MAE 6500 Potential Flow - Fall 2022
Panel Method Project Components

Cory Goates

P 1. Download the panel method starter files from https://github.com/corygoates/python-panel-method-starter. This repository contains all the Python files you will need to get started (mostly geometry input and output) as well as some example meshes for you to play with.

If you're interested in experimenting with your own meshes, NASA's Open Vehicle Sketch Pad (OpenVSP) is a good tool for generating meshes. You can download it from http://openvsp.org/download.php. It also has a panel method built in, so you can compare solutions.

P 2. As part of the `Panel` class defined in `panels.py`, write member functions `calc_area`, `calc_normal`, `calc_edge_tangents`, and `calc_edge_outward_normal` that calculate the panel area, panel outward normal vector, edge tangent vectors, and edge outward normal vectors, respectively, and store these as member variables of `Panel`. Add calls to these functions to the initializer for `Panel`. Remember that Python syntax for initialization and member functions is

```
class Panel:

    def __init__(self):
        # Do whatever initializations
        self.calc_area()
        self.calc_normal()

    def calc_area(self):
        self.A = # Some calcs

    def calc_normal(self):
        self.n = # Some other calcs
```

In the output section of your code, add the panel area and outward normal vector as outputs to the VTK file. Run a few meshes and verify these functions are working properly, particularly that your normal vector is pointing outward.

Also as part of the `Panel` class, write a member function `calc_centroid` which calculates the centroid of the panel and stores it as a member variable. Add a call to this function to the initializer for `Panel`.

P 3. In the `Panel` initializer, create a 3-by-3 matrix called `A_g_to_l`. This will be an orthogonal transformation matrix which can transform between global and local panel coordinates. The last row of `A_g_to_l` is the panel normal vector. For the first row, pick an edge tangent vector. Compute the middle row to complete a right-handed coordinate system.

To convert from global to local coordinates, we may use the formula

$$\mathbf{P}' = A_{g\to l}\left(\mathbf{P} - \mathbf{P}_{cent}\right) \tag{1}$$

where $\mathbf{P}_{cent}$ is the panel centroid. Throughout, we will use $'$ to denote the local coordinate system.

P 4. As part of the `Panel` class, write member functions `calc_doublet_potential` and `calc_doublet_velocity` which calculate the potential and velocity vector induced (respectively) at an arbitrary point in space, $\mathbf{P}$, by the panel

assuming a unit constant doublet strength (i.e. $\mu \equiv 1$ everywhere on the panel. For each panel, we say that edge i begins as vertex $\mathbf{Q}_i$ and ends at vertex $\mathbf{Q}_{i+1}$. For incompressible flow, the potential induced by a constant-strength doublet panel is given by

$$\phi_\mu = \frac{\mu}{4\pi} sign(h) \sum_{i=1}^{N_{edges}} \tan^{-1}\left(\frac{a_i(l_{2_i}c_{1_i} - l_{1_i}c_{2_i})}{c_{1_i}c_{2_i} + a_i^2 l_{1_i}l_{2_i}}\right) \tag{2}$$

and the velocity induced by a constant-strength doublet panel is given by

$$\mathbf{u}_\mu = \frac{\mu}{4\pi} \sum_{i=1}^{N_{edges}} \frac{(r_i + r_{i+1})(\mathbf{r}_i \times \mathbf{r}_{i+1})}{r_i r_{i+1}(r_i r_{i+1} + \mathbf{r}_i \cdot \mathbf{r}_{i+1})} \tag{3}$$

where

$$\mathbf{r}_i = \mathbf{Q}_i - \mathbf{P} \tag{4}$$

$$r_i = \|r_i\| \tag{5}$$

$$c_{1_i} = g_i^2 + |h| r_i \tag{6}$$

$$c_{2_i} = g_i^2 + |h| r_{i+1} \tag{7}$$

$$a_i = \mathbf{r}_i \cdot \boldsymbol{\nu}_i \tag{8}$$

$$l_{1_i} = \mathbf{r}_i \cdot \mathbf{t}_i \tag{9}$$

$$l_{2_i} = \mathbf{r}_{i+1} \cdot \mathbf{t}_i \tag{10}$$

$$h = -\mathbf{r}_i \cdot \mathbf{n} \tag{11}$$

$$g_i^2 = a_i^2 + h^2 \tag{12}$$

and $\mathbf{t}_i$ is the unit tangent vector for edge i and $\boldsymbol{\nu}_i$ is the unit outward normal vector for edge i, which were already calculated in your panel initialization.

