Pre-Processeur LMGC90

Buts Préprocesseur LMGC90

MultiCorps

MultiPhysiques

Programmation Orienté Objet

Installation

Modules annexes(hors distribution python) `numarray`

repertoire d'installation = `PRE_LMGC`

sous répertoire un fichier `pre_lmgc.py`

1. `domain` contient l'ensemble des donnees pour construire des `BODIES`
2. `interaction` pas grand chose pour le moment
3. `output_files` place des fichiers de sorties par defaut.
4. `files` contient les fichiers permettant d ecrire les fichiers de sorties
5. `material` contient des fichiers permettant de gerer les materiaux
6. `common` contient des donnees pour tous.

PYTHON_PATH pour utiliser pre_lmgc de n importe où il faut modifier la variable d environnement

```
PYTHON_PATH=.:PRE_LMGC:PRE_LMGC/material/:PRE_LMGC/common/
```

Informations session intéractive

`python` lance python en session intéractive

`importation du module pre_lmgc` `from pre_lmgc import *`

`help(...)` pour trouver une information sur une classe, une methode de classe,...etc. Exemple :

```
help(mesh)
```

```
classe permettant de definir les parties maillees a partir d un fichier de maillage
       Methodes :
         - __init__
         - lecture
         - defineModele
         - defineMateriau
         - defineContacteurs
         - imposeDrivenDof
         - imposeInitValue
         - rotateBody
         - moveBody
         - addNodes
         - addGroups
         - addElements
```

Maillage définition dans `domain/MAILx`

`lecture d un fichier` `p=mesh(fichierMaillage='../epr.msh')`

`print p` donne des informations

`Instance de mesh` contient des ensembles:

1. d'éléments contenue sous `p.elements`
2. de noeuds contenue sous `p.nodes`
3. de groups contenue sous `p.groups`

`iterateur` chacun des ensembles est un iterateur `common/iterateur.py` avec des méthodes

Pour avoir l ensemble des groupes : `print p.groups.liste()`

On remarquera qu'il y a un groupe par entité physique mais aussi par type d'éléments et que le groupe 'tous' est défini.

Maillage Les groupes

On peut trouver un ensemble d'une entité (`nodes` ou `elements`) à l'aide de la méthode `find` d'une instance de `mesh` exemple : `p.find(type='nodes',group='6')`

Modification d'un nom `p.groups.modifieNom('6','bordGauche')`

Modification de plusieurs noms `p.groups.modifieNoms('6': 'bordGauche','9':'bordDroit')`

Maillage Les modèles

`class model` pour définir un modèle. Lorsqu'on definit un modele, on définit automatiquement un type de d.d.l associé.

`exemple` : `m =model('T3DNL',element='T3xxx',type='THERx',capaStorage='lump_')`
`print m` ; pour des informations.

`mesh` : association d un modèle est d'une partie. `p.defineModele(group='T3xxx',modele=m)` le 'model' ne s'appliquera qu'aux éléments géométriques compatibles avec ceux du 'model'. *defineModele(modele='T3DNx') est aussi valable mais il n y a plus la possibilité d'imposer des d.d.l.*

`conteneur` `ms=models();ms.addModele(m)`

Maillage Les matériaux

`class material` pour definir un matériau

`exemple` : `mat =material(nom='acier',type='ELAS',elas_modele='1',young=200000,nu=0.3)`

```
print mat ;

Materiau:acier
        Comportement de type    :ELAS
        Proprietes definies :
                              young    :       2000000.0
                                 nu    :       0.3
                         elas_modele   :       elas \Neo Hookeen \hyper
```

Tout ne doit pas encore être défini

Le conteneur de parts `domain/parts.py`

`class parts` création d'une instance `ps=parts()`

`Ajout d'une part` `p=mesh(numero='1',fichierMaillage='tt.msh');ps.addPart(p)`

`Accès à une part` `print ps['1']` pour imprimer les infos sur une partie.

`Relecture d un 'BODIES.DAT'` ps=parts_lmgc(chemin='') !A REVOIR

Un exemple complet sans imposition de dof

```
from pre_lmgc import * # importation du module pour le pre-processeur

p=mesh(numero='1',fichierMaillage='epr.msh') # lecture du maillage
                                             # 'epr.msh' et reperage par le numero 1

p.defineModele(modele='T3DNL')      # definition d un nom pour le modele
p.defineMateriau(materiau='acier')           # definition du nom du materiau
ps=parts()

ps.addPart(p)

writeBodies(ps)      # ecriture du fichier
                                             # 'BODIES.DAT' dans le repertoire pointe par la variabl
                                             # OUTPUT_DIR de common/variables.py
```

Les d.d.l

Définition à la création du modèle

Affectation à l'affectation d'un modèle sur un groupe.

Imposition sur un groupe pour un modèle et suivant une composante d'une valeur initiale ou imposée sous la forme d'une fonction `evolution`

Stockage en chaque noeud dans un conteneur `ddls` dont les clé sont celles du modèles.

Exemple p est une instance de mesh

```
...
>>>m=model(nom='M3DNL',type='MECAx',element='T3xxx')
>>>p.defineModele(group='T3xxx',modele=m)
>>>p.imposeDrivenDof(group='10',modele='MECAx',composante=1,ct=10.)
....
>>>writeDrvDof(ps)
>>>print p.nodes['1'].ddls['MECAx']
        ddl de type     :       vecteur  repere         :       global
                        composante 1 :  0.0      impose         :       True
                        composante 2 :  0.0      impose         :       False
                        composante 3 :  0.0      impose         :       False
```

Les Contacteurs

`Création` `p.defineContacteurs(...)` crée un conteneur p.contacteurs dérivant d'`iterateur`

`Caractéristiques` voir définition des candidats et antagonistes.

`Exemple`

```
...
>>>p.defineContacteurs(group='Line',type='CLxxx',apab=0.5)
...
>>>writeBodies(ps,chemin='')
```

Vérifier le `BODIES.DAT`.

Certains passage sont optionnel suivant le type de contacteurs ici `apab`. Si on refait une définition de contacteurs alors on va rajouter des contacteurs sur de même éléments 'lignes', ...