Organisme(s) : EDF-R&D/AMA

Manuel d'Utilisation
Fascicule U7.0- : Echanges de données
Document : U7.03.03

Méthodes Python de pilotage de GMSH

Résumé :

Ce document présente le superviseur permettant de piloter GMSH depuis Python, et donc depuis le fichier de commande Aster.

Ce superviseur produit tout type de maillages 2D en utilisant le logiciel GMSH (www.geuz.org/gmsh). Il est notamment utilisé dans Aster par l'outil de post-traitement interactif STANLEY afin de générer des éléments de maillages pour le post-traitement, mais peut être étendu à d'autres applications : maillage paramétrique, remaillage, etc.

1 Mode d'emploi

Il y a quatre étapes à suivre pour produire un maillage avec le superviseur GMSH :

- Définition de la géométrie ;
- Définition des discrétisations ;
- Création du maillage GMSH et des group_ma et objets Physical associés ;
- Importation du maillage GMSH dans Aster.

Exemple simple d'utilisation :

Dans l'exemple suivant, on utilise les fonctionnalités du superviseur pour générer le maillage d'une plaque rectangulaire :

Géométrie

```
  From sup_gmsh import *

  larg = 5.
  H_beton = 3.
  H_S1 = 4.
  t_beton   = 25.
  prog_S1 = 1.1
```

On importe le module et on définit quelques paramètres.

```
# Geometrie
  O = Point(0    ,  0      )
  A = Point(larg,  0      )
  B = Point(larg,  H_beton)
  C = Point(0    ,  H_beton)
  D = Point(0    , -H_S1   )
  E = Point(larg, -H_S1   )

  OA = Line(O,A)
  AB = Line(A,B)
  BC = Line(B,C)
  OC = Line(O,C)

  OD = Line(O,D)
  DE = Line(D,E)
  AE = Line(A,E)

  S2 = Surface(OA,AB,BC,OC)
  S1   = Surface(OD,DE,AE,OA)
```

On crée des points, des lignes entre les points et des surfaces à partir des lignes.

```
# Discretisation
  OA.Transfinite(1)
  BC.Transfinite(1)
  DE.Transfinite(1)

  N_beton = int(H_beton/t_beton + 0.5)
  AB.Transfinite(N_beton)
  OC.Transfinite(N_beton)

  N_S1 = Progress(H_S1, r=prog_S1, h=t_beton)
  OD.Transfinite(N_S1,prog_S1)
  AE.Transfinite(N_S1,prog_S1)

  S2.Transfinite()
  S1.Transfinite()
```

On définit la discrétisation des lignes et des surfaces.

```
 # Maillage
  mesh = Mesh()
  mesh.Physical('FOND',DE)
  mesh.Physical('LAT_G',OC,OD)
  mesh.Physical('LAT_D',AB,AE)
  mesh.Physical('INTERFAC',OA)
  mesh.Physical('HAUT',BC)
  mesh.Physical('S2',S2)
  mesh.Physical('S1',S1)
```

On crée l'objet maillage et on définit les groupes de mailles qui seront des `group_ma` dans la `SD maillage` Aster et des *Physical* dans GMSH (ces derniers seront nommés GM1, GM2, etc...).

```
MA = mesh.LIRE_GMSH(
                    MODI_QUAD = 'OUI'
                    )
```

Importation du maillage dans Aster : MA est un maillage Aster.

2 Liste des fonctions disponibles

La liste des fonctions est extraite directement du source, sup_gmsh.py, ce qui explique qu'elle soit en anglais.

2.1 Classe générique pour les objets géométriques

class Geometric :

```
private attribute
 parameters  : dictionnary of the attributes (except relation and parameters itself)
              see __getattr__ and __setattr__

Attributes
 num        : index among gmsh objects
 md         : mesh descriptor
 mesh       : related mesh object
 relation   : model object in case of coincidence


Public methods
 Is_point : return true is the object inherits of the Point class

 Is_line  : return true is the object inherits of the Line class

 Is_surface : return true is the object inherits of the Surface class

 Is_volume : return true is the object inherits of the Volume class

 Is_same_dimension : return true is both objects are of the same dimension
                   (point, line, surface or volume)
   in -> object to compare to self

 Duplicate   : duplicate an object and base its mesh_descriptor
              on the mesh_descriptor of the model

 Coincide    : assert that an object is coincident with a model one
              All the attributes are then automatically read from
              the model object (see __setattr__ and __getattr__).
   in -> model object


Private method

 Root :
   Provides the root object of an object, ie the object itself if there is no relation
   or the deepest model in case of relation.

 Geometric_coincide : check if a geometrical coincidence is possible
                    return information about the coincidence, false else.
   in -> model object

 Deep_coincide : proceed recursively to ensure coincidence of the relevant sub-objects
   in -> model object
   in -> correspond (information returned by Geometric_coincide)
```

__setattr__ : distinguish two sets of attributes
relation (to express a relation with a model object in case of coincidence)
all the other attributes which are stored in the dictionnary parameters
instead of the usual __dict__ if there is no relation (see Coincide)
and in the model object if there is a coincidence

__getattr__ : if the object is related (relation <> None) the attribute is read
in the model object. Else, it is read in the current object, actually
in the dictionnary parameters (see __setattr__)

Thanks to these two overloaded methods, the access to the attributes is usual if there is no relation whereas the attributes of the model object are accessed transparently if there is a relation.

