

Elastic Calculator

Visualización de Soluciones Elásticas en Dominios Planos Discretizados con Elementos Finitos

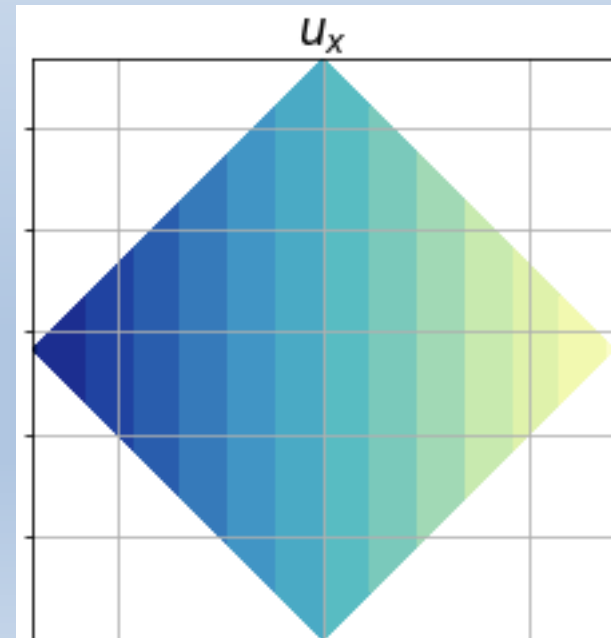
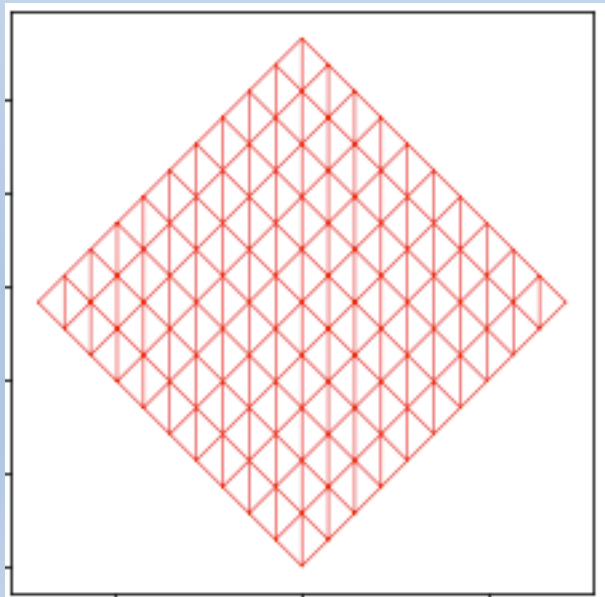
Modelación Computacional

Programa de Análisis de Sólidos Elásticos por el Método de los Elementos Finitos

SolidsPy

(<https://github.com/jgomezc1/Elastic-Calculator>)

El programa permite la visualización de soluciones sobre dominios planos utilizando métodos de interpolación local. Para visualizar una solución se requiere una discretización del dominio por medio de elementos finitos. Esta puede hacerse con la ayuda del mallador libre Gmsh. Posteriormente se evalúa la solución en los nodos de los elementos y esta se almacena en arreglos de tamaño variable dependiendo del problema a resolver. En la parte final la solución es pasada al modulo de visualización el cual triangula cada uno de los elementos finitos y realiza interpolación local para permitir una visualización de alta resolución.



Estructura del Programa

elasticity.py

Modulo que almacena todas las soluciones disponibles.

interfaces.py
generategeo.py

Modulos (de uso opcional) con interfaces gráficas simples para ingreso de parámetros de las soluciones disponibles y con geometrías automatizadas en Gmsh.

plotter.py

Modulo con las rutinas de graficación como tal. Además contiene algunas rutinas para transformación de tensores, etc.

Estructura del Programa

Para poder visualizar una solución el programa requiere: la solución como tal: esta debe ser función de las variables independientes x y y ; un arreglo con las conectividades de todos los elementos en los que se dividió la geometría del problema; y un arreglo con las coordenadas de todos los nodos en los que se evaluará la solución y que conforman los elementos.

