# Finite Element Analysis Code solids_ISO.py

Juan Gomez

January 22, 2016

## Introduction

Program **solids_ISO** finds the displacement, strain and stress solution for an arbitrary two-dimensional domain discretized into finite elements and subjected to point loads. It has been created for academic purposes and it is part of the teaching material developed for the courses IC0602 Introduction to the Finite Element Methods and IC0285 Computational Modelling at Universidad EAFIT. The code is written in python 2.0 dialect and it is organized in independent modules for pre-processing, assembly and post-processing allowing the user to easily modify it or add features like new elements. In this document we briefly describe its use, initially through a simple example corresponding to a square plate under point loads. In a second problem we show the user how to create a model and visualize results for a more complex problem using the third-party software GMESH.

### Input files

The code requires the domain to be input in the form of text files containing the nodes, elements, loads and material information. These files must reside in the same directory of the python code solids_ISO.py and must have the names eles.txt, nodes.txt, mater.txt and loads.txt. Assume that we want to find the response of the $2 \times 2$ square under unitary vertical point loads shown in fig. 1.
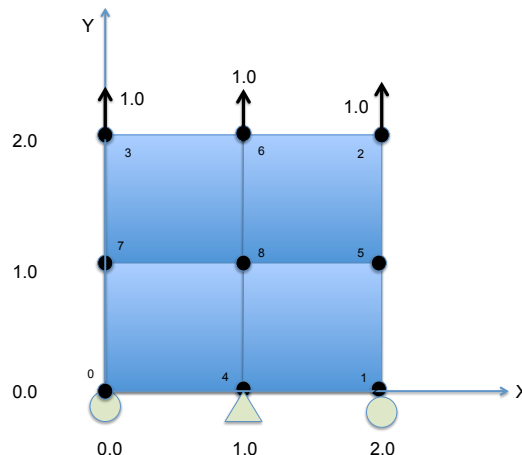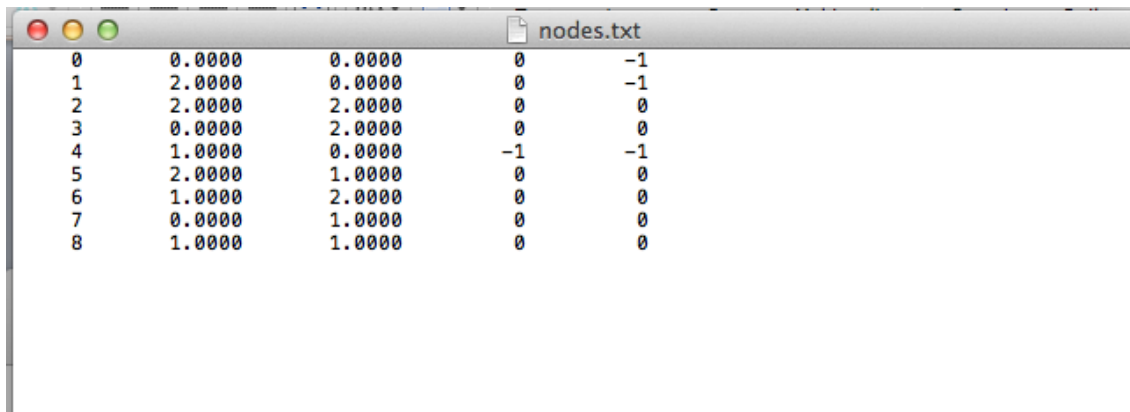


Figure 1: 4-element solid under point load.

The corresponding input files required to conduct the analysis are included. Figure 2 shows the nodes.txt file composed of the following fields:

- Column 1: Nodal identifier (Integer)

- Column 2: x-coordinate (Real)

- Column 3: y-coordinate (Real)

- Column 4: Boundary condition flag along the x-direction (0 free; -1 restrained)

- Column 5: Boundary condition flag along the y-direction (0 free; -1 restrained)

```
nodes.txt
0    0.0000    0.0000     0    -1
1    2.0000    0.0000     0    -1
2    2.0000    2.0000     0     0
3    0.0000    2.0000     0     0
4    1.0000    0.0000    -1    -1
5    2.0000    1.0000     0     0
6    1.0000    2.0000     0     0
7    0.0000    1.0000     0     0
8    1.0000    1.0000     0     0
```
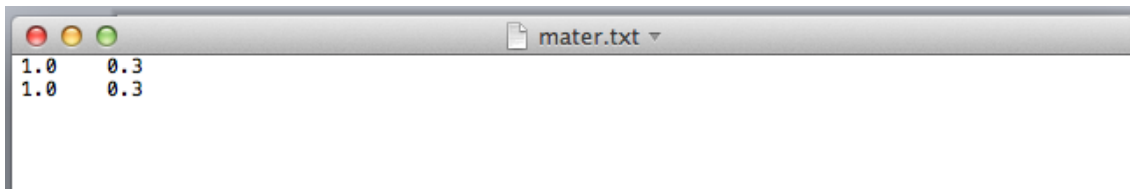
Figure 2: File nodes.txt.

