \overline{b}\nabla\vec{v}_s + \frac{1}{M}p.$$
- Relation de comportement en filtration (Loi de Darcy) :
$$\vec{q}_{f/s} = -\frac{\overline{k}}{\mu}\vec{\nabla}p.$$

Dans Ω :

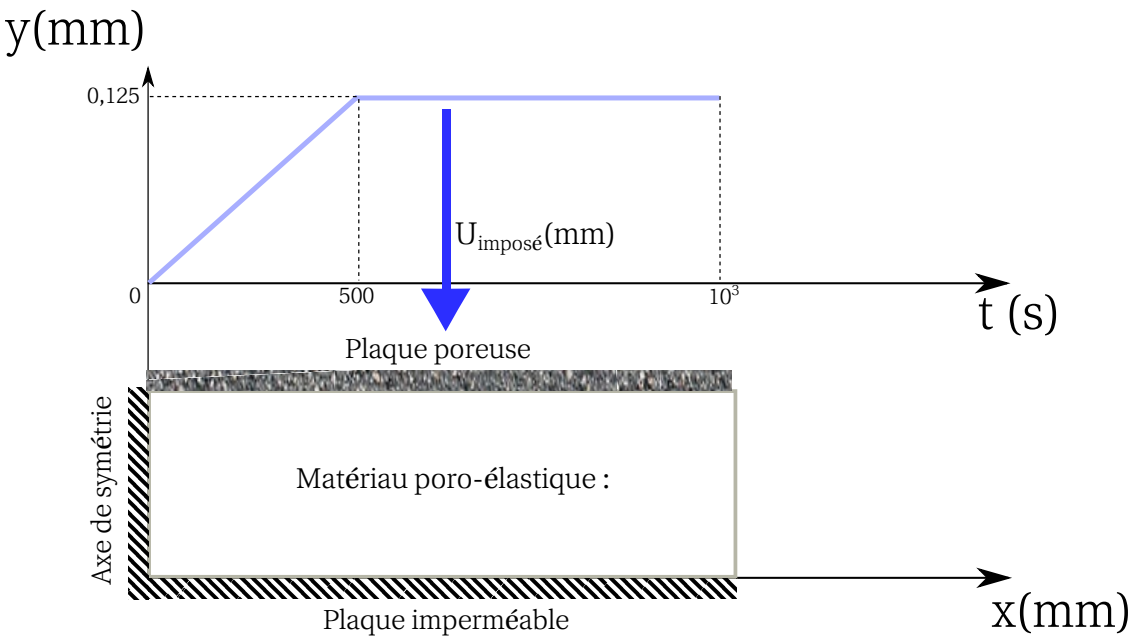
$\vec{\nabla}\vec{\sigma}_s - b\vec{\nabla}p + \tilde{\rho}\vec{g} = \vec{0}$	Avec la loi de comportement : $\sigma_s = \overline{\overline{K}}\varepsilon$
$\frac{1}{M}\frac{dp}{dt} + b\nabla\vec{v}_s + = -\nabla\vec{q}_{f/s}$	Avec la loi de comportement : $\vec{q_{f/s}} = -\frac{\overline{k}}{\mu}\vec{\nabla}p$

Sur les frontières :

Sur $\delta_1\Omega$: $\vec{f}_d = \sigma\vec{n}$	Sur $\delta_3\Omega$: $\vec{V}_d = \vec{v}_s$
Sur $\delta_2\Omega$: $\vec{q}_d = \vec{q}_{f/s}$	Sur $\delta_4\Omega$: $p_d = p$