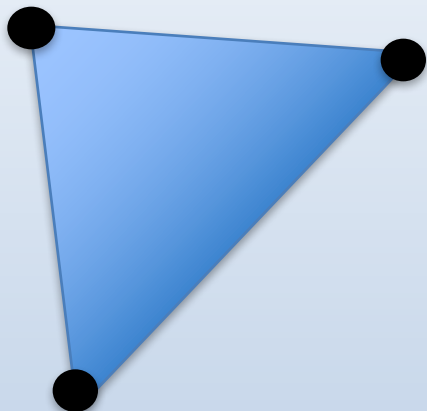
Genera discretización por elementos finitos resultando en los arreglos
nodes y elements

Evalúa la solución en todos los nodos del arreglo nodes resultando en
vectores solución SOL[]

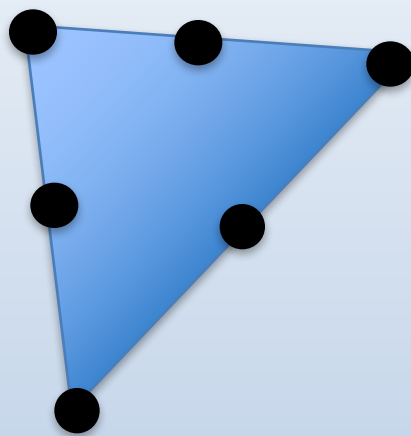
Pasa la malla (nodes y elements) y el vector solución al modulo de
graficación `plotter.py`

Tipos de Elementos

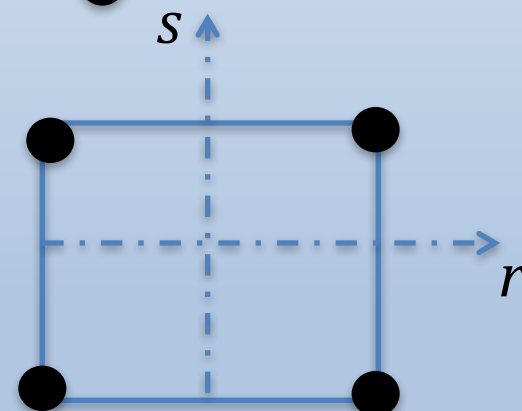
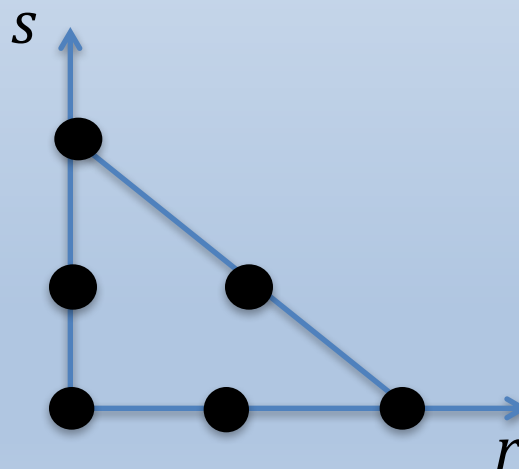
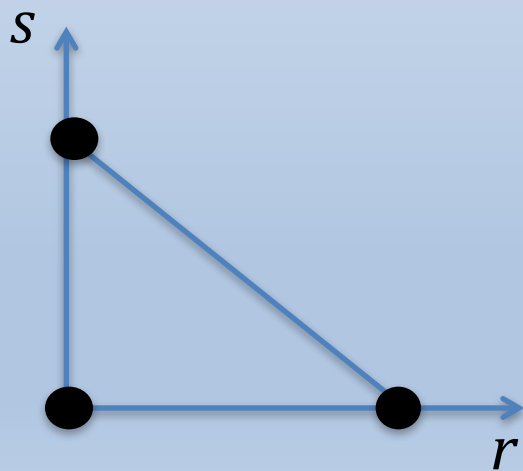
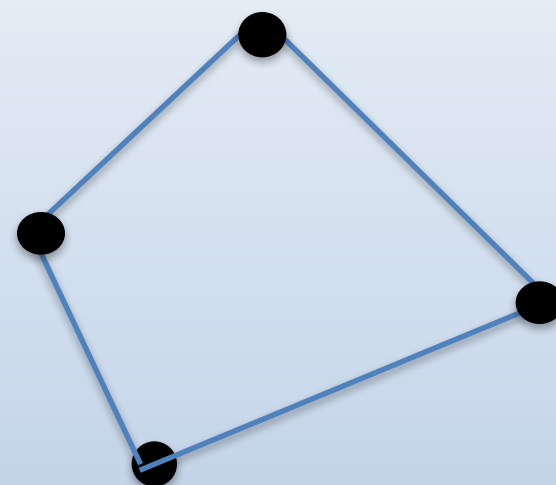
ele_type = 2
orden = 1



