Tutorial: Solución de modelos 2D usando el Método de Elementos Finitos

Área de Mecánica Computacional

19 de mayo de 2017

Resumen

Este documento es un tutorial sobre cómo generar una geometría (específica) usando Gmsh[1] y su posterior procesamiento para la generación de archivos de entrada para un programa de Elementos Finitos en Python[2] . Este documento no pretende ser una introducción al manejo de Gmsh, para ello se sugiere el tutorial de la reference [4], el tutorial oficial de Gmsh [6] (para el manejo en modo texto) o los *screencasts* oficiales [7] (para el manejo de la interfaz gráfica). Este documento tampoco pretende ser un tutorial sobre el manejo de Python para Ingeniería, para ello se sugiere las *notas de clase* "SciPy Lecture notes" [3] o el libro [2].

1. Modelo a resolver

El ejemplo que se piensa resolver corresponde con la determinación de esfuerzos en un cilindro en la *Prueba Brasilera*. La Prueba Brasilera es una técnica que se usa para la medida indirecta de la resitencia de rocas. Es una técnica simple y efectiva, y por ello es comúnmente usada para medidas de rocas. En algunas ocasiones esta prueba se usa también para concreto [1].

La siguiente figura presenta un esquema del modelo a resolver. Ya que el modelo original puede presentar desplazamientos de cuerpo rígido, se decide utilizar la simetría del problema. Entonces, el problema a resolver es un cuarto del problema original y las superficies inferior e izquierda presentan restricciones de *rodillo*.

[1]Gmsh es un software libre para la generación de mallas 2D y 3D para simulaciones de elementos finitos [5]

[2]El programa de elementos finitos trabajado en clase escrito en Python que puede descargarse del sitio `https://github.com/jgomezc1/FEM_PYTHON`.

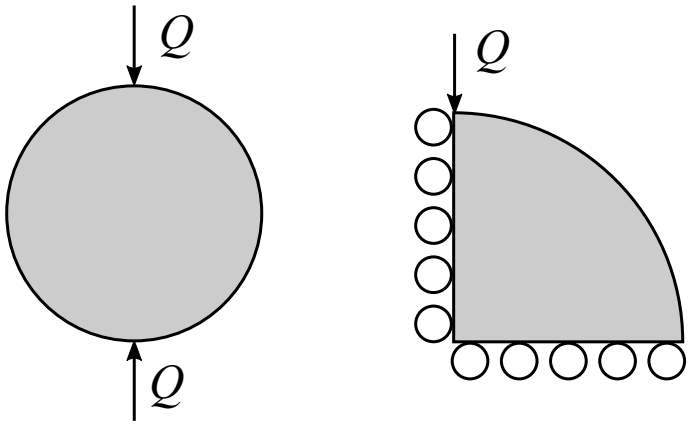

Figura 1. Esquema del modelo a resolver.

2. Generación de la geometría y malla en Gmsh

Como primer paso, se sugiere crear un nuevo archivo en Gmsh, como se muestra en la siguiente figura.

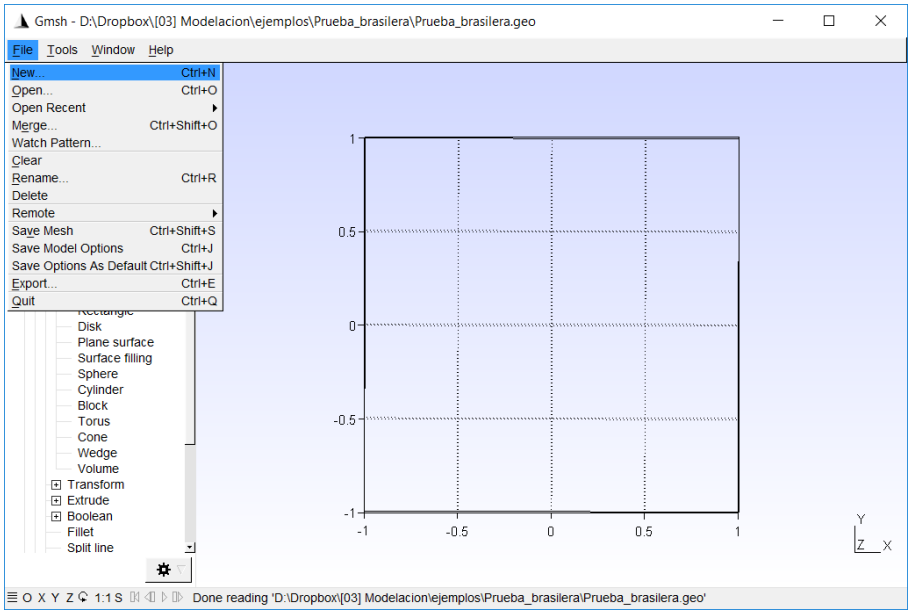

Figura 2. Creación de nuevo archivo en Gmsh.

Al crear un nuevo documento es posible[3] que Gmsh pregunte sobre cuál motor geométrico usar. No nos detendremos en cuáles son las diferencias y

[3]Si la versión es 3.0 o mayor, esta ventana emergente aparecerá

usaremos `built-in`.