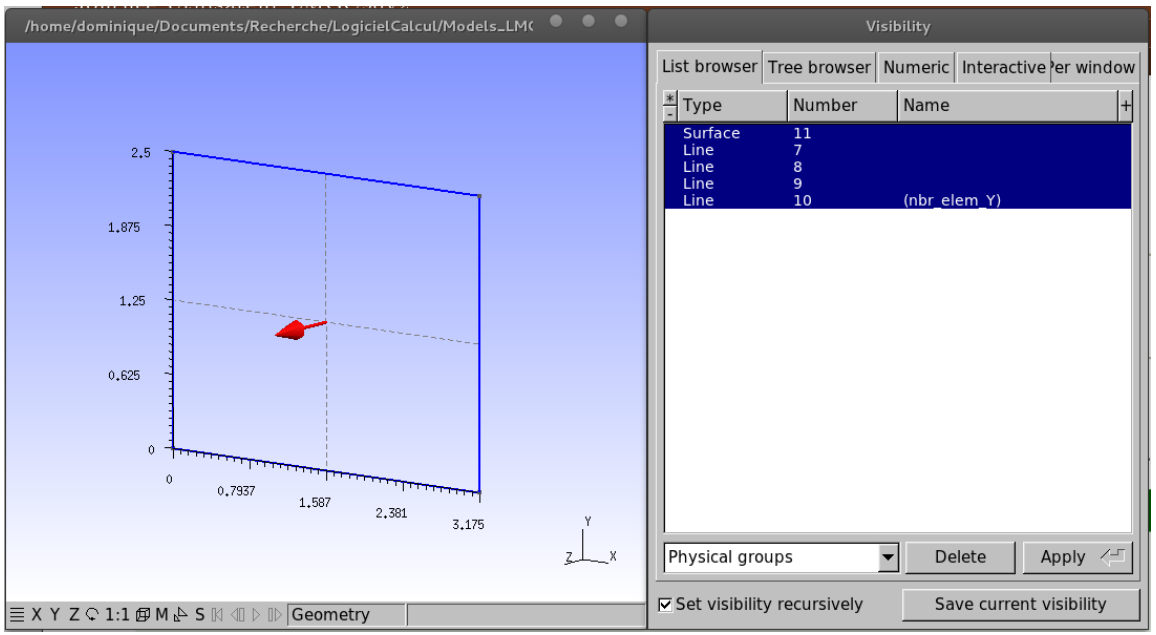
Problématique :

- Echantillon de matériaux poro-élastique de diamètre : $d = 6.35mm$ et $h = 2.5mm$.
- Posé sur un support étanche.
- Ecrasé par une machine d'essai à déplacement controlé à l'aide d'un plateau étanche.
- Mesure de l'effort à l'aide d'un capteur de force.



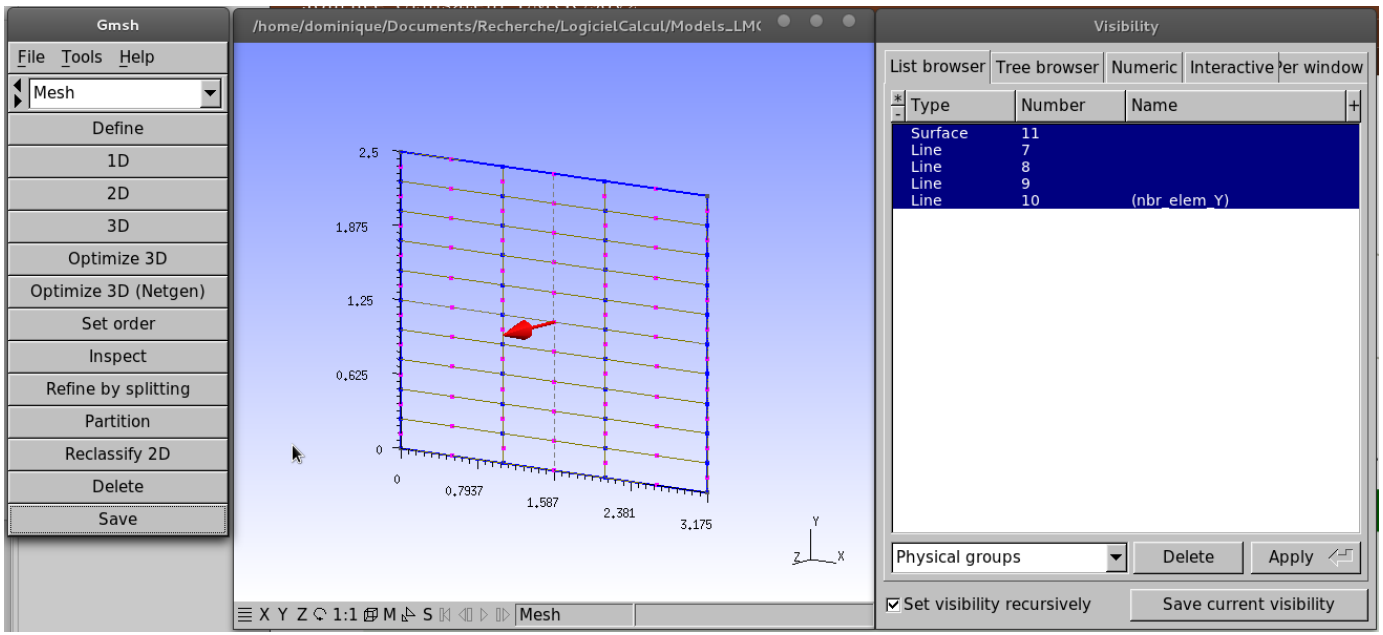
Génération du maillage avec gmsh :