ele_type = 9
orden = 2



ele_type = 3
orden = 1



Arreglo nodes[]

Id_Nudo X-Coord Y-Coord

Id_Node: Identificador de nodo (I5).

X-Coord: Coordenada en x (f10).

Y-Coord: Coordenada en y (f10).

1	0.0	0.0
2	2.0	0.0
3	4.0	0.0
4	0.0	3.0
5	2.0	3.0
6	4.0	3.0
7	0.0	6.0
8	2.0	6.0
9	4.0	6.0

Arreglo elements[]

Id_Elemento **Id-Tipo-Ele** 0 **Id-N₁** **Id-N₂** **Id-N₃****Id-N_N**

Id_Elemento: Identificador de elemento (I5).

Id-Tipo-Ele: Identificador de tipo de elemento (3-triángulo de 3 nudos; 2-triángulo de 6 nudos; 1-cuadrilátero de 4 nudos).

Id-N_i: Identificadores de nudo. Tantos nudos como se defina de acuerdo el tipo de elemento definido por **Id-Tipo-**

1	1	0	1	2	5	4
2	1	0	2	3	6	5
3	1	0	4	5	8	7
4	1	0	5	6	9	8

Nodos que
conforman los
elementos

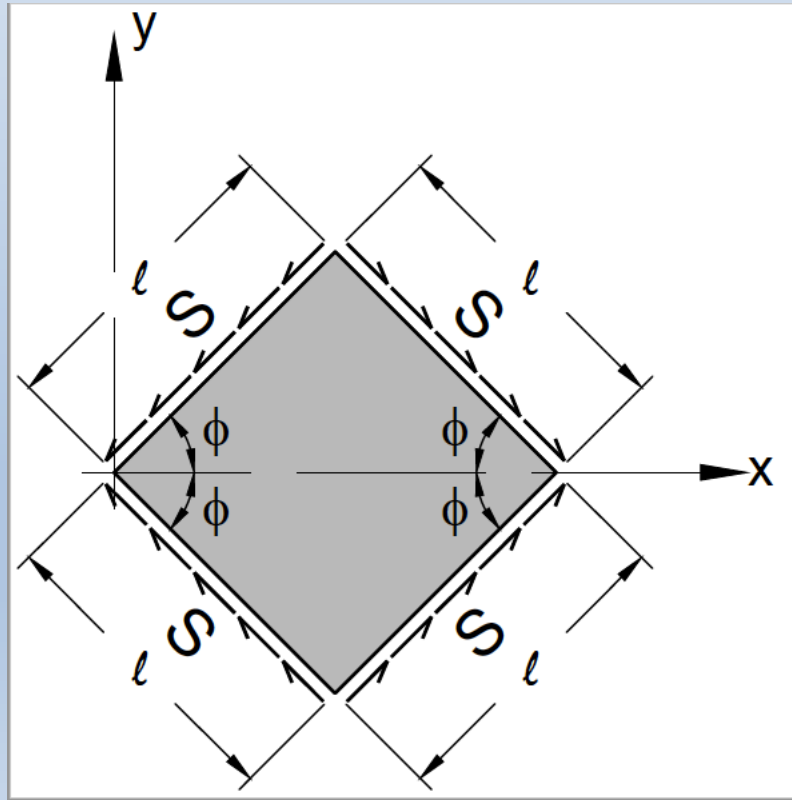
Tipo de elemento

Ejemplo de implementación (wedge.py)