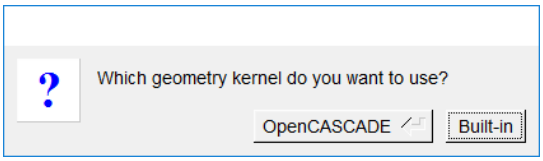

Figura 3. Ventana emergente preguntando por el motor geométrico.

Para crear un modelo, inicialmente creamos los puntos. Para ello, vamos a la opción: `Geometry >Elementary Entities >Add >Point`, como se muestra en la siguiente figura. Luego, se añaden las coordenadas de los puntos en la ventana emergente y "Add". Para finalizar podemos cerrar la ventana emergente y presionar `e`.

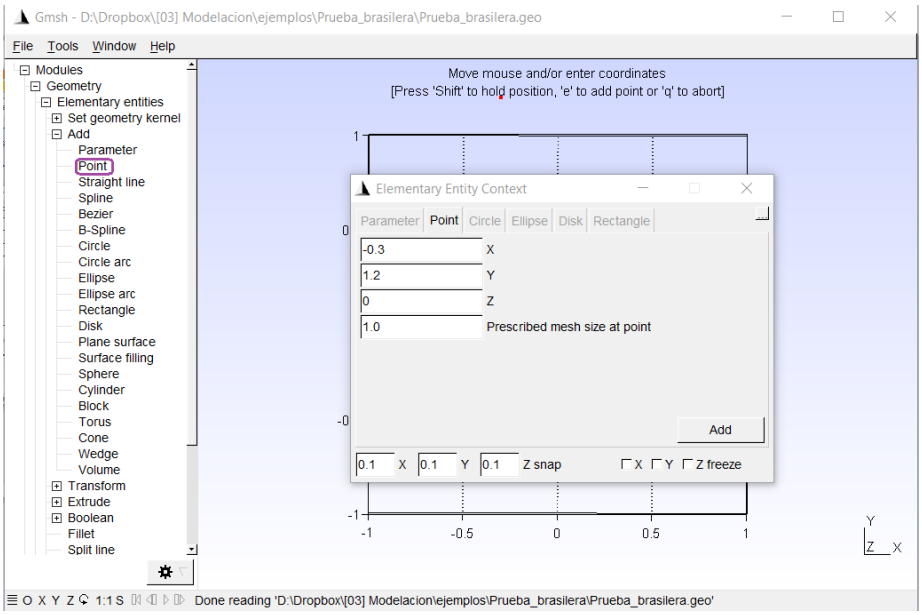

Figura 4. Agregar puntos al modelo.

Posteriormente creamos líneas. Para ello, vamos a la opción: `Geometry >Elementary Entities >Add >Straight line`, como se muestra en la siguiente figura, y seleccionamos los puntos iniciales y finales para cada línea. Al finalizar, podemos presionar `e`.

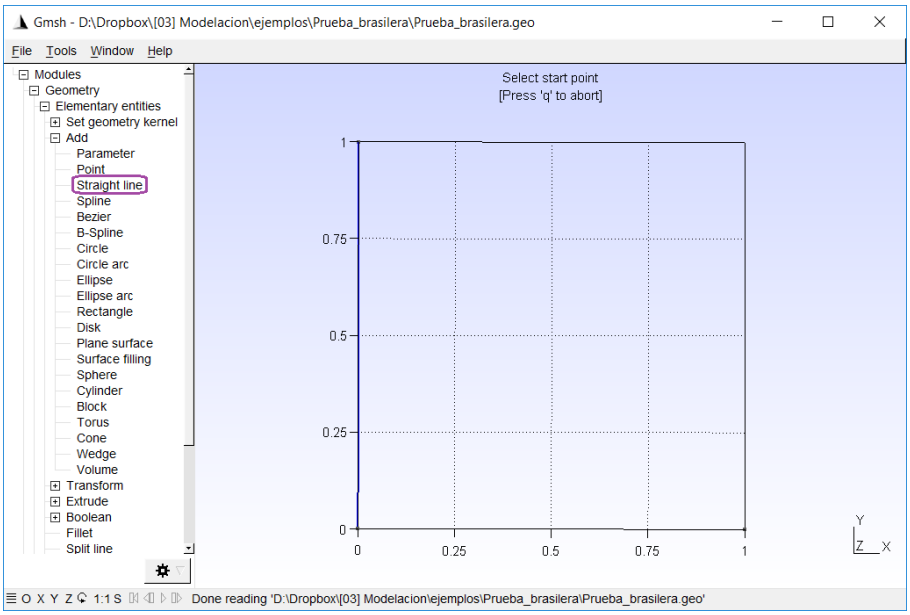

Figura 5. Agregar líneas rectas al modelo.

También creamos los arcos de circunferencia. Para ello, vamos a la opción: `Geometry >Elementary Entities >Add >Circle Arc`, como se muestra en la siguiente figura, y seleccionamos los puntos iniciales, centrales y finales para cada arco (en ese orden). Al finalizar, podemos presionar `e`.