Figure 3 shows the mater.txt file containing the material information. Each line in the file corresponds to a material profile to be assigned to the different elements in the elements file. In this example there are two identical material profiles. Each line in the file is composed of the following fields:

- Column 1: Young's modulus for the current profile (Real)

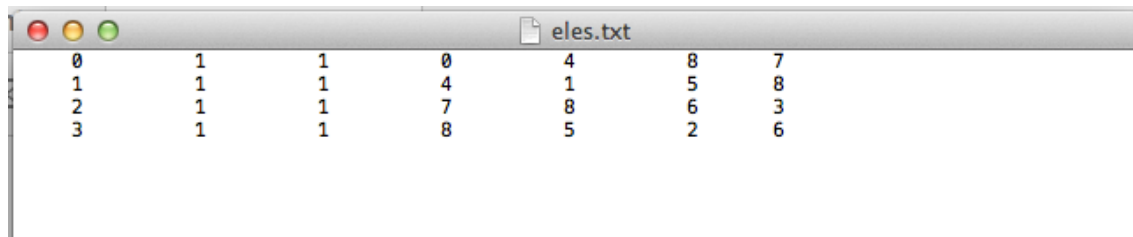- Column 2: Poisson's ratio for the current profile (Real)

```
mater.txt
1.0    0.3
1.0    0.3
```

Figure 3: File mater.txt.

Figure 4 shows the eles.txt file containing the element information. Each line in the file defines the information for a single element and is composed of the following fields:

- Column 1: Element identifier (Integer)

- Column 2: Element type (Integer) (iet=1 4-noded quadrilateral; iet=2 6-noded triangle; iet=3 3-noded triangle)

- Column 3: Material profile for the current element (Integer)

- Column 4 y adicionales: Element connectivities or list of nodes conforming the element (I). The nodes must be listed in counterclockwise direction.
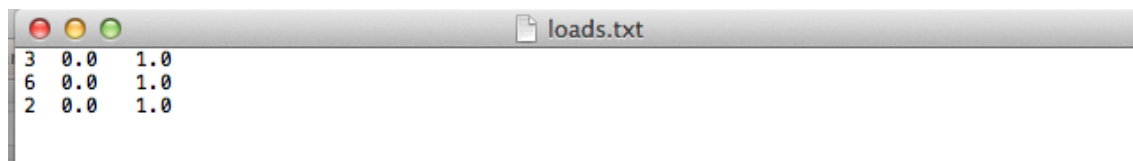


Figure 4: File eles.txt.

Figure 5 shows the loads.txt file containing the point loads information. Each line in the file defines the load information for a single node and is composed of the following fields:

- Column 1: Nodal identifier (Integer)

- Column 2: Load magnitude for the current node along the x-direction (Real)

- Column 3: Load magnitude for the current node along the y-direction (Real)



Figure 5: File loads.txt.