Implementar la solución

$$\sigma = \begin{bmatrix} +SCot(\phi) & 0 \\ 0 & -STan(\phi) \end{bmatrix}$$

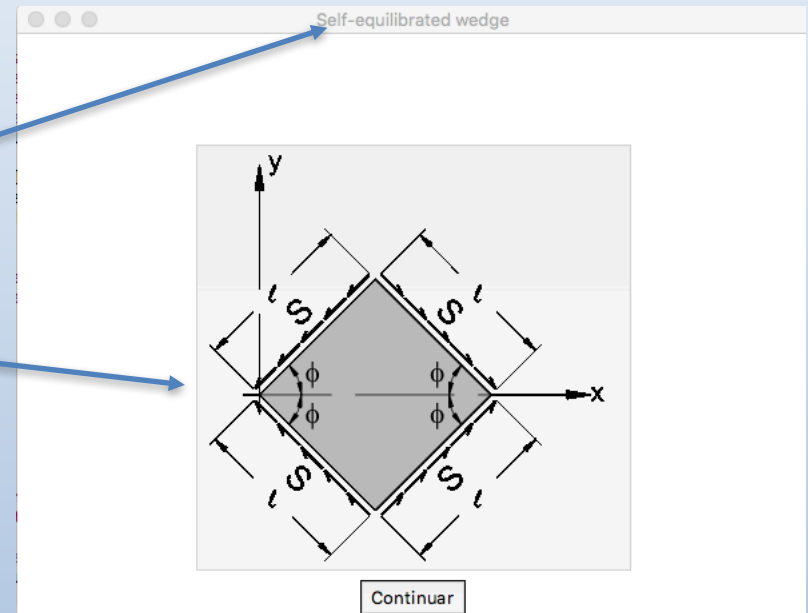
En el dominio correspondiente a la cuña mostrada en la figura.



Ayuda de la solución (Opcional)

Es posible agregar al modulo interfaces.py una rutina que muestre una imagen con el dominio de solución y los parámetros que sean relevantes. En este caso la rutina se denomina `gui.wedge_hlp`.

```
def wedge_hlp():  
    try:  
        import easygui  
        easygui.msgbox(msg="",  
            title="Self-equilibrated wedge",  
            ok_button="Continuar",  
            image='cunia.gif')  
    except:  
        print ("No easygui module")  
  
    return
```



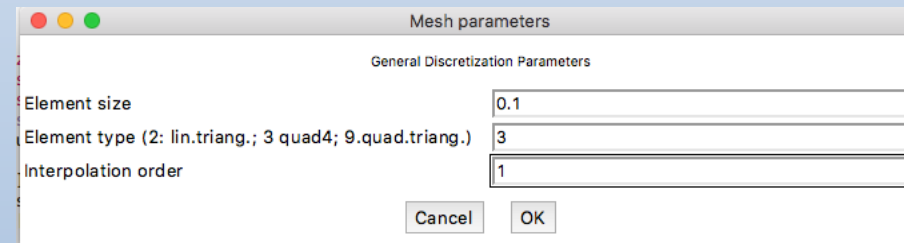
Parametros de la malla (Opcional)

La interface de mallado es igual para todas las soluciones y se usa al principio del script principal para definir las características de la malla de elementos finitos, solicitando al usuario 3 datos:

- Tamaño característico del elemento
- Tipo de elemento (2 triangulo líneal; 3 cuadrilatero líneal; 9 triangulo cuadratico)
- Orden de interpolación: 1 o 2 según sea el caso.