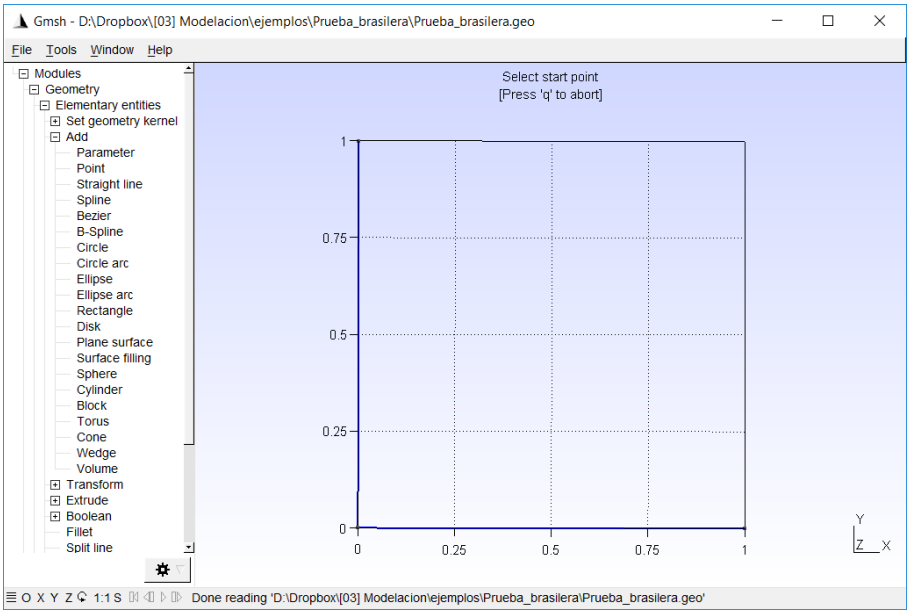

Figura 6. Agregar arcos al modelo.

Como ya tenemos un contorno cerrado, podemos definir una superficie. Para ello, vamos a la opción: `Geometry >Elementary Entities >Add`

`>Plane Surface`, como se muestra en la siguiente figura, y seleccionamos los contornos en orden. Al finalizar, podemos presionar `e`.

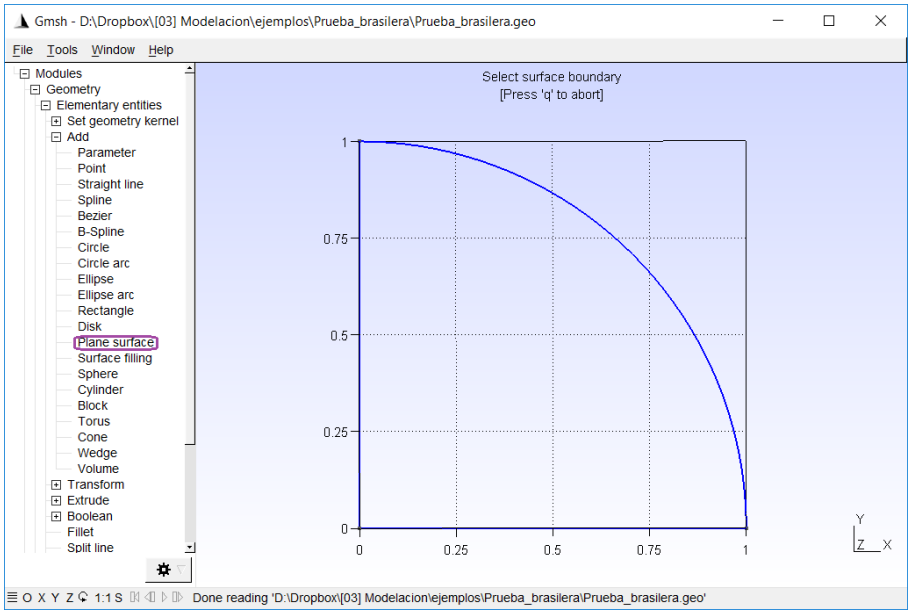

Figura 7. Agregar superficies al modelo.

Ahora, necesitamos definir *grupos físicos*. Los grupos físicos permiten asociar nombres a diferentes partes del modelo como líneas y superficies. Esto nos permitirá definir la región en la que resolveremos el modelo (y asociaremos un material), las regiones que tienen desplazamientos restringidos (condiciones de frontera) y las regiones sobre las que aplicaremos la carga. En nuestro caso tendremos 4 grupos físicos:

- Región del modelo, en donde definiremos un material;
- Borde inferior, en donde restringiremos el desplazamiento en y;
- Borde izquierdo, en donde restringiremos el desplazamiento en x; y
- Punto superior, en donde aplicaremos la carga puntual.

Para definir los grupos físicos vamos a `Geometry >Physical groups >Add >Plane Surface`, como muestra la siguiente figura. En este caso, podemos dejar el campo de `Name` vacío y permitir que Gmsh nombre los grupos por nosotros, los cuales serán números que luego podemos consultar en el archivo de texto.