## Executing the program

Once the input files are copied to the main program folder the code can be executed. The user will be prompted by a job name as shwon in fig. 6. This name is used only for post-processing purposes if the user intents to visualize results using the third-party software GMESH. In this case solids_ISO.py uses the job name to locate a file jobname.msh in the MESHUTILS folder.
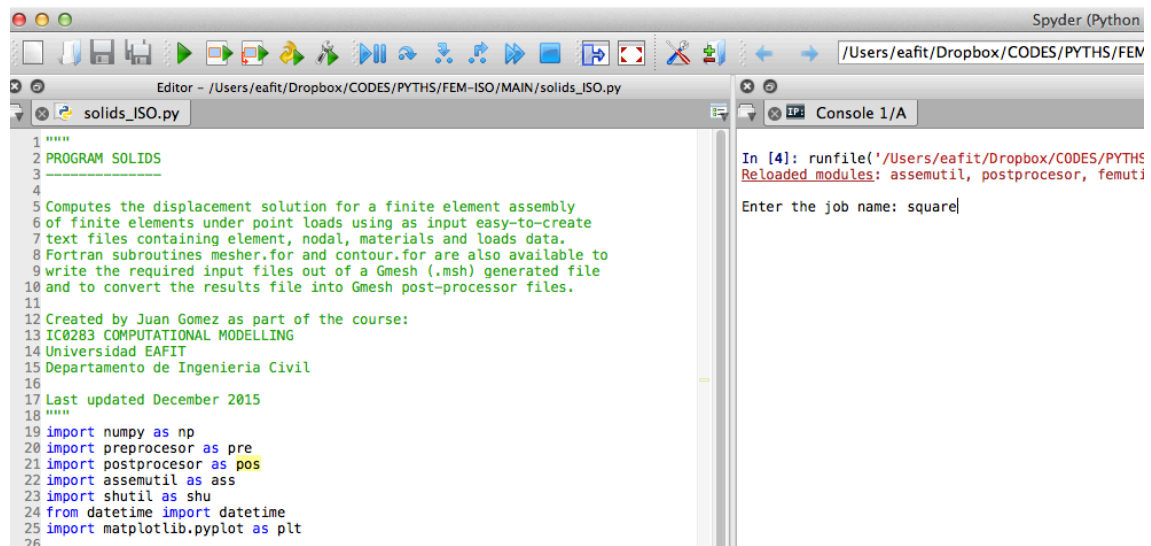
Figure 6: Executing the code.

