

Methods Python of piloting of GMSH

Summary:

This document introduces the supervisor making it possible to control GMSH since Python, and thus since the command file Aster.

This supervisor produces any type of grids 2D by using software GMSH (www.geuz.org/gmsh). It is in particular used in Aster by the tool for postprocessing interactive STANLEY in order to generate elements of grid for postprocessing, but can be wide with other applications: parametric grid, mending of meshes, etc.

1 Instructions

There are four stages to follow to produce a grid with supervisor GMSH:

- 1) Definition of the geometry;
- 2) Definition of the discretizations;
- 3) Creation of grid GMSH and of `GROUP_MA` and associated objects "Physical";
- 4) Importation of grid GMSH in Aster.

Simple example of use:

In the following example, one uses the features of the supervisor to generate the grid of a rectangular plate:

Geometry

```
from Utilitai.sup_gmsh importation *

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

One imports the module and one defines some parameters.

```
# Geometry
O = Not (0 , 0 )
With = Not (larg, 0 )
B = Not (larg, H_beton)
C = Not (0 , H_beton)
D = Not (0 , - H_S1 )
E = Not (larg, - H_S1 )

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

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

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

One creates points, lines between the points and of surfaces starting from the lines.

```
# Discretization
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 ()
```

One defines the discretization of the lines and surfaces.

```
# Grid
mesh = Mesh ()
mesh. Physical ('BOTTOM', OF)
mesh. Physical ('LAT_G', OC, OD)
mesh. Physical ('LAT_D', AB, AE)
mesh. Physical ('INTERFAC', OA)
mesh. Physical ('HIGH', BC)
mesh. Physical ('S2', S2)
mesh. Physical ('S1', S1)
```

One creates the object grid and one defines the groups of meshes which will be `GROUP_MA` in SD grid Aster and of “*Physical*” in GMSH (the latter will be named *GM1*, *GM2*, etc...).

```
MY = mesh.LIRE_GMSH (
    MODI_QUAD = 'YES'
)
```

Importation of the grid in Aster: *MA* is a grid Aster.

2 List of the functions available

The list of the functions is extracted directly from the source, `sup_gmsh.py`, which explains why it is in English.

2.1 Generic class for the geometrical objects

```
class Geometric:
```

```
    private attribute
    parameters : dictionary of the attributes (except relation and parameters
itself)
                see __getattr and __setattr
```

Attributes

```
num          : index among gmsh objects
Mandeleivium : mesh descriptor
mesh         : related mesh object
relation     : model object in box of coincidence
```

Public methods

```
Is_point: return true is the object inherits of the Not 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 class Volume
```

```
Is_same_dimension: return true is both objects are of the same
dimension
```

```
                (not, line, surface gold volume)
in - > object to compares to coil
```

```
Duplicate : duplicate year object and bases its mesh_descriptor
one the mesh_descriptor of the model
```

```
Coincide  : assert that year object is coincides with has 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 year object, IE the object itself yew
there is No relation
however the deepest model in box of relation.
```

```
Geometric_coincide: check yew has geometrical coincidence is possible
return information about the coincidence, false
else.
```

```
in - > model object
```

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

`__setattr__`: distinguish two sets of attributes
relation (fast to has relation with has model object in box of coincidence)
all the other attributes which are stored in the dictionary parameters
instead of the usual `__dict__` yew there is No relation (see `Coincides`)
and in the model object yew there has coincidence

`__getattr__`: yew 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 dictionary parameters (see `__setattr__`)

Thanks to thesis two overloaded methods, the access to the attributes is usual yew
there is No relation whereas the attributes of the model object are accessed
transparently yew there has relation .

`__cmp__`:
The comparison of two objects involves possible coincidence. It is No more the object ids
that are compared goal the object roots (`.relation yew any`).

`Gmsh` : produce the source codes for Gmsh
in - > mesh

`Gmsh_send`: send has line code to the gmsh to interpret
in - > line_code (G-string)

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

`Object meshing`: produce the source codes for the current object
VAr - > object number (modified yew several objects are created)

2.2 Functions for the objects NOT

class Not (Geometric):

Public methods

`__init__`:
in - > coordinates (the 3rd is zero by defect)

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

`Attractor`: define the not ace year attractor
in - > `scale_x`: size amplification Factor in the X-direction
in - > `scale_y`: size amplification Factor in the there-direction
in - > `distance`: influence outdistances for the disturbance

Attributes

coord: coordinates
size: neighbouring element size
attractor: parameters of the attractor

2.3 Functions for the objects LINE

```
class Line (Geometric):
```

```
    LINE OBJECT
```

```
    Public methods
```

```
    Attractor: define the not ace year attractor
```

```
        in - > scale_x: size amplification Factor in the X-direction
```

```
        in - > scale_y: size amplification Factor in the there-direction
```

```
        in - > distance: influence outdistances for the disturbance
```

```
class Circle (Line):
```

```
    CIRCLE OBJECT
```

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

```
    CURVE OBJECT (in - > list of points)
```

2.4 Functions for the objects 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 has closed loop and returns the  
orientation of the edges
```

```
    Ruled: the surface is declares has ruled one
```

```
    Relocate: relocate the surface
```

```
        in - > tran: (numpy) vector of translation
```

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

```
    Transfinite: The mesh to Be transfinite declares
```

```
Attributes
```

```
    lines: list of external boundary lines
```

```
    holes: list of internal holes (surfaces)
```

```
    ruled: indicates (false gold true) yew the surface has ruled surface
```

loops: list of boundary (external and internal) loops (computed when meshing)

2.5 Functions for the operations on the grids

class Mesh_Descriptor:

```
Attributes
  relation      Another mesh descriptor provides the mesh parameters
  parameters    dictionary of the mesh parameters
                size          Not size
                transfinite   Transfinite mesh (0 gold 1)
                number        Number of elements along has line
(transfinite)
                progression   Progression of element size
(transfinite)
                recombine     Recombine mesh gold not
```

Specific access:

```
md.parameter_name = xxx - > the relation is destroyed (set to None)
xxx = md.parameter_name - > yew there has relation, the effective
parameter is looked for recursively
```

Deep copying: relation is set to the model instead of has true Copy

class Mesh:

```
def __init (coil, algorithm = 2, gmsh=' gmsh'):
def Physical (coil, name, *l_obj): creation of Physical (GMSH object)
def Save (coil, file = 'fort.geo'): save the geo file
def View (coil): launch GMSH with the current geo file
def Create (coil, file = 'fort.19'): save the geo file and create the msh
file
def Name (coil, MY, CREA_GROUP_NO ) : create the group_ma and/or the
group_no
def LIRE_GMSH (coil,
  UNITE_GMSH      = 19,
  UNITE_MALLAGE   = 20,
  MODI_QUAD       = 'NOT' ,
  CREA_GROUP_NO   = 'YES'
) :
```

Reading of the grid (format Aster) starting from its definition (format sup_gmsh)

```
UNITE_GMSH      = logical Number of unit for the file msh
UNITE_MALLAGE   = logical Number of unit for the file e-mail
MODI_QUAD       = 'YES' if line->quad, 'NOT' if not
CREA_GROUP_NO   = 'YES' if they are created GROUP_NO, 'NOT' if not
```

2.6 Functions for the geometrical transformations

```
def VectorProduct (U, v):
```

```
def VectorNorm (U):
```

```
class Rotation:
```

in - > A, C, B