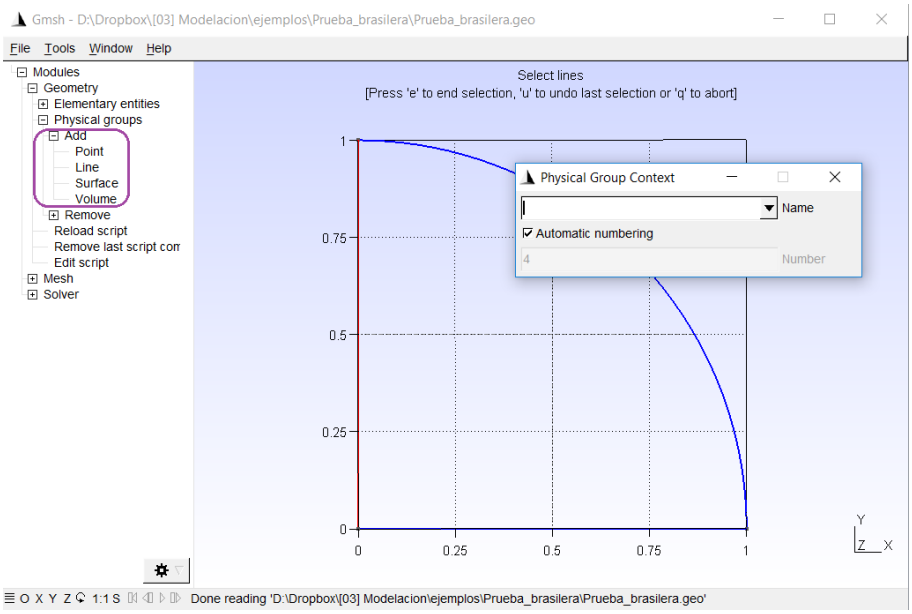

Figura 8. Agregar grupos físicos al modelo.

Luego de editar ligeramente, el archivo de texto (.geo) este luce de la siguiente manera. Agregamos un parámetro `L`, que podremos variar a nuestro antojo para cambiar el tamaño de los elementos al crear la malla.

```
L = 0.1;

// Puntos
Point(1) = {0, 0, 0, L};
Point(2) = {1, 0, 0, L};
Point(3) = {0, 1, 0, L};

// Líneas
Line(1) = {3, 1};
Line(2) = {1, 2};
Circle(3) = {2, 1, 3};

// Superficie
Line Loop(1) = {2, 3, 1};
Plane Surface(1) = {1};

// Grupos Físicos
Physical Line(1) = {1};
Physical Line(2) = {2};
Physical Point(3) = {3};
Physical Surface(4) = {1};
```

Código 1. Archivo `.geo` para el modelo descrito.

Ahora, procedemos a crear la malla. Para ello, vamos a `Mesh >2D`. Como vemos en la figura a continuación.

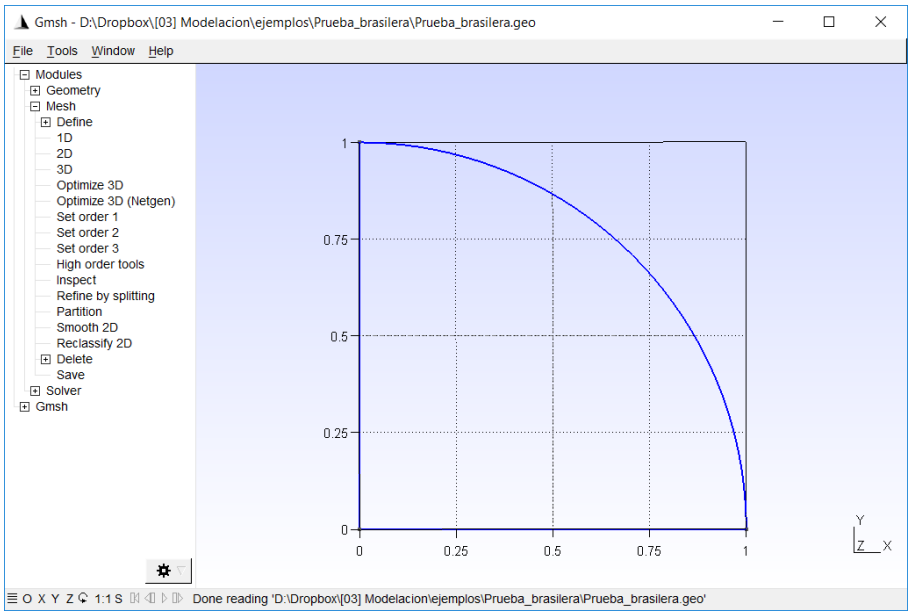

Figura 9. Crear la malla del modelo.

Adicionalmente, podemos cambiar la configuración para que muestre en colores, los elementos de la malla. Para ello, vamos a `Tools >Options >Mesh` y marcamos el cuadro que indica `Surface faces`.