- Fichier d'entré `Mesh.geo`.
- Vérification des normales.
- Maille d'ordre 2.
- Connaissance des groupes physiques.



Génération du maillage avec gmsh :

- Fichier d'entré `Mesh.geo`.
- Vérification des normales.
- Maille d'ordre 2.
- Connaissance des groupes physiques.



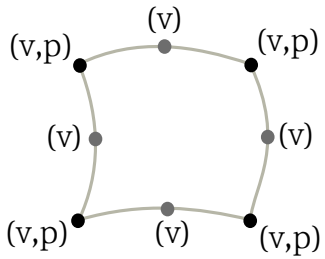
Création du script :

- Fichier d'entré **Pre-PoroElastic.py**

Importation du **Pre** et définition de conteneur

```
1  ##################################################################################
2  # Importation du module pre_lmgc
3  from pre_lmgc import *
4
5  # definition des conteneurs:
6  #   * de corps
7  bodies = avatars()
8  #   * de modeles
9  mods = models()
10 #   * de materiaux
11 mats = materials()
12 #   * pour les tables de visibilite
13 svs = see_tables()
14 #   * pour les lois de contact
15 tacts = tact_behavs()
```

- Variable nodale v - vitesse de la matrice solide $\vec{v}_s$.
- Variable nodale p - pression intersticielle p.
- Elément de Taylor-Hood.

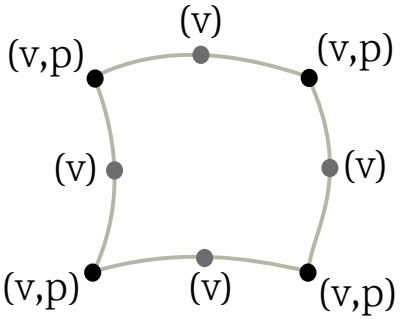


Q84xx

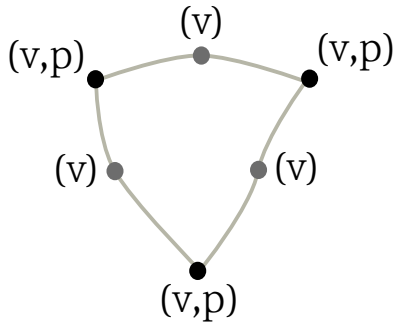
Création de la maille poro-élastique Quad8

```
1  # on se place en 2D
2  dim = --
3  # definition d'un modele diffusif couple en terme source
4  Porous_model = model(nom='-----', type='-----', element='-----',\
5                       dimension = ---, \
6                       external_model='-----', kinematic='-----',\
7                       material='-----',anisotropy='-----',\
8                       mass_storage='-----',\
9                       physical_type = '-----',\
10                      capacity_storage='-----',\
11                      convection_type = '-----')
12
13 # on ajoute le modele dans le conteneur
14 mods.addModel(Porous_model)
```