For debugging, Table 1 lists induced velocities and potentials from a panel with unit doublet strength at various points. To instantiate a single panel, use

```
import numpy as np
from panels import Panel
my_test_panel = Panel(np.array([0.0, 0.0, 0.0],
                      np.array([1.0, 0.0, 0.0],
                      np.array([0.0, 1.0, 0.0], 0, 0, 0)
```

P 5. Since the doublet strengths for each panel are not known a priori, we must set up a system of equations which will allow us to determine these strengths. Everywhere, we wish to enforce the zero-normal-flow boundary condition

$$\mathbf{u} \cdot \mathbf{n}_i = 0 \tag{13}$$

on each panel i. Since $\mathbf{u}$ is a combination of the freestream velocity and the velocities induced by each panel, we may write this as

Table 1 Induced potentials and velocities from a unit-strength doublet panel with corners at $(0,0,0)$, $(1,0,0)$, and $(0,1,0)$.

Point	ϕ	$\mathbf{v}$
$(0,0,1)$	2.7043361992348181E-002	$(-1.87565899e-02, -1.87565899e-02, 3.75131798e-02)$
$(0,0,2)$	8.8602364006150035E-003	$(-1.97711818e-03, -1.97711818e-03, 7.90847270e-03)$
$(1,1,1)$	1.4623304674318490E-002	$(1.45411425e-02, 1.45411425e-02, 8.43089478e-03)$
$(2,2,2)$	2.6771014779820080E-003	$(1.38680762e-03, 1.38680762e-03, 3.47069909e-04)$
$(0,0,-1)$	-2.7043361992348181E-002	$(1.87565899e-021.87565899e-023.75131798e-02)$
$(0.5,0.5,0)$	0.0	$(0.00000000e+00, 0.00000000e+00, 4.50158158e-01)$

$$\sum_{j=1}^{N} \mu_j \mathbf{u}_{ij} \cdot \mathbf{n}_i = -\mathbf{u}_\infty \cdot \mathbf{n}_i \tag{14}$$

where $\mathbf{u}_{ij}$ is the velocity induced by panel j on some point on the surface of panel i (typically the centroid). Applying this equation over N panels, we obtain a linear system of equations of the form

$$\begin{bmatrix} \mathbf{u}_{11}\cdot\mathbf{n}_1 & \mathbf{u}_{12}\cdot\mathbf{n}_1 & \dots & \mathbf{u}_{1N}\cdot\mathbf{n}_1 \\ \mathbf{u}_{21}\cdot\mathbf{n}_2 & \mathbf{u}_{22}\cdot\mathbf{n}_2 & \dots & \mathbf{u}_{2N}\cdot\mathbf{n}_2 \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{u}_{N1}\cdot\mathbf{n}_N & \mathbf{u}_{N2}\cdot\mathbf{n}_N & \dots & \mathbf{u}_{NN}\cdot\mathbf{n}_N \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{bmatrix} = \begin{bmatrix} -\mathbf{u}_\infty\cdot\mathbf{n}_1 \\ -\mathbf{u}_\infty\cdot\mathbf{n}_2 \\ \vdots \\ -\mathbf{u}_\infty\cdot\mathbf{n}_N \end{bmatrix} \tag{15}$$

Implement the formation of the above equation in your code and solve it using `numpy.linalg.solve`. You should get a singular matrix error. Why is this?