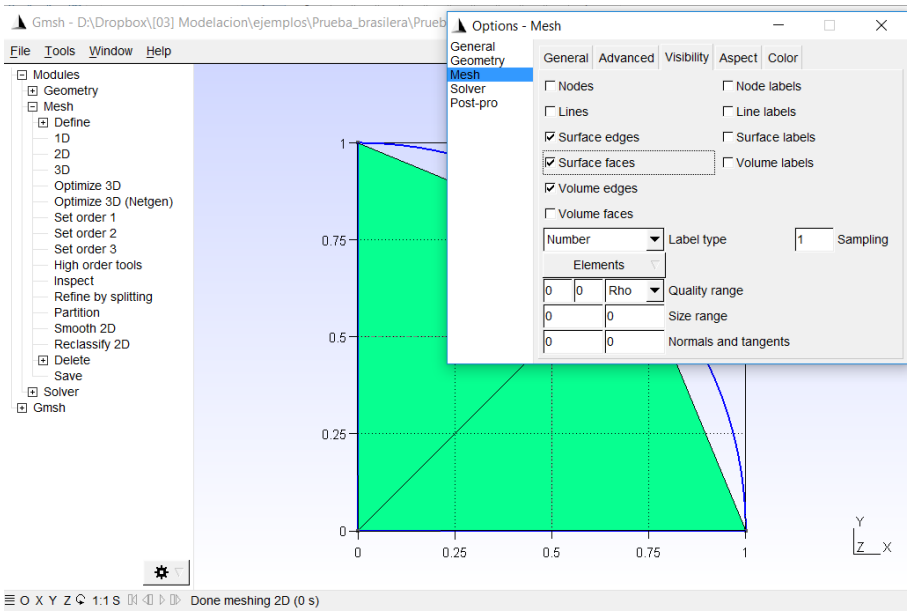

Figura 10. Crear la malla del modelo.

Podemos entonces refinar la malla yendo a `Mesh >Refine by Splitting`, o midificando el parámetro `L` en el archivo de entrada (.geo). Como últi-

mo paso, queremos salvar la malla. Para ello, vamos a `Mesh >Save`, o `File >Save Mesh`, como se muestra a continuación.

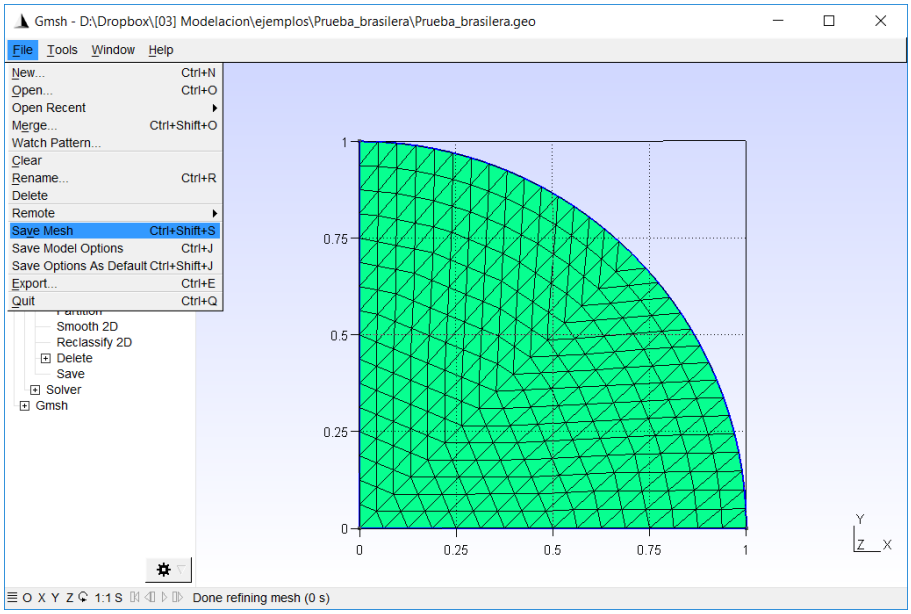

Figura 11. Guardar la malla a un archivo `.msh`.

3. Código para generar los archivos de entrada

Debemos crear archivos con la información de los nodos (`nodes.txt`), elementos (`eles.txt`), cargas (`loads.txt`) y materiales (`mater.txt`).

El siguiente código genera los archivos de entrada necesarios para correr el programa de elementos finitos en Python.

```
# -*- coding: utf-8 -*-
"""
Genera archivos de entrada para el programa de elementos finitos
FEM_iso para la prueba brasilera, usando 2 simetrías en el modelo.

@author: Nicolas Guarin-Zapata
@date: Mayo 18, 2017
"""
from __future__ import division, print_function
import meshio
import numpy as np


points, cells, point_data, cell_data, field_data = \
```

```
    meshio.read("Prueba_brasilera.msh")

# Datos elementales
eles = cells["triangle"]
els_array = np.zeros([eles.shape[0], 6], dtype=int)
els_array[:, 0] = range(eles.shape[0])
els_array[:, 1] = 3
els_array[:, 3::] = eles

# Nodos
nodes_array = np.zeros([points.shape[0], 5])
nodes_array[:, 0] = range(points.shape[0])
nodes_array[:, 1:3] = points[:, :2]

# Fronteras
lines = cells["line"]
bounds = cell_data["line"]["physical"]
nbounds = len(bounds)

# Cargas
id_cargas = cells["vertex"]
nloads = len(id_cargas)
load = -10e8 # N/m
loads_array = np.zeros((nloads, 3))
loads_array[:, 0] = id_cargas
loads_array[:, 1] = 0
loads_array[:, 2] = load

# Condiciones de frontera
id_izq = [cont for cont in range(nbounds) if bounds[cont] == 1]
id_inf = [cont for cont in range(nbounds) if bounds[cont] == 2]
nodes_izq = lines[id_izq]
nodes_izq = nodes_izq.flatten()
nodes_inf = lines[id_inf]
nodes_inf = nodes_inf.flatten()
nodes_array[nodes_izq, 3] = -1
nodes_array[nodes_inf, 4] = -1

#  Materiales
mater_array = np.array([[186e9, 0.29],
                        [70e9, 0.35]])
maters = cell_data["triangle"]["physical"]
els_array[:, 2]  = [0 if mater == 4 else 1 for mater in maters]

# Generar archivos
np.savetxt("eles.txt", els_array, fmt="%d")
```

```
61 np.savetxt("nodes.txt", nodes_array,
62             fmt=("%d", "%.4f", "%.4f", "%d", "%d"))
63 np.savetxt("loads.txt", loads_array, fmt=("%d", "%.6f", "%.6f"))
64 np.savetxt("mater.txt", mater_array, fmt="%.6f")
```