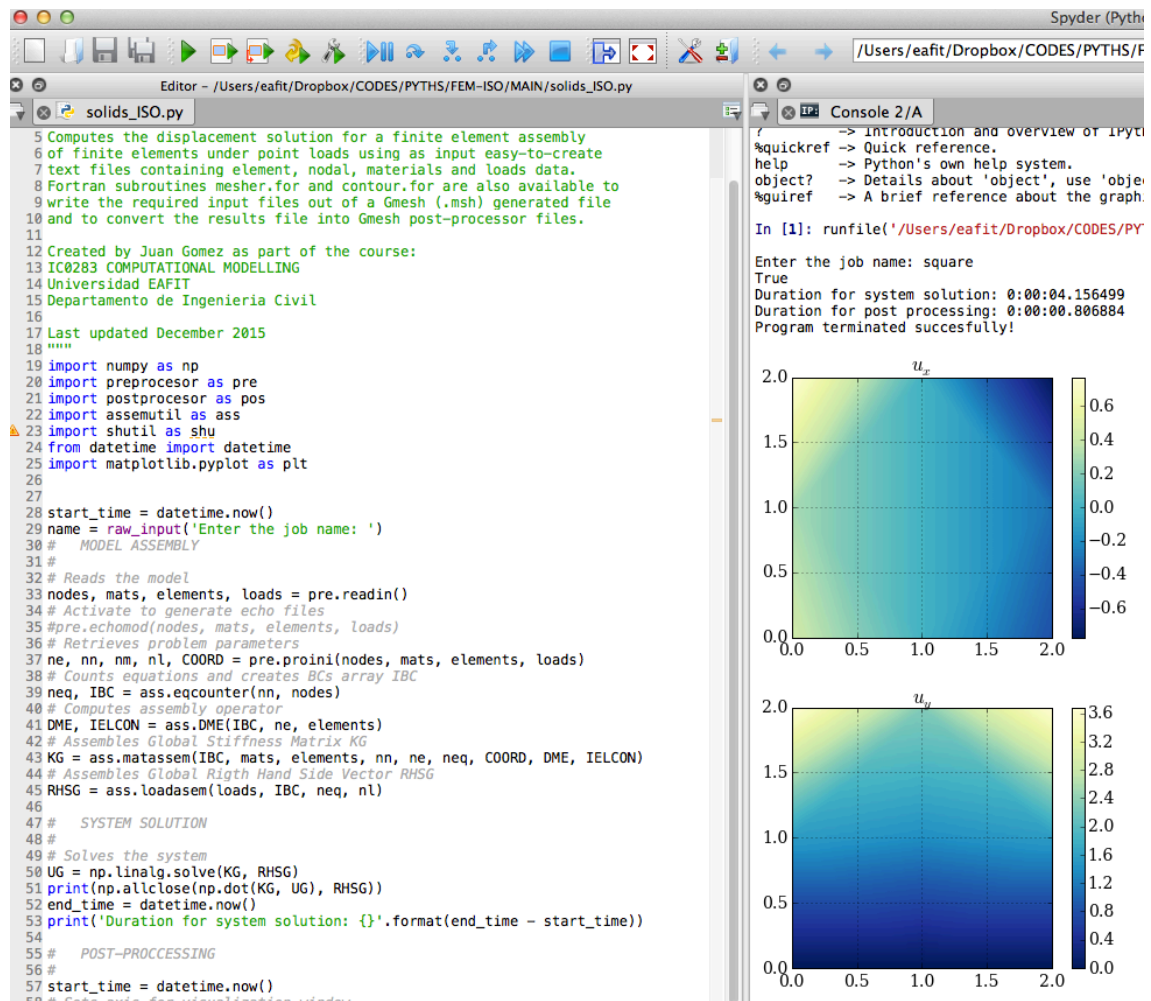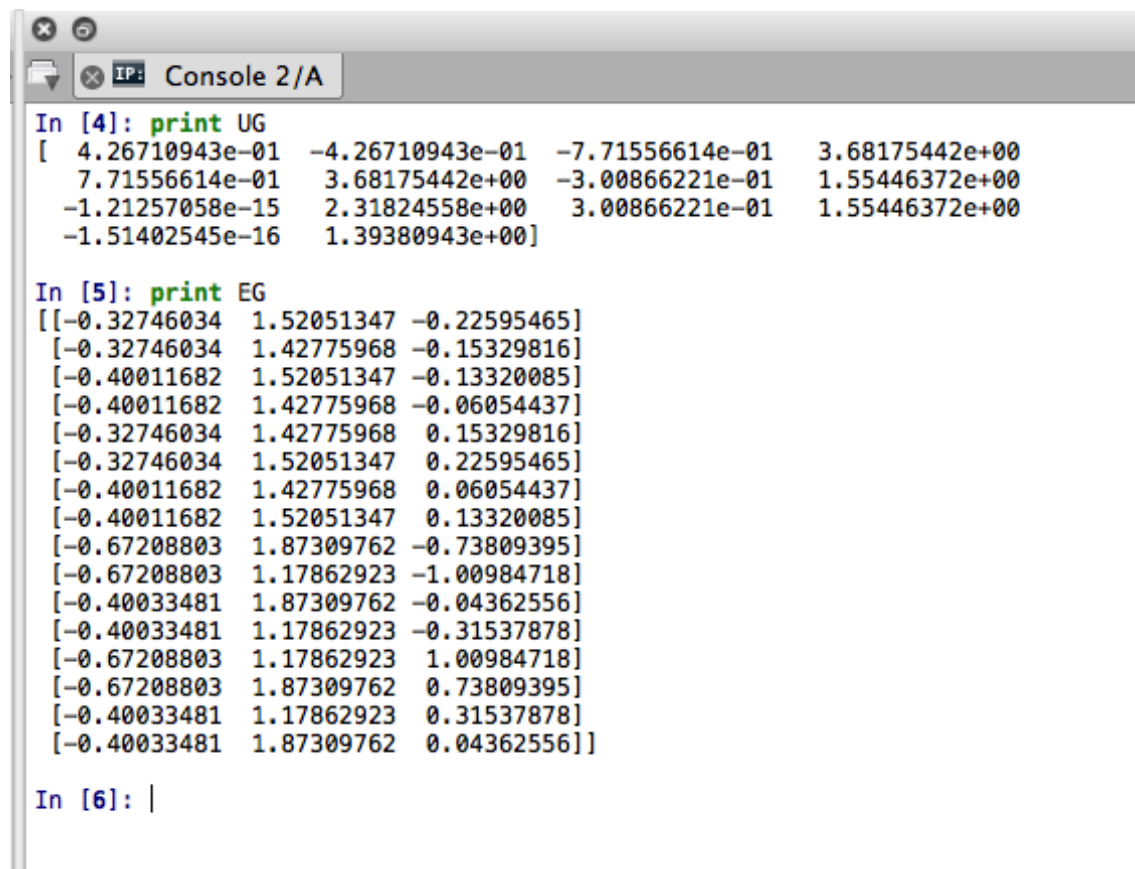Once the code is executed the displacement and stress solution is displayed as shown in fig. 7.

Figure 7: Results output.

The displacement solution for the nodal points and the strain solution at the element integration points is also available to conduct additional postproccesing operations as shown in fig. 8 where we print out the displacement vector and the strain tensor at the integration points.