To get a solution, use `numpy.linalg.lstsq`. Write the doublet strength for each panel out to your results file. Pull up some of the results in Paraview. What do you observe about the doublet strength distribution? It should be smooth. If you're seeing oscillations, go back and check your equations.

P 6. For the current formulation, the velocity on the surface of each panel is given by

$$\mathbf{u} = \nabla\mu \tag{16}$$

Hence, it is necessary to estimate the gradient of doublet strength on the surface of the body. This may seem strange as for each panel the doublet strength is constant. However, this is meant to be a discrete approximation of a smooth distribution of doublet strength across the surface of the body. Thus, we estimate the doublet gradient as if the doublet strength were smoothly distributed from panel centroid to panel centroid. To do this, we assume the doublet strength is given in a region local to the centroid of panel i by a linear distribution

$$\mu - \mu_i = c_1\Delta x' + c_2\Delta y' \tag{17}$$

where $'$ denotes local panel coordinates having the panel centroid as the origin. Since the velocity normal to the panel is zero, we are only interested in calculating the velocity tangent to the panel. Hence, there are no z' terms in Eq. (17). The gradient in the plane of the panel is then simply given by

$$\nabla'\mu = [c_1, c_2, 0.0]^T \tag{18}$$

This is readily transformed back to global coordinates in order to calculate velocities on the surface using

$$\mathbf{u} = A_{g\to l}^T \nabla'\mu \tag{19}$$

In order to calculate c_1 and c_2, a least-squares approach is taken. Applying Eq. (17) to the centroids of a given set of M panels neighboring panel i, we obtain

$$\begin{bmatrix} \Delta x'_1 & \Delta y'_1 \\ \Delta x'_2 & \Delta y'_2 \\ \vdots & \vdots \\ \Delta x'_M & \Delta y'_M \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \mu_1 - \mu_i \\ \mu_2 - \mu_i \\ \vdots \\ \mu_M - \mu_i \end{bmatrix} \tag{20}$$

As long as $M \geq 2$, this system may be solved, either directly or using a least-squares approach (`numpy.linalg.lstsq`). The resulting coefficients are used to determine the gradient.

In the starter code, the panels adjacent to each panel are already calculated, and these neighbors are stored in the vector `Panel.neighbors`. Implement the above equations to be able to estimate the velocity vector on each panel. Add the calculated velocity to your output file.

P 7. The incompressible pressure coefficient on each panel is given by

$$C_{P_i} = 1 - \frac{\|\mathbf{u}_i\|}{\|\mathbf{u}_\infty\|} \tag{21}$$

Implement this calculation for each panel in your code and add the pressure coefficient to your output file. Following the tutorial video, plot the pressure coefficient distribution over the regularly-paneled sphere. It should line up closely with the analytic result.

P 8. Add to your code the ability to calculate total force coefficients for an arbitrary configuration. The force contribution of each panel is given by

$$\Delta \mathbf{F}_i = C_{P_i} \mathbf{n}_i \frac{A_i}{A_{ref}} \tag{22}$$

Output the total force coefficients to the terminal. Run a few meshes and see what forces are generated. What do you notice?

P 9. We will now add to our panel method to remove the need for least-squares solution of the matrix equation. On the surface of the configuration, we want to enforce the boundary condition

$$\mathbf{u} \cdot \mathbf{n} = 0 \tag{23}$$

which we may write as

$$\nabla \Phi \cdot \mathbf{n} = 0 \tag{24}$$

Recall that source panels induce a jump in the normal derivative of the velocity potential. Since we are not using source panels, we can write the boundary condition in terms of the velocity potential inside the configuration

$$\nabla \Phi_i \cdot \mathbf{n} = 0 \tag{25}$$

If we choose $\Phi_i \equiv c$, where c is some constant, then our boundary condition will be satisfied. Let us choose $\Phi_i \equiv 0$, which leads to the necessary relation

$$\phi_i = -\phi_\infty \tag{26}$$

where the freestream velocity potential, ϕ_∞, is given by (verify for yourself this is correct)

$$\phi_\infty = \mathbf{P} \cdot \mathbf{u}_\infty \tag{27}$$

Hence, if we can enforce

$$\phi_i = -\mathbf{P} \cdot \mathbf{u}_\infty \tag{28}$$

then the boundary condition will be satisfied.