Ahora, comentemos las diferentes partes del código para ver qué hace cada una.

3.1. Encabezado y lectura de archivo `.msh`

La primera parte carga los módulos de Python necesarios y lee el archivo de malla que en este caso se llama `Prueba_brasilera.msh` (línea 6 y 7). Para que Python sea capaz de leer el archivo, este debe estar en el mismo directorio que el archivo de Python que lo procesará.

```
1 from __future__ import division, print_function
2 import meshio
3 import numpy as np
4
5
6 points, cells, point_data, cell_data, field_data = \
7     meshio.read("Prueba_brasilera.msh")
```

3.2. Datos de elementos

La siguiente sección del código crea los datos para elementos. La línea 18 crea una variable `eles` con la información de los nodos que conforman cada triángulo. La línea 19 crea un arreglo (lleno de ceros) con la cantidad de filas igual al número de elementos (`eles.shape[0]`) y 6 columnas[4]. Luego asignamos un número a cada elemento, esto lo hacemos en la línea 20 con `range(eles.shape[0])` y esto lo asignamos a la columna 0. Todos los materiales son triángulos, por eso debemos poner 3 en la columna 1. Por último, en esta sección, asignamos los nodos de cada elemento al arreglo con `els_array` (línea 22), y esta asignación la hacemos desde la columna 3 hasta el final con `els_array[:, 3::]`.

```
17 # Datos elementales
18 eles = cells["triangle"]
19 els_array = np.zeros([eles.shape[0], 6], dtype=int)
20 els_array[:, 0] = range(eles.shape[0])
21 els_array[:, 1] = 3
22 els_array[:, 3::] = eles
```

[4]Para elementos cuadriláteros se utilizarían 7 columnas, pues cada elemento está definido por 4 nodos.

3.3. Datos de nodos

En la siguiente sección creamos la información relacionada con los nodos. Para ello, en la línea 25 creamos un arreglo `nodes_array` con 5 columnas y tantas filas como puntos tenemos en el modelo (`point.shape[0]`). Luego, asignamos el número de elemento en la línea 26. Y, por último, asignamos la información de las coordenadas de los nodos en la línea 27 con `nodes_array[:, 1:3] = points[:, :2]`, en donde estamos poniendo la información en las columnas 1 y 2.

```
24 # Nodos
25 nodes_array = np.zeros([points.shape[0], 5])
26 nodes_array[:, 0] = range(points.shape[0])
27 nodes_array[:, 1:3] = points[:, :2]
```

3.4. Datos de Fronteras

En la siguiente sección encontramos la información de líneas. Para esto, leemos la información de `cells` en la posición `"line"`[5] (línea 30). El arreglo resultante `lines` tendrá, entonces, la información de los nodos que forman cada línea que está en la frontera del modelo. Luego, leemos la información de las líneas físicas (línea 31), y calculamos cuántas líneas pertenecen a las líneas físicas (línea 32).

```
30 # Fronteras
31 lines = cells["line"]
32 bounds = cell_data["line"]["physical"]
33 nbounds = len(bounds)
```

3.5. Datos de carga

En la siguiente sección debemos definir la información de cargas, en este caso, las cargas las asignamos en un único punto que definimos como grupo físico. En la línea 31 leemos los nodos (en este caso, uno). Luego, creamos un arreglo que tiene tantas filas como cargas (`nloads`) y 3 columnas. Asignamos el número del nodo al que pertenece cada carga (línea 35, las cargas en x (línea 36) y las cargas en y (línea 37).

```
30 # Cargas
31 id_cargas = cells["vertex"]
32 nloads = len(id_cargas)
33 load = -10e8 # N/m
```

[5] `cells` es un diccionario y permite almacenar información asociada a unas palabras clave, en este caso es `"lines"`.

```
34 loads_array = np.zeros((nloads, 3))
35 loads_array[:, 0] = id_cargas
36 loads_array[:, 1] = 0
37 loads_array[:, 2] = load
```

3.6. Condiciones de frontera

Ahora, procederemos a aplicar las condiciones de frontera, es decir, las regiones del modelo que tienen restricciones en el desplazamiento. Inicialmente, identificamos cuáles líneas tienen como identificador 1 (que serían las del lado izquierdo) con

```
id_izq = [cont for cont in range(nbounds) if bounds[cont] == 1]
```