Esta interface no es necesaria si la malla se crea en un paso independiente del script principal.

```
def mesh_gui():  
    try:  
        import easygui  
        msg = "General Discretization Parameters"  
        title = "Mesh parameters"  
        fieldNames = ["Element size", "Element type",  
            "(2: lin.triang.; 3 quad4; 9.quad.triang.)", "Interpolation order"]  
        fieldValues = [] # we start with blanks for the values  
        fieldValues = easygui.multenterbox(msg, title, fieldNames)  
  
        c = float(fieldValues[0])  
        ietype = int(fieldValues[1])  
        order = int(fieldValues[2])  
    except:  
        c1 = raw_input("Element size")  
        ietype1 = raw_input("Element type")  
        order1 = raw_input("Interpolation order")  
        c = float(c1)  
        ietype = int(ietype1)  
        order = int(order1)  
  
    return c, ietype, order
```

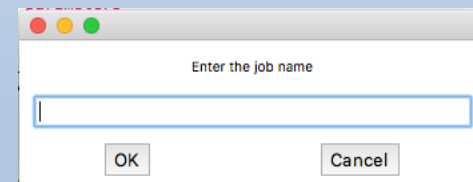
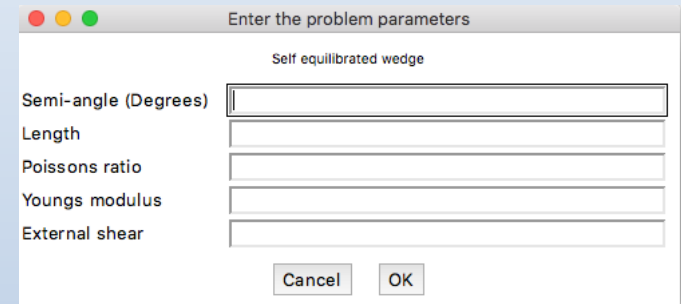


Mesh parameters	
General Discretization Parameters	
Element size	0.1
Element type (2: lin.triang.; 3 quad4; 9.quad.triang.)	3
Interpolation order	1
<div>Cancel OK</div>	

Parametros de la solución (Opcional)

En este menú se programa el ingreso de los parámetros relevantes de la solución como dimensiones, propiedades de los materiales, magnitudes de cargas, etc.

```
def wedge_prs():  
  
    try:  
        import easygui  
        msg = "Self equilibrated wedge"  
        title = "Enter the problem parameters"  
        fieldNames = ["Semi-angle (Degrees)", "Length", "Poissons ratio" ,  
            "Youngs modulus" , "External shear"]  
        fieldValues = []  
        fieldValues = easygui.multenterbox(msg,title, fieldNames)  
  
        phid = float(fieldValues[0])  
        l = float(fieldValues[1])  
        enu = float(fieldValues[2])  
        emod = float(fieldValues[3])  
        S = float(fieldValues[4])  
    except:  
        a1 = raw_input("Semi-angle")  
        b1 = raw_input("Length")  
        c1 = raw_input("Poissons ratio")  
        d1 =raw_input("Youngs modulus")  
        e1 = raw_input("External shear")  
        phid = float(a1)  
        l = float(b1)  
        enu = float(c1)  
        emod = float(d1)  
        S = float(e1)  
  
    return phid , l , enu , emod , S
```



Generación de la malla (Opcional)