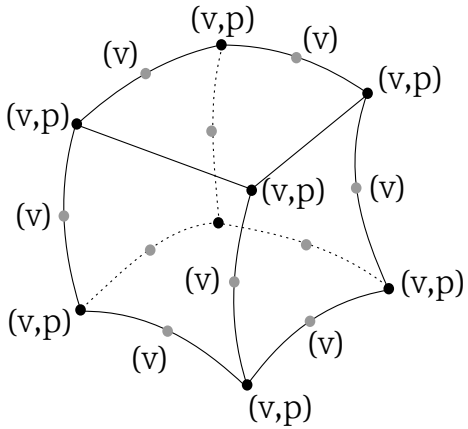
Elements finis poro élastique LMGC90v2



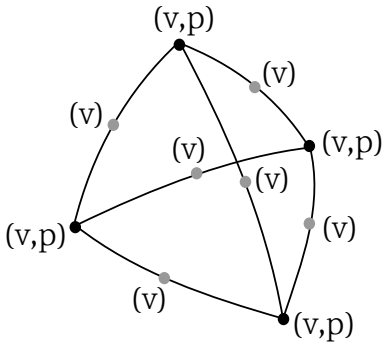
Q84xx



T63xx



H208x



T104x

Briques poreuses `solid` :

$$\nabla\vec{\sigma}_s - b\nabla\vec{p} + \tilde{\rho}\vec{g} = \vec{0}$$
$$\frac{1}{M}\frac{dp}{dt} + b\nabla\vec{v}_s = \nabla\vec{q}_{f/s}$$

Briques fluides `fluid` :

$$\nabla\vec{\sigma}_f - \nabla\vec{p} + \tilde{\rho}\vec{g} = \vec{0}$$
$$\nabla\vec{v}_f = 0$$

Modèle discret dans Ω :

Gestion de la partie solide - `physical type = 'solid'` :

$$\begin{bmatrix} M & 0 \\ 0 & C \end{bmatrix} \frac{d}{dt}\begin{bmatrix} V \\ P \end{bmatrix} + \begin{bmatrix} 0 & -B \\ B^T & 0 \end{bmatrix}\begin{bmatrix} V \\ P \end{bmatrix} + \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} U \\ \int P dt \end{bmatrix} = \begin{bmatrix} F \\ Q \end{bmatrix} \quad (1)$$

Gestion de la partie fluide - `physical type = 'fluid'` :

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \frac{d}{dt}\begin{bmatrix} V \\ P \end{bmatrix} + \begin{bmatrix} K & -B \\ B^T & 0 \end{bmatrix}\begin{bmatrix} V \\ P \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} U \\ \int P dt \end{bmatrix} = \begin{bmatrix} F \\ Q \end{bmatrix} \quad (2)$$

- Module de Young $E = 0.4467 Mpa$.
- Coefficient de Poisson $\nu = 0.4467$.
- Perméabilité $\frac{k}{\mu} = 0.0075 N.s/m^4$.
- Coefficient de Biot $b = 1.0$.
- Coefficient de Compressibilite $\frac{1}{M} = 0.0$.

Création du matériau poro-élastique

```
1  # on definit le materiau constitutif du probleme
2  Biot = material(nom='-----', type='PORO_ELAS', density=---, \
3                  specific_capacity=---, conductivity=---, \
4                  elas = 'standard', young = ---, nu = ---, \
5                  anisotropy = 'isotropic',hydro_cpl = ---)
6
7  # on l'ajoute dans le conteneur
8  mats.addMaterial(Biot)
```

- Lecture du maillage gmsh **Mesh.msh**.
- Affectation des noeuds et des mailles au corps maillé.
- Affectation du modèle et du matériaux au mailles au groupe physique.