Esto crea una lista con los números (`cont`) para los cuales se cumple la condición (`bounds[cont] == 1`). Ahora, en la línea 46 obtenemos los nodos que pertenecen a estas líneas, sin embargo, este arreglo tiene tantas filas como líneas en el lado izquierdo y dos columnas. Primero volvemos este arreglo como un arreglo unidimensional con `nodes_izq.flatten()`. Posteriormente, en la línea 50, asignamos el valor de -1 en la tercera columna del arreglo `nodes_array` para los nodos que pertenezcan al lado izquierdo. De igual forma, se repite este proceso para los nodos en la frontera inferior.

```
30 # Condiciones de frontera
31 id_izq = [cont for cont in range(nbounds) if bounds[cont] == 1]
32 id_inf = [cont for cont in range(nbounds) if bounds[cont] == 2]
33 nodes_izq = lines[id_izq]
34 nodes_izq = nodes_izq.flatten()
35 nodes_inf = lines[id_inf]
36 nodes_inf = nodes_inf.flatten()
37 nodes_array[nodes_izq, 3] = -1
38 nodes_array[nodes_inf, 4] = -1
```

3.7. Materiales

En la siguiente sección asignamos los materiales correspondientes a cada elemento. En este caso, sólo tenemos un material. Sin embargo, se presenta el ejemplo como si hubieran dos diferentes. Primero, creamos un arreglo con la información de materiales en donde la primera columna representa el módulo de Young y la segunda la relación de Poisson (línea 54). Luego, leemos la información de los grupos físicos de superficies en la línea 56. Finalmente, asignamos el valor de 0 a los materiales que tengan como grupo físico 4 (ver archivo `.geo` arriba) y 1 a los otros, que en este caso serán cero (línea 57). Esta información va en la columna 2 del arreglo `els_array`.

```
#  Materiales
mater_array = np.array([[186e9, 0.29],
                        [70e9, 0.35]])
maters = cell_data["triangle"]["physical"]
els_array[:, 2]  = [0 if mater == 4 else 1 for mater in maters]
```

3.8. Materiales

La última sección usa la función `savetxt` de `numpy` para exportar los archivos.

```
# Generar archivos
np.savetxt("eles.txt", els_array, fmt="%d")
np.savetxt("nodes.txt", nodes_array,
           fmt=("%d", "%.4f", "%.4f", "%d", "%d"))
np.savetxt("loads.txt", loads_array, fmt=("%d", "%.6f", "%.6f"))
np.savetxt("mater.txt", mater_array, fmt="%.6f")
```

4. Solución del modelo por Elementos Finitos

Para resolver el modelo se debe ejecutar el programa `solids_ISO_GUI.py`[6]. Luego de correr este programa aparecerá una ventana emergente como se muestra a continuación. En esta ventana emergente se debe ubicar el directorio en donde están los archivos de entrada generados anteriormente. Tenga en cuenta que la apariencia de esta ventana puede variar entre sistemas operativos. También tenga en cuenta que en algunas ocasiones la ventana emergente puede quedar oculta por otras ventanas en su escritorio.

[6]Para hacer uso de la interfaz gráfica debe estar instalado `eaygui`.

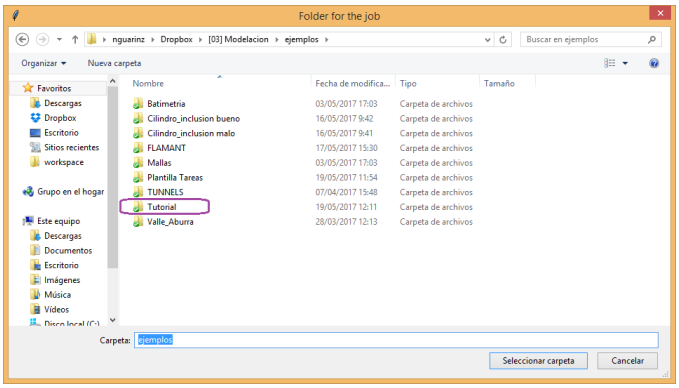

Figura 12. Ventana emergente para ubicar el directorio con los archivos de entrada.

Luego aparece una ventana emergente que pregunta por el nombre del proyecto.

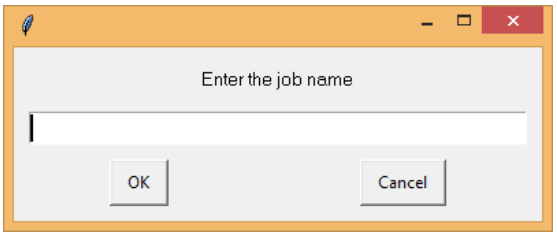

Figura 13. Ventana emergente para escribir el nombre del proyecto.

A continuación, aparece una ventana emergente preguntando si desea exportar algunos archivos, presiones No. Luego aparece una ventana emergente que pregunta por el nombre del proyecto.

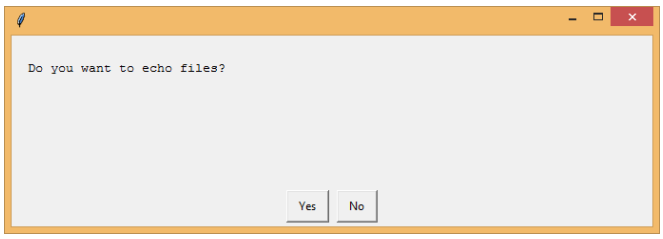

Figura 14. Ventana emergente preguntando por la escritura de archivos.