En el caso de dominios simples es posible automatizar la creación de la geometría mediante una rutina que escribe un archivo con los comandos de Gmsh para generar el archivo .geo así como la ejecución de Gmsh para crear la malla como tal. Tanto la creación de la geometría como la ejecución de Gmsh puede realizarse por fuera del script en cuyo caso estas 2 rutinas no son requeridas.

```
def wedge(l, fi, c, ietype):
    """
    Creates model.
    geo.ring(a, b, c, ietype)
    c : element size
    ietype = 3 (Bi-linear quad)
    ietype = 9 (Quadratic triangle)
    ietype = 2 (Linear triangle)
    """
    try:
        import easygui
        var = easygui.enterbox("Enter the job name")

    except:
        var = raw_input('Enter the job name: ')
        file_name=open(var+'.geo', 'w')

    file_name.write('%23s \n' % ('// Input .geo for wedge'))

    file_name.write('%21s \n' % ('// author: Juan Gomez'))

    file_name.write('%1s \n' % (' '))

    file_name.write('%4s %6.3f %25s \n' % ('c = ', c, ';'))

    file_name.write('%0s \n' % (''))

    file_name.write('%3s %40.36f %1s \n' % ('l= ', l, ';'))

    file_name.write('%4s %12.8f %1s \n' % ('fi= ', fi, ';'))
```

```
def create_mesh(order, fname=''):
    """Create a mesh file using Gmsh

    Parameters
    -----
    order : int
        Order of the finite elements.
    var : string
        Name of the .geo file.
    seemesh : Boolean
        Right now, it does nothing.

    Returns
    -----
    file_creation : Boolean
        ``True`` if the mesh file was created.

    """
    orden = str(order)
    path = os.path.dirname(__file__)
    exit_status = os.system("gmsh {}.geo -2 -order {}".format(fname,
                                                                orden))

    if exit_status != 256:
        config_file = open("{}config.yml".format(path))
        config_data = yaml.load(config_file)
        gmsh_path = config_data["gmsh_path"]
        os.system("{}gmsh {}.geo -2 -order {}".format(
            gmsh_path, fname, orden))
        file_creation = os.path.isfile("{}msh".format(fname))
    return file_creation
```

Procesamiento de la malla (Obligatorio)

Una vez generada la malla ahora es necesario procesar la misma para generar los arreglos de nudos y elementos en el formato presentado anteriormente. Para procesar la malla se requiere tener instalado el modulo **meshio** y la siguiente rutina del modulo **generate.geo**

```
def writefiles(ietype, var=''):
    points, cells, point_data, cell_data, field_data = \
        meshio.read(var + '.msh')
    if ietype == 2:
        elements = cells["triangle"]
        els_array = np.zeros([elements.shape[0], 6], dtype=int)
    elif ietype == 9:
        elements = cells["triangle6"]
        els_array = np.zeros([elements.shape[0], 9], dtype=int)
    elif ietype == 3:
        elements = cells["quad"]
        els_array = np.zeros([elements.shape[0], 7], dtype=int)
    els_array[:, 0] = range(elements.shape[0])
    if ietype == 2:
        els_array[:, 1] = 2
    elif ietype == 3:
        els_array[:, 1] = 3
    elif ietype == 9:
        els_array[:, 1] = 0
    els_array[:, 1:] = elements
    nn = points.shape[0]
    nodes_array = np.zeros([points.shape[0], 3])
    nodes_array[:, 0] = range(points.shape[0])
    nodes_array[:, 1:3] = points[:, :2]
    np.savetxt("eles.txt", els_array, fmt="%d")
    np.savetxt("nodes.txt", nodes_array, fmt="%d", "%.4f", "%.4f")

    nodes = np.loadtxt('nodes.txt')
    elements = np.loadtxt('eles.txt')
    nn = len(nodes[:, 0])

    return nodes, elements, nn
```

Lee la malla de gmsh y la
almacna en diccionarios
listos para procesamiento

Captura los nodos de los
elementos de acuerdo con
el tipo de elemento

Escribe el tipo de elemento
al archivo eles.txt

Escribe los nudos de cada
elemento en el archivo
eles.txt

Escribe la información
nodal en el archivo
nodes.txt y finalmente los
almacena en los arreglos
nodes y elements listos
para evaluar y graficar la
solución.