__cmp__ :
The comparison of two objects involves possible coincidence. It is no more the object ids that are compared but the object roots (.relation if any).

Gmsh : produce the source code for Gmsh
in -> mesh

Gmsh_send : send a line code to the gmsh interpreter
in -> line_code (string)

Intermediate_meshing : produce the source code for the intermediate objects
in -> mesh

Object meshing : produce the source code for the current object
var -> object number (modified if several objects are created)

2.2 Fonctions pour les objets POINT

class Point(Geometric) :

Public methods
__init__ :
in -> coordinates (the 3rd is zero by default)

Size : set the size of the neighbouring elements
in -> size

Attractor : define the point as an attractor
in -> scale_x : size amplification factor in the x-direction
in -> scale_y : size amplification factor in the y-direction
in -> distance: influence distance for the perturbation

Attributes
coor : coordinates
size : neighbouring element size
attractor : parameters of the attractor

2.3 Fonctions pour les objets LIGNE

class Line(Geometric) :

LINE OBJECT

Public methods

Attractor : define the point as an attractor
in -> scale_x : size amplification factor in the x-direction
in -> scale_y : size amplification factor in the y-direction
in -> distance: influence distance for the perturbation

class Circle(Line) :

CIRCLE OBJECT

def Curve(l_x,l_y,l_z=None) :

CURVE OBJECT (in -> list of points)

2.4 Fonctions pour les objets SURFACE

class Surface(Geometric) :

SURFACE OBJECT (inherit from the Geometric class)

Public methods
__init__ :
in -> lines : external bounday of the surface (lines should be connected)

Holes : set the internal holes (surfaces)
in -> holes : list of holes

Boundary : checks that the boundary is a closed loop and returns the orientation of the edges

Ruled : declare the surface is a ruled one

Translate : translate the surface
in -> tran : (numpy) vector of translation

Recombine : recombine the surface (try to mesh with quadrangles instead of triangles)

Transfinite : Declare the mesh to be transfinite

Attributes
lines : list of external boundary lines
holes : list of internal holes (surfaces)
ruled : indicates (false or true) if the surface is a ruled surface
loops : list of boundary (external and internal) loops (computed when meshing)

2.5 Fonctions pour les opérations sur les maillages

class Mesh_Descriptor :

```
Attributes
  relation     Another mesh descriptor provides the mesh parameters
  parameters   dictionnary of the mesh parameters
               size         Point size
               transfinite  Transfinite mesh (0 or 1)
               number       Number of elements along a line (transfinite)
               progression  Progression of element size (transfinite)
               recombine    Recombine mesh or not

Specific access :
  md.parameter_name = xxx -> the relation is destroyed (set to None)
  xxx = md.parameter_name -> if there is a relation, the effective
                             parameter is looked for recursively

Deep copying : a relation is set to the model instead of a true copy
```

class Mesh :

```
def __init__(self, algo = 2, gmsh='gmsh') :

def Physical(self, name, *l_obj) : creation of Physical (GMSH object)

def Save(self, file = 'fort.geo') : save the geo file

def View(self) : launch GMSH with the current geo file

def Create(self, file = 'fort.19') : save the geo file and create the msh file

def Name(self, MA, CREA_GROUP_NO) : create the group_ma and/or the group_no

def LIRE_GMSH(self,
  UNITE_GMSH    = 19,
  UNITE_MAILLAGE = 20,
  MODI_QUAD     = 'NON',
  CREA_GROUP_NO = 'OUI'
  ) :

    Lecture du maillage (format Aster) a partir de sa definition
    (format sup_gmsh)
    UNITE_GMSH     = Numero d'unite logique pour le fichier msh
    UNITE_MAILLAGE = Numero d'unite logique pour le fichier mail
    MODI_QUAD      = 'OUI' si line->quad, 'NON' sinon
    CREA_GROUP_NO  = 'OUI'  si on cree les group_no, 'NON' sinon
```

2.6 Fonctions pour les transformations géométriques

def VectorProduct(u,v) :

def VectorNorm(u) :

class Rotation :
in -> A,C,B

Page laissée intentionnellement blanche.