ME608 Spring 2016 - Homework 1

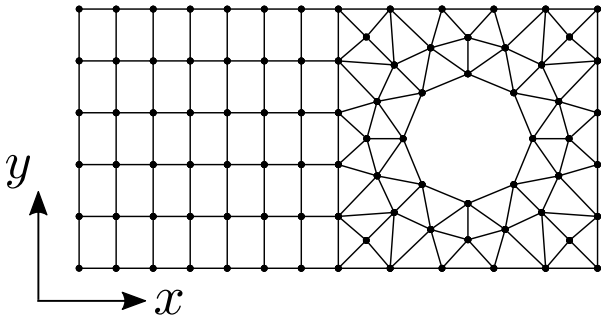
Unstructured Meshes & First Spatial Discretization

(Due September 18, 2017)

Please prepare this homework in the folder `/homework1/` of your git repository and submit it following the instructions in the last section of Homework 0. The main script should be named `homework.py` and placed in the `/homework1/code/` folder. The submitted code needs to run and create all of the required plots in the folder `/homework1/report/figures/` at once. The use of LaTeX for your report is strongly recommended (but not required) and you can start with the template provided to you. Discussions and sharing of ideas are encouraged but individually prepared submissions (codes, figures, written reports, etc.) are required. A plagiarism detection algorithm will be run against all codes submitted. Do NOT include in your git repository files that are not required for homework submission (e.g. the syllabus, zip files with python libraries, other PDFs, sample python sessions etc). Follow exactly the suggested folder structure of your homework git repository as all of the grading tools on the instructor's end are scripted. Points will be deducted from late submissions at a rate of 20% of the overall homework value per day late.

Problem 1

Define a computational domain in ANSYS ICEM CFD with a level of geometrical complexity comparable to the one shown in figure (don't shoot yourself in the foot with something too complicated). Your domain should not be simply connected (i.e. there needs to be at least one *hole*) and your mesh should include both `QUAD` and `TRI` elements. Think of a heat transfer problem with one or multiple boundaries labeled as `HOT` and other as `COLD`, as done in class. You are NOT allowed to use the same sample mesh/geometry provided to you, please come up with something different, possibly of interest to you.



For the same geometry, prepare four different levels of mesh resolution (call them A,B,C and D), spanning *at least* 2 orders of magnitude in terms of number of cells, or control volumes, starting with approximately `ncv=100` cells. Display the four computational grids as shown in the figure above.

[20%]

Display contour plots of mesh quality metrics via the built-in functionality in ANSYS ICEM CFD (take screenshots). Very briefly discuss what you can infer from this analysis.

[10%]

Problem 2

By explicitly looping over the connectivity arrays, and on a grid of intermediate size created in Problem 1, solve the *poor-man* unsteady heat transfer problem defined by the following steps:

1. Allocate an array of nodal temperature values `Tn` with total length `nno`, initialized with zeroes everywhere. Assign values of 300K and 500K on nodes belonging to the `COLD` and `HOT` boundaries (or zones) respectively;
2. Allocate an array of cell-center temperature values `Tcv` with total length `ncv` initialized with zeroes everywhere;
3. Loop over each control volume and store the arithmetic average of the surrounding nodal values into the corresponding cell-center temperature value;
4. Loop over each *internal* node and update its value with the arithmetic average of the surrounding cells;
5. Repeat steps 3 and 4 until convergence.

Plot the temperature field at different stages of its pseudo-temporal evolution (see Appendix).