Solución (Obligatorio)

La solución como tal se debe agregar al modulo **elasticity.py**. En la figura se muestra la solución correspondiente al problema de la cuña

```
def cunia(x,y,phi,l,nu,E,S):  
    """Computes the solution for self-equilibrated wedge  
        at a point (x , y)  
  
    Parameters  
    -----  
    nu :float, (-1, 0.5)  
        Poisson coefficient.  
    S :float  
        Applied shear traction over the faces of the wedge.  
    E :float, >0  
        Young modulus.  
    l :float, >0  
        Length of the inclined face of the wedge.  
    phi :float, >0  
        Half-angle of the wedge.  
  
    Returns  
    -----  
    ux : float  
        Horizontal displacement at (x , y).  
    uy : float  
        Vertical displacement at (x , y).  
    References  
    -----  
    .. [1] Timoshenko, S. & Goodier, J., 1970. Theory of Elasticity,  
        McGraw-Hill, 3rd Ed.  
  
    """  
    #  
    K1=(np.cos(phi)/np.sin(phi))+nu*(np.sin(phi)/np.cos(phi))  
    K2=(np.sin(phi)/np.cos(phi))+nu*(np.cos(phi)/np.sin(phi))  
    ux=(S/E)*K1*(x-l*np.cos(phi))  
    uy=-(S/E)*K2*y  
    sigx = S*(np.cos(phi)/np.sin(phi))  
    sigy =-S*(np.sin(phi)/np.cos(phi))  
    return ux , uy , sigx , sigy
```

La función recibe como parámetros de entrada las coordenadas del punto y los parámetros particulares de la solución. En este caso el ángulo de la cuña, la longitud de la cara, la relación de Poisson, el módulo de elasticidad y el cortante aplicado

La función devuelve como parámetros los campos de interés a graficar. En este caso la solución está dada en términos de las componentes horizontal y vertical del vector de desplazamientos así como las tensiones normales en dirección x y y.

Script de la solución

El script de la solución genera la malla y captura los parámetros de la solución para posteriormente evaluarla en los nodos y pasarla al modulo plotter para hacer la visualización.

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Wed Nov 11 12:36:56 2015
```

```
@author: eafit
```

```
"""
```

```
import numpy as np
from os import sys
sys.path.append('../CALCULATOR/')
from sympy import init_printing
init_printing()
import elasticity as ela
import plotter as plo
import generategeo as geo
import interfaces as gui
```

```
Define the model and problem parameters
```

```
"""
```

```
gui.wedge_hlp()
c , ietype , order =gui.mesh_gui()
phid , l , nu , E , S = gui.wedge_prs()
phi = ela.radians(phid)
b = l*np.cos(phi)
h = l*np.sin(phi)
```

```
var = geo.wedge(l , phid, 0.1 , ietype)
geo.create_mesh(order , var )
nodes , elements , nn = geo.writefiles(ietype , var)
plo.viewmesh(nodes , elements , True)
coords=np.zeros([nn,2])
U=np.zeros([nn , 2])
Sig=np.zeros([nn , 2])
#
coords[:,0]=nodes[:,1]
coords[:,1]=nodes[:,2]
```

```
Computes the solution
```

```
"""
```

```
for i in range(0,nn):
    x = coords[i,0]
    y = coords[i,1]
    X = x+b
    Y = y-h
    ux , uy , sx , sy = ela.cunia(X , Y , phi , l , nu , E , S)
    U[i , 0] = ux
    U[i , 1] = uy
    Sig[i , 0] = sx
    Sig[i , 1] = sy
```