En este punto, el programa debe resolver su modelo. Si se usan los archivos de entrada sin modificaciones el programa debe imprimir lo siguiente en la consola de Python

```
Number of nodes: 123
Number of elements: 208
```

```
Number of equations: 224
Duration for system solution: 0:00:00.086983
Duration for post processing: 0:00:00
Analysis terminated successfully!
```

los tiempos que se toma en solucionar el sistema pueden cambiar un poco de un computador a otro.

Como último paso, el programa genera unos gráficos con los campos de desplazamientos, deformaciones y esfuerzos, como se muestra en la siguiente figura.

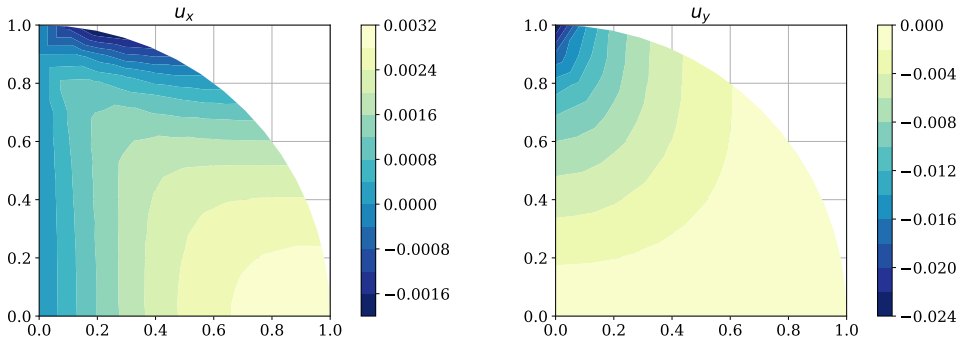

Figura 15. Solución de desplazamientos para el modelo. **(Izquierda)** Desplazamientos horizontales. **(Derecha)** Desplazamientos verticales.

A. Ejemplo de archivos de entrada para una malla sencilla

Supongamos que tenemos un cuadrado de lado 1, con dos cargas en los nodos superiores y los nodos inferiores empotrados, y que está hecho de acero 1020, el cual tiene un módulo de Young $E = 186$ GPa y una relación de Poisson de $\nu = 0{,}29$.

Si subdividimos el cuadrado en dos triángulos, tendríamos como archivos de entrada

```
1 0  0.0  0.0 -1 -1
2 1  1.0  0.0 -1 -1
3 2  1.0  1.0  0  0
4 3  1.0  0.0  0  0
```

Código 2. Archivo `nodes.txt`.

```
0  3  0  0  1  3
1  3  0  1  2  3
```

Código 3. Archivo `eles.txt`.

```
2  1  1
3  1  1
```

Código 4. Archivo `loads.txt`.

```
185e9  0.29
```

Código 5. Archivo `mater.txt`.

Si quisiéramos generar esta malla a partir de Gmsh, tendríamos como archivo `.geo`:

```
// Puntos
Point(1) = {0, 0, 0, 2.0};
Point(2) = {1, 0, 0, 2.0};
Point(3) = {1, 1, 0, 2.0};
Point(4) = {0, 1, 0, 2.0};

// Líneas
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};

// Superficie
Line Loop(1) = {1, 2, 3, 4 };
Plane Surface(1) = {1};

// Grupos Físicos
Physical Line(1) = {1};
Physical Line(2) = {3};
Physical Surface(4) = {1}

// Mallado regular
Transfinite Surface {1};
```

Código 6. Archivo `Cuadrado.geo`.

Referencias

[1] ASTM D3967–16 (2016), Standard Test Method for Splitting Tensile Strength of Intact Rock Core Specimens, ASTM International,`www.astm.org`.

[2] Linge, Svein, y Hans Petter Langtangen. 2016. Programming for computations – Python: a gentle introduction to numerical simulations with Python. `http://link.springer.com/book/10.1007%2F978-3-319-32428-9`.

[3] Gaël Varoquaux, Emmanuelle Gouillart, y Olav Vahtras (2016). Getting started with Python for science, en Scipy Lecture Notes. Fecha de consulta: mayo 13, 2017 desde `http://www.scipy-lectures.org/intro/index.html`.

[4] Avdis, A., y S. L. Mouradian (2012), *A Gmsh tutorial*, Fecha de consulta: mayo 13, 2017 desde `https://albertsk.files.wordpress.com/2012/12/gmsh_tutorial.pdf`.

[5] Geuzaine, Christophe, y Jean-François Remacle (2009), *Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities*. International Journal for Numerical Methods in Engineering, 79.11.

[6] Geuzaine, Christophe, y Jean-François Remacle (2017), Tutorial Oficial de Gmsh. Fecha de consulta: mayo 13, 2017 desde `http://gmsh.info/doc/texinfo/gmsh.html#Tutorial`.

[7] Geuzaine, Christophe, y Jean-François Remacle (2017), *Screencasts* oficiales de Gmsh. Fecha de consulta: mayo 13, 2017 desde `http://gmsh.info/screencasts/`.