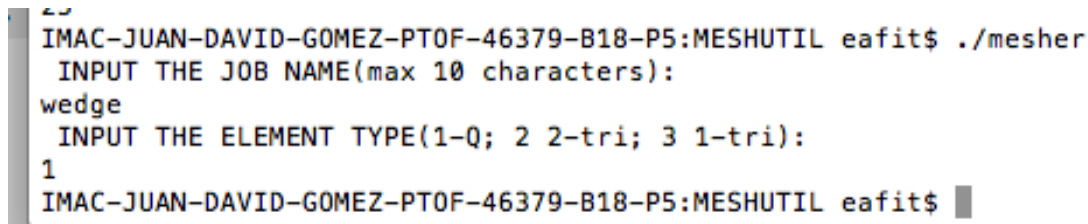Figure 8: Results output.

# Pre and post-processing with GMESH

The input files required to conduct an analysis with solids_ISO.py can be created with additional software avaiable in the folder MESHUTILS. Particularly, the elements and nodes input files can be created with the pre and post-processing third-party software GMESH. GMESH basically discretizes the model geometry into finite elements. It requires the model to be defined by a text file (e.g., square.geo) containing different GMESH geometry creation commands. The user is referred to the GMESH manual. Once the .geo file is available it can be processed by the python code tumesh.py as shown in fig. 9 for the current examnple. Once this code is executed it creates a GMESH readable file (e.g., square.msh). At this point the model can be visualized in GMESH by directly double-clicking the .msh file.

Figure 9: Creating a mesh with GMESH.

In the final step the user needs to execute the fortran code mesher.for available in the MESHUTILS folder. Upon execution this last program creates the files eles.txt and nodes.txt required by solids_ISO.py as shown in fig. 10.



Figure 10: Creating the eles.txt and nodes.txt files with the fortran code mesher.for.

Similarly the results from solids_ISO.py can be visualized with GMESH. For that purpose the user needs to copy the problem .msh file into the MESHUTILS folder. For instance the file square.msh must be copied to the MESHUTILS folder. The program solids_ISO.py creates a results file out.txt into that folder. In the final step the user must execute the fortran code contours.for which uses out.txt to create ready to visualize .msh files.
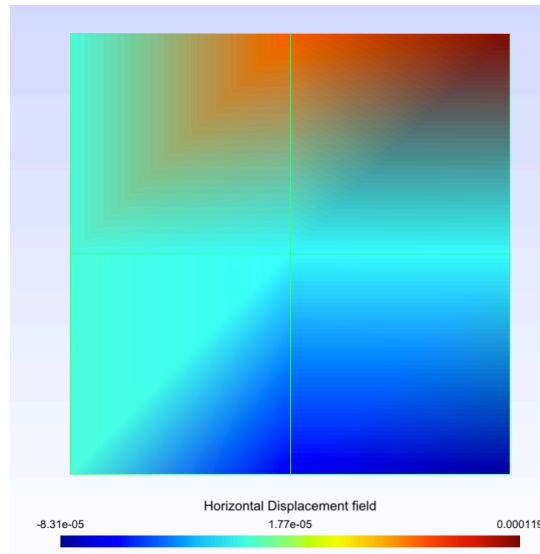
Figure 11: Results visualized with GMESH.

## Sample problem 1: Concrete dam

Consider the concrete dam shown in fig. 12. We will create the finite element model using GMESH. In the first step we create the geometric file dam.geo (see fig. 13). In this case the dam is to be meshed with linear triangular elements which is specified simply by commenting out the line reading "Recombine surface 1" in the file dam.geo.
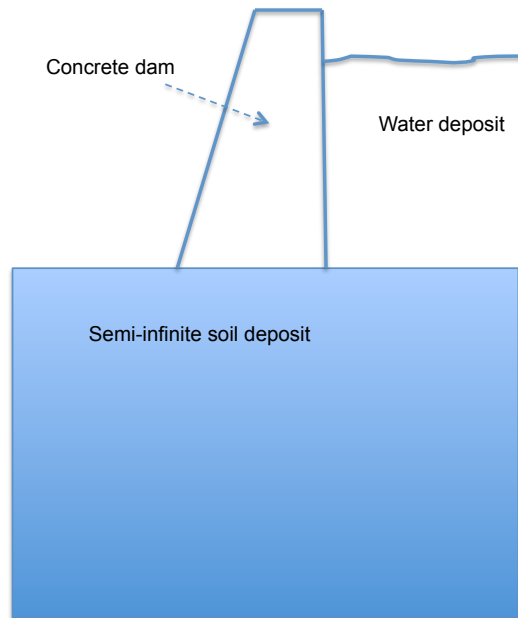