Création du corps maillé et affectation du modèle et du matériau

```
1  # definition maillage en Qua8 :
2  # on lit le mailage pour recuperer la liste des noeuds et des elements
3  mail1 = lecture('Mesh.msh', dim)
4
5  # Gection des molecules 1
6  # il s'agit d'un corps maille
7  TP1 = avatar(number=---, type='-----', dimension=----)
8  # on ajoute les noeuds au corps maille
9  TP1.addBulks(-----.bulks)
10 # on ajoute les elements au corps maille
11 TP1.addNodes(-----.nodes)
12 # on definit les groupes physiques pour le corps maille
13 TP1.defineGroups()
14 # on affecte son modele au corps maille sur un groupe physique
15 TP1.defineModel(model=------, group = '--')
16 # on affecte son materiau au corps maille sur un groupe physique
17 TP1.defineMaterial(material=------, group = '--')
18
19 # ajout du cube dans le conteneur de corps
20 bodies += TP1
```

- Affectation des conditions aux limites sur les groupes physiques `group = '7'`.
- Application sur chaques DDL `component = 1`.
- Choix du type de conditions : vitesse `dofty = 'vlocy'` ou force `dofty = 'force'`.

Création du corps maillé et affectation du modèle et du matériau

```
1  # application des conditions aux limites sur
2  #              les groupes physiques du maillage GMSH:
3  # [CT.....+.....AMP..*..cos.(..OMEGA.*.time.+.PHI..)]...
4  #    *
5  # [RAMPI.....+.....RAMP.*.time]
6  TP1.imposeDrivenDof(group='---', component=--  , dofty='-----')
7  TP1.imposeDrivenDof(group='---', component=--  , dofty='-----')
8  TP1.imposeDrivenDof(group='---', component=--  , dofty='-----')
9  TP1.imposeDrivenDof(group='---', component=--  , dofty='-----',\
10                     type = 'evolution', evolutionFile = 'Vimp_t.txt')
11 TP1.imposeDrivenDof(group='---', component=--  , dofty='-----',\
12                     ct = 0.0, rampi = 0.0, amp = 1.0, omega = 0.0,\
13                     phi = 0.0, ramp = 0.0)
14
15 # application des conditions initiales :
16 TP1.imposeInitValue(group='---',component=--  ,value=0.0)
17 TP1.imposeInitValue(group='---',component=--  ,value=0.0)
18 TP1.imposeInitValue(group='---',component=--  ,value=0.0)
```

- Ecriture de la base de donnée.
- Application de la gravité.

Création du corps maillé et affectation du modèle et du matériau

```
1  # Ecriture des fichiers pour LMGC
2  writeBodies(bodies, chemin='./DATBOX/')
3  writeDofIni(bodies, chemin='./DATBOX/')
4  writeDrvDof(bodies, chemin='./DATBOX/')
5  writeModels(mods,chemin='./DATBOX/')
6  writeGPVIni(bodies, chemin='./DATBOX/')
7  writeBulkBehav(mats, chemin='./DATBOX/', dim=dim, gravy=[0., 0., 0.])
8  writeTactBehav(tacts,svs,chemin='DATBOX/')
9  writeVlocRlocIni(chemin='DATBOX/')
```

- Paramétrage de la discrétisation temporelle.
- Définition de la dimension 2D axi.