[20%]

In the steps above, two linear operators have been used without being explicitly created. These are averaging operators from nodal values to cell centers, $\mathcal{A}_n^{cv}$, and from cell centers to internal nodes, $\mathcal{A}_{cv}^{n,int}$, which should be used in the following way in your Python code:

```
Tcv = An2cv.dot(Tn)
Tn_int = Acv2n_int.dot(Tcv)
```

Construct such operators in the form of a sparse matrices and use the `spy` function of `pylab` to plot them for all available grid sizes. [15%]

Bonus: In a table, report the speed-up associated with using the explicitly defined linear operators versus looping over connectivity arrays for various grid sizes. [+2%]

Problem 3

Analytically define a two-component two-dimensional divergence-free vector field

$$\vec{V} = (u(x,y), v(x,y)) \tag{1}$$

and sample it on all nodal locations of the computational grids created in Problem 1. Let `Un` and `Vn` be the arrays containing the sampled nodal values of $u(x,y)$ and $v(x,y)$. Plot the sampled field on a coarse grid using the `quiver` function. Do not use "simple" fields, such as $\vec{V} = (x, -y)$ or $\vec{V} = (\sin(x), -y)$, they might keep numerical issues hidden from you. [10%]

By looping over the connectivity arrays, numerically evaluate the divergence of (1) for each cell by applying the Gauss theorem: 1) average the nodal values `Un` and `Vn` at the center location of each face (given by `xy_fa[]`); 2) take the dot product with the face normal (to be oriented outwards with respect to the cell, careful here...) and multiply by the length of the face; 3) sum over all faces of the cell and divide the result by the cell area. Plot a flooded color contour of the value of the numerically estimated divergence for each cell. Plot in log-log scale the RMS of the discretization error (i.e. you need to take the square-root of the sum of the squares of the numerical divergences, where the sum of the squares needs to be weighted by the square of the corresponding cell area) versus number of total control volumes for each grid. What is the order of accuracy of this spatial discretization scheme? [10%]

Build two sparse linear operators, `Dx_n2cv` and `Dy_n2cv`, implementing the steps above in algebraic form and such that:

```
DIV =  Dx_n2cv.dot(Un)+Dy_n2cv.dot(Vn).
```

Plot such operators with the `spy` function of `pylab`. Building these operators is NOT trivial but it is required for your future unstructured Navier-Stokes solver. You can build them by imaging to calculate the divergence of the vector fields $\vec{V}_u = (u(x,y), 0)$ and $\vec{V}_v = (0, v(x,y))$ separately. [15%]

Bonus: Keeping the velocity values at the nodes and defining pressure at the cell centers, discretize the following equations

$$\frac{d}{dt}p' = -\rho_0 a_0^2 \frac{\partial}{\partial x_j} u_j{}' \tag{2a}$$

$$\frac{d}{dt}u_i{}' = -\frac{1}{\rho_0}\frac{\partial}{\partial x_i} p' \tag{2b}$$

on an unstructured grid and domain of your choice simply imposing zero velocity at the boundaries (with no boundary condition needed for pressure... isn't that interesting?). The subscript '0' indicates base state conditions and the prime the fluctuations; you can use values relevant for air, where a_0 and ρ_0 are the base speed of sound and density. Note that on the right-hand-side of (2a) has a divergence operator that goes from nodal values to cell centers and (2b) has a gradient operator that goes from cell centers to nodes (not trivial to create). Initialize your time-resolved simulation with a Gaussian bump somewhere in the center of the domain with $p'_{\max} = 10^{-4}\rho_0\, a_0^2$ and zero velocity everywhere. [+5%]

Appendix

Plotting unstructured data

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata

x = ... # x coordinates of data
y = ... # y coordinates of data
phi = .. # values of data

# define regular grid spatially covering input data
n = 50
xg = np.linspace(x.min(),x.max(),n)
yg = np.linspace(y.min(),y.max(),n)
X,Y = np.meshgrid(xg,yg)

# interpolate Z values on defined grid
Z = griddata(np.vstack((x.flatten(),y.flatten())).T, \
  np.vstack(phi.flatten()),(X,Y),method='cubic').reshape(X.shape)
# mask nan values, so they will not appear on plot
Zm = np.ma.masked_where(np.isnan(Z),Z)

# plot
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')
ax.pcolormesh(X,Y,Zm,shading='gouraud')
ax.set_title('Using pcolormesh')
```