Figure 12: Concrete dam under water pressure.

```
// Input .geo for dam
// author: Juan Gomez

c = 5.0;                                          // for size elements

// Define vertex points
Point(1) = {-50.0,-150.0, 0, c};                  // {x,y,z, size}
Point(2) = {100.0,-150.0, 0, c};
Point(3) = {100.0, 0.000, 0, c};
Point(4) = {50.00, 0.000, 0, c};
Point(5) = {0.000, 0.000, 0, c};
Point(6) = {-50.0, 0.000, 0, c};
Point(7) = { 50.0, 100.0, 0, c};
Point(8) = { 25.0, 100.0, 0, c};


// Define boundary lines
Line(1) = {1, 2};                                 // {Initial_point, end_point}
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 5};
Line(5) = {5, 6};
Line(6) = {6, 1};
Line(7) = {4, 7};
Line(8) = {7, 8};
Line(9) = {8, 5};

// Joint Lines
Line Loop(1) = {1, 2, 3, 4, 5, 6};                // {Id_line1,id_line2, ... }
Line Loop(2) = {-4, 7, 8, 9};

// surface for mesh                               // {Id_Loop}
Plane Surface(1) = {1};
Plane Surface(2) = {2};

// For Mesh 4 nodes
//Recombine Surface {1};                          // {Id_Surface}

// "Structure" mesh
//Transfinite Surface {1,2};                      // {Id_Surface}

// Physical surface. Two material
Physical Surface(100) = {1};
Physical Surface(200) = {2};

//Physical line. Boundary
//Physical Line(1000) = {1};
//Physical Line(2000) = {2};
//Physical Line(3000) = {4};
//Physical Line(4000) = {6};
//Physical Line(5000) = {7};
```

Figure 13: Geometric file to create the GMESH files.

In the next step we create the file dam.msh file containing the actual GMESH model. This is accomplished using the python code tumesh.py as shown in fig. 14. The "-order 1" command indicates that the geometry is to be meshed with linear elements.

```python
1 # -*- coding: utf-8 -*-
2 """
3 Created on Nov 26 16:45:00 2015
4
5 @author:
6
7 Ruttine for mesh by gmesh
8 """
9 from __future__ import division
10 import os
11
12 os.system ('/Applications/Gmsh.app/Contents/MacOS/gmsh dam.geo -2 -order 1')
13 print('End Proccess')
14
```

Figure 14: Execution of the code tumesh.py to create the model dam.msh.

At this point the actual mesh can be visualized using GMESH by double-clicking the file dam.mesh. The mesh should look like the one shown in fig. 15.
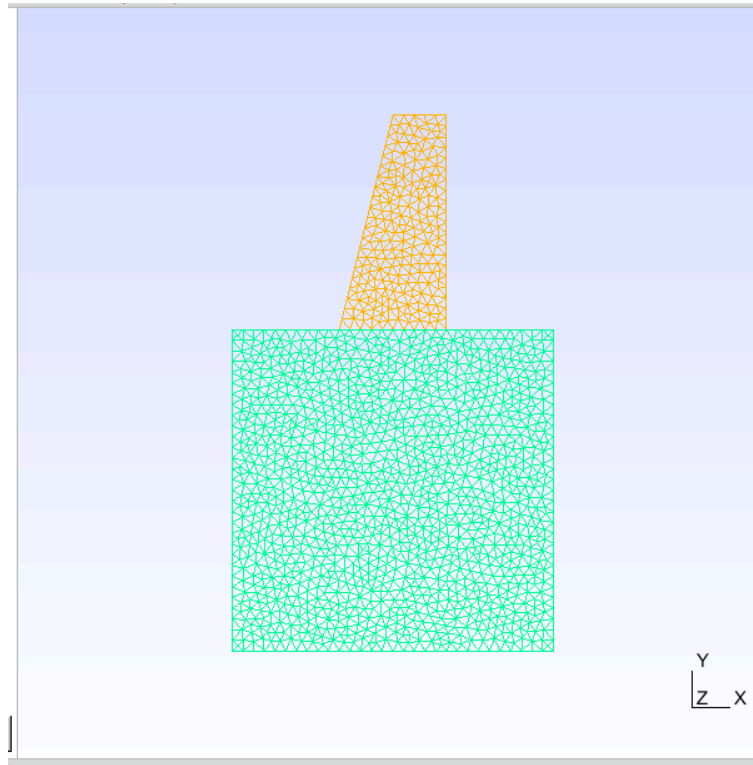
Figure 15: Finite element mesh created and visualized in GMESH.