```
Plot the solution
```

```
"""
```

```
plo.plot_disp(U, nodes, elements , 1 , plt_type="contourf", levels=12 , savefigs = True)
plo.plot_stress(Sig, nodes, elements , 2 , savefigs = True)
```

Importa módulos

Muestra la ayuda gráfica,
captura los parámetros de
la malla y los parámetros
de la solución

Escribe el archivo con la
geometría del modelo.

Ejecuta Gmsh y crea la malla.

Procesa la malla generando los
arreglos de nodos y elements listos
para calcular y visualizar la solución.

Dimensiona arreglos para
almacenar la solución..

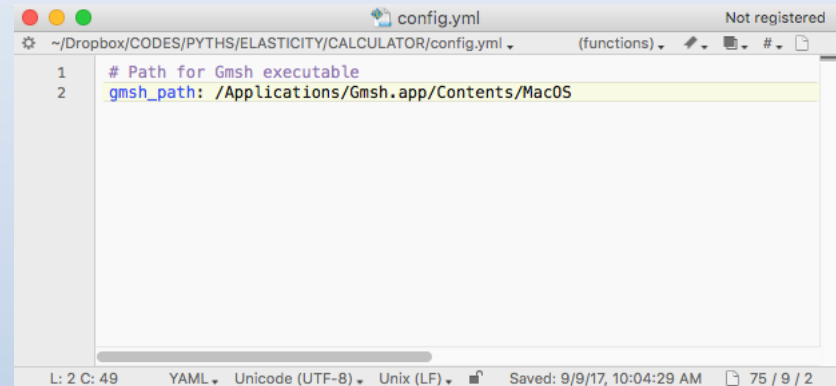
Evalúa la solución un nudo a la vez y
almacena el resultado en los arreglos
de desplazamiento y tensiones

Pasa los arreglos de nodos, elements y
de solución a las rutinas de graficación.

Instalación

Para utilizar el programa una vez descargadas todas las carpetas del mismo, simplemente modifique el archivo **config.yml** que se encuentra en la carpeta **CALCULATOR**. Este debe almacenar la ruta del ejecutable de Gmsh en cada computador particular.

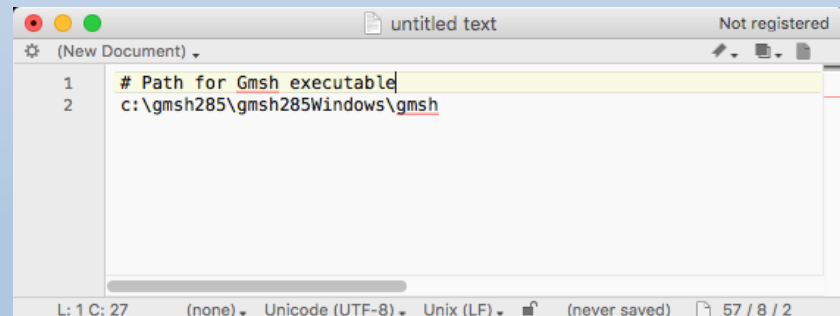
En Mac el archivo sería como este:



```
config.yml
~/Dropbox/CODES/PYTHS/ELASTICITY/CALCULATOR/config.yml
1 # Path for Gmsh executable
2 gmsh_path: /Applications/Gmsh.app/Contents/MacOS
```

The screenshot shows a text editor window titled 'config.yml' with a file path of '~/Dropbox/CODES/PYTHS/ELASTICITY/CALCULATOR/config.yml'. The editor contains two lines of text: a comment '# Path for Gmsh executable' and a line 'gmsh_path: /Applications/Gmsh.app/Contents/MacOS'. The status bar at the bottom indicates 'L: 2 C: 49', 'YAML', 'Unicode (UTF-8)', 'Unix (LF)', and 'Saved: 9/9/17, 10:04:29 AM'.

En Windows el archivo sería como este:



```
untitled text
(New Document)
1 # Path for Gmsh executable
2 c:\gmsh285\gmsh285Windows\gmsh
```

The screenshot shows a text editor window titled 'untitled text' with a '(New Document)' label. The editor contains two lines of text: a comment '# Path for Gmsh executable' and a line 'c:\gmsh285\gmsh285Windows\gmsh'. The status bar at the bottom indicates 'L: 1 C: 27', '(none)', 'Unicode (UTF-8)', 'Unix (LF)', and '(never saved)'.