Paramètres de la discrétisation temporelle et de la dimension du problème

```
1  # Time discretization:
2  dt = 10.0
3  theta = 1.0
4
5  chipy.TimeEvolution_SetTimeStep(dt)
6  chipy.NewtonRaphson_SetFinalTime(dt)
7  chipy.NewtonRaphson_SetMinTimeStep(dt)
8  chipy.NewtonRaphson_SetMaxTimeStep(dt)
9  chipy.NewtonRaphson_SetMaxIter(20)
10 chipy.NewtonRaphson_SetIncPatience(999999)
11
12 # Initialize theta integrator pour toutes les physisques
13 chipy.Integrator_InitTheta(theta)
14
15 # Declaration d'un probleme 2D(2) AXI(3),PLANE STRAIN(1),PLANE STRESS(2)
16 # Declaration d'un probleme 3D(3)
17 chipy.overall_DIME(2,3)
18 # Verification des repertoires necessaires
19 chipy.checkDirectories()
```

- Chargement de la base de donnée.

Paramètres de la discrétisation temporelle et de la dimension du problème

```
1  # READ and WRITE BODIES
2  chipy.MAILx_ReadBodies()
3  chipy.overall_WriteBodies()
4  chipy.MAILx_WriteBodies()
5
6  # READ and WRITE MODELS
7  chipy.models_ReadModels()
8  chipy.models_WriteModels()
9
10 # READ and WRITE BEHAVIOURS
11 chipy.bulk_behav_ReadBehaviours()
12 chipy.bulk_behav_WriteBehaviours()
13
14 # INIT MODELS
15 chipy.models_InitModels()
16 chipy.ExternalModels_InitModels()
```

- Chargement du modèle **poroMAILx**.
- Chargement des conditions initiales et aux limites.

Paramètres de la discrétisation temporelle et de la dimension du problème

```
1  # Chargement du modele poroMAILx
2  chipy.poroMAILx_LoadModels()
3  chipy.poroMAILx_LoadBehaviours()
4  chipy.poroMAILx_PushProperties()
5  chipy.poroMAILx_WithRenumbering()
6  chipy.models_StoreProperties()
7  chipy.ExternalModels_CheckProperties()
8
9  # READ and WRITE Initial condition
10 chipy.overall_ReadIniDof()
11 chipy.poroMAILx_ReadIniDof()
12
13 chipy.overall_ReadIniGPV()
14 chipy.poroMAILx_ReadIniGPV()
15
16 # READ and WRITE DRVDOF
17 chipy.poroMAILx_ReadDrivenDof()
18 chipy.overall_WriteDrivenDof()
19 chipy.poroMAILx_WriteDrivenDof()
```

- Boucle en temps de résolution.
- Boucle de convergence non-linéaire.

Début de la résolution en temps et initialisation de boucle de Newton

```
1  # Recuperation du nombre de noeuds du corps
2  nb_node = chipy.poroMAILx_GetNbNodes(1)
3  n = 0
4  while n < 100:
5          # Initialisation d'un nouveau pas de temps
6          chipy.TimeEvolution_IncrementStep()
7          chipy.poroMAILx_IncrementStep()
8          chipy.TimeEvolution_DisplayStep()
9
10         # Boucle de Newton
11         chipy.NewtonRaphson_Initialize(1.0e-10)
12         convergence = 1
13         iter = 0
14         while convergence ==1:
```

- Calcul des matrices élémentaires.
- Assemblage des matrices et efforts élémentaires.
- Résolution du système matriciel.

Résolution du problème poro-élastique

```
1  # Calcul des matrices elementaires
2  chipy.poroMAILx_ComputeFext()
3  chipy.poroMAILx_ComputeDamping()
4  chipy.poroMAILx_ComputeBulk()
5
6  # Assemblage du systeme
7  chipy.poroMAILx_AssembRHS()
8  chipy.poroMAILx_AssembKT()
9
10 # Resolution du systeme linearise
11 chipy.poroMAILx_ComputeFreeVelocity()
12 chipy.poroMAILx_ComputeDof()
13
14 # Verification de la convergence si NL
15 if iter > 1 :
16         norm = chipy.poroMAILx_ComputeResidueNorm()
17         convergence  = chipy.NewtonRaphson_CheckConvergence(norm)
18 iter += 1
```

Un élément fini poro-élastique avec une formulation V-P

Modèle continu dans Ω :