In the final step we use the fortran code mesher.for to create the eles.txt and nodes.txt files out of the dam.msh file. The execution of the code mesher.for is shown in fig. 16.



Figure 16: Execution of the mesher.for file to create the eles.txt and nodes.txt files.

Once the code is executed and the elements and nodal files have been created these must be copied to the main program folder in order to perform the finite element analysis with solids_ISO.py (see fig. 17). It is important to use as job name the same file name used for the .msh file previously used and available in the MESHUTIL folder.

```
1 """
2 PROGRAM SOLIDS
3 --------------
4
5 Computes the displacement solution for a finite element assembly
6 of finite elements under point loads using as input easy-to-create
7 text files containing element, nodal, materials and loads data.
8 Fortran subroutines mesher.for and contour.for are also available to
9 write the required input files out of a Gmesh (.msh) generated file
0 and to convert the results file into Gmesh post-processor files.
1
2 Created by Juan Gomez as part of the course:
3 IC0283 COMPUTATIONAL MODELLING
4 Universidad EAFIT
5 Departamento de Ingenieria Civil
6
7 Last updated December 2015
8 """
9 import numpy as np
0 import preprocesor as pre
1 import postprocesor as pos
2 import assemutil as ass
3 import shutil as shu
4 from datetime import datetime
5 import matplotlib.pyplot as plt
6
7
```

```
Python 2.7.11 |Anaconda 2.3.0 (x86_64)| (defa
Type "copyright", "credits" or "license" for

IPython 4.0.0 -- An enhanced Interactive Pyth
?         -> Introduction and overview of IPy
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'obj
%guiref   -> A brief reference about the grap

In [1]: runfile('/Users/eafit/Dropbox/CODES/F

Enter the job name: dam
```

Figure 17: Execution of the solids_ISO.py code to conduct the finite element analysis.

Upon execution solids_ISO.py display horizontal and vertical displacements contours (see fig. 18 and at the same time creates copies of the file dam.msh in order to use them during the GMESH post-processing step. In particular solids_ISO.py creates the files damH.msh (for the horizontal displacements), damV.msh (for the vertical displacements) and damF.msh (for the full displacement field).
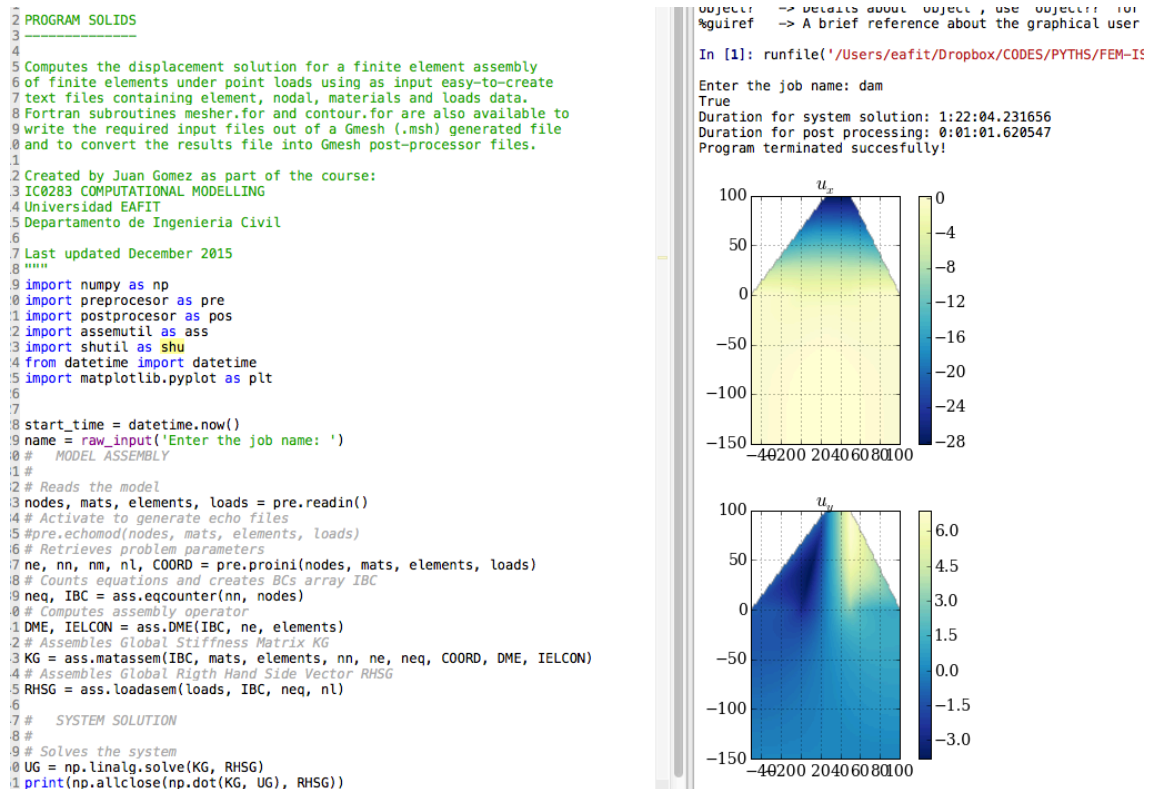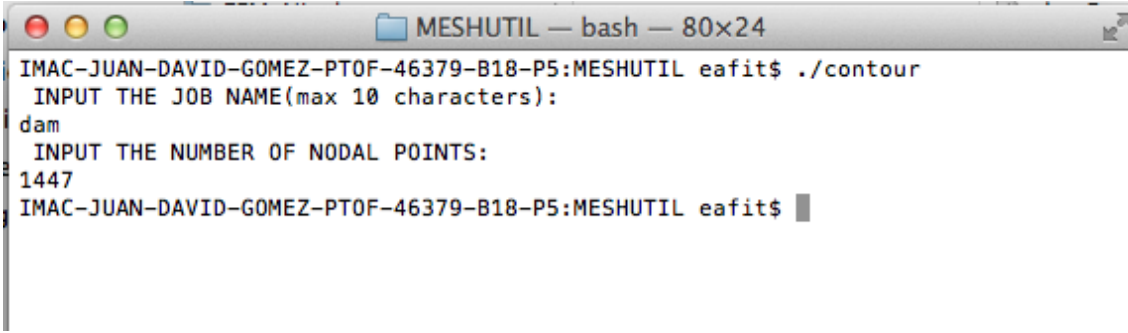


Figure 18: Results after conducting the analysis with solids_ISO.py.

As a final step we modify the files damH.msh, damV.msh and damF.msh in order to visualize the results in GMESH. For that purpose we execute the fortran code contour.for. Upon execution

11

in the terminal window contour.for will prompt the user for the number of nodes in the model. This value can be obtained from the python console using the command "print nn" which in this case will return the value 1447 corresponding to the number of nodes in the model (see fig. 19). The files damH.msh, damV.msh and damF.msh can now be directly double-clicked to visualize the horizontal, vertical and full displacement contours over the dam domain (see fig. 20)



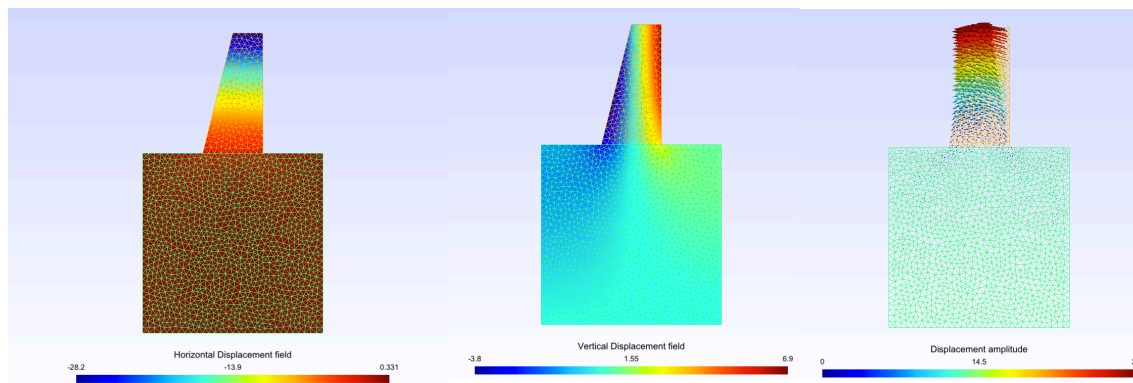Figure 19: Execution of contour.for in order to generate GMESH results files.



Figure 20: Results for the dam problem visualized with GMESH.