$\nabla\vec{\sigma}_s - b\nabla\vec{p} + \tilde{\rho}\vec{g} = \tilde{\rho}\vec{\gamma}_s$ Avec la loi de comportement : $\sigma_s = \overline{\overline{K}}\varepsilon$

$\frac{1}{M}\frac{dp}{dt} + b\nabla\vec{v}_s + = -\nabla\vec{q}_{f/s}$ Avec la loi de comportement : $\vec{q_{f/s}} = -\frac{\overline{k}}{\mu}\nabla\vec{p}$

Modèle discret dans Ω :

$$\begin{bmatrix} M & 0 \\ 0 & C \end{bmatrix} \frac{d}{dt}\begin{bmatrix} V \\ P \end{bmatrix} + \begin{bmatrix} 0 & -B \\ B^T & D \end{bmatrix}\begin{bmatrix} V \\ P \end{bmatrix} + \begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} U \\ \int P dt \end{bmatrix} = \begin{bmatrix} F \\ Q \end{bmatrix} \quad (3)$$

- $\begin{bmatrix} M & 0 \\ 0 & C \end{bmatrix} \mapsto$ `poroMAILxComputeMass()` - paramètres : ρ_s, $\frac{1}{M}$
- $\begin{bmatrix} 0 & -B \\ B^T & D \end{bmatrix} \mapsto$ `poroMAILxComputeDamping()` - paramètres : b, $\frac{k}{\mu}$
- $\begin{bmatrix} K & 0 \\ 0 & 0 \end{bmatrix} \mapsto$ `poroMAILxComputeBulk()` - paramètres : E, ν
- $\begin{bmatrix} F \\ Q \end{bmatrix} \mapsto$ `poroMAILxComputeFext()` - paramètres : $DRVDOF.DAT$

- Si convergence.
- Actualisation des résultats aux noeuds.
- Actualisation des résultats aux points de Gauss.
- Actualisation du temps.

Actualisation des solutions aux noeuds et aux points de Gauss

```
1  # Verification de la convergence
2  istate = NewtonRaphson_ComputeTimeStep()
3  if not istate == 1 :
4          # Actualisation aux noeuds et aux points de Gauss
5          chipy.poroMAILx_UpdateDof()
6          chipy.poroMAILx_UpdateBulk()
7          chipy.TimeEvolution_UpdateStep()
```

- Récupération de la solution aux noeuds.
- Ecriture des fichiers *.vtk.

Ecriture de la solution aux noeuds

```
1  # Recuperation des informations vectorielles
2  U = chipy.poroMAILx_GetBodyVector('X____',1,nb_node*2)
3  U1 = U.reshape((2,nb_node),order = 'F').T
4  F = chipy.poroMAILx_GetBodyVector('Fint_',1,nb_node*2)
5  F1 = F.reshape((2,nb_node),order = 'F').T
6  # Recuperation des informations scalaires
7  P = chipy.poroMAILx_GetBodyVector('P____',1,nb_node)
8  P1 = np.zeros((P.shape[0],1),float)
9  P1[:,0] = P
10
11 # Ecriture du fichier resultats
12 Solution = np.concatenate((np.concatenate((\
13             U1[:nb_node],\
14             F1[:nb_node]),axis = 1),\
15             P1),axis = 1)
16 mesh.write_vtk_nodal_value(result = Solution, vec_title=['U','F'],\
17                            sca_title = ['P'], mesh = 'Qua4',\
18                            n = n, path = os.getcwd() + os.sep + 'DISPLAY',\
19                            name = 'solution', dim = 2)
20
21 n += 1
```

A faire :

- Incorporer dans les fichiers de sortie .vtk les résultats en vitesse du squelette solide.
- Ajouter de la compressibilité $\frac{1}{M}$ au modèle.
- Changer les conditions aux limites pour traiter un problème confiné.
- Utiliser un maillage sur base traingle 6 noeuds.
- Résoudre le problème de stokes pour une cavité.