To enforce the above condition on the inner potential, we must place a number of control points inside the configuration equal to the number of unknown doublet strengths. In this case, this is the number of panels. We may thus place one control point per panel. The simplest way to do this is to place a control point just below the centroid of each panel in the opposite direction of the panel outward normal, as in

$$\mathbf{P}_{cp_i} = \mathbf{P}_{cent_i} - l\mathbf{n}_i \tag{29}$$

where l is a user-specified offset distance (call this parameter `control_point_offset` in your input file). Assemble the linear system of equations as before, except now the AIC matrix now consists of the potential induced by each panel at each control point and the right-hand vector is the negative of the freestream potential at each control point, as in

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N1} & \phi_{N2} & \dots & \phi_{NN} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{bmatrix} = \begin{bmatrix} -\mathbf{u}_\infty \cdot \mathbf{P}_{cp_1} \\ -\mathbf{u}_\infty \cdot \mathbf{P}_{cp_2} \\ \vdots \\ -\mathbf{u}_\infty \cdot \mathbf{P}_{cp_N} \end{bmatrix} \tag{30}$$

where ϕ_{ij} is the potential induced by panel j on control point i. Solve the equation using `numpy.linalg.solve`. Why are we no longer getting a singular matrix error? How do the computed doublet strengths, velocities, and pressures compare to the Neumann formulation you already implemented?

Modify your code so that the user may choose either boundary condition formulation. Call the previous formulation `neumann` and this one `dirichlet` and have the key in the input be called `formulation`.

P 10. Extra credit: We will add a third formulation to your panel method called the Morino formulation. It is a Dirichlet formulation, but, unlike our previous Dirichlet formulation, it incorporates sources. As such, you will need to add a member function to `Panel` called `calc_source_potential` which returns the potential induced by source panel of unit strength at an arbitrary point in space. For incompressible flow, the potential induced by a constant-strength source panel is given by

$$\phi_\sigma = \frac{\sigma}{4\pi}\left[-|h| \sum_{i=1}^{N_{edges}} \tan^{-1}\left(\frac{a_i(l_{2_i}c_i - l_{1_i}c_{i+1})}{c_i c_{i+1} + a_i^2 l_{1_i} l_{2_i}}\right) + \sum_{i=1}^{N_{edges}} a_i F_i\right] \tag{31}$$

where

$$F_i = \begin{cases} \log\left(\frac{(r_i - l_{1_i})(r_{i+1} + l_{2_i})}{g_i^2}\right), & l_{1_i} l_{2_i} < 0 \\ sign(l_{1_i}) \log\left(\frac{r_{i+1} + |l_{2_i}|}{r_i + |l_{1_i}|}\right), & \text{otherwise} \end{cases} \tag{32}$$

For the Morino formulation, we may calculate the source strength on each panel analytically using

$$\sigma_i = -\mathbf{u}_\infty \cdot \mathbf{n}_i \tag{33}$$

As before, we then place a number of control points inside the body equal to the number of unknown doublet strengths. At each of these control points, we want the perturbation potential to be zero; that is, the sources and doublets should cancel each other out inside the body, leaving only the freestream potential. Thus, for a given control point k, we have the condition

$$\sum_{i=1}^{N} \mu_i \phi_{\mu_{ki}} + \sum_{j=1}^{N} \sigma_j \phi_{\sigma_{kj}} = 0 \tag{34}$$

Applying this at N control points and separating known and unknown variables, we obtain the system

$$\begin{bmatrix} \phi_{\mu_{11}} & \phi_{\mu_{12}} & \cdots & \phi_{\mu_{1N}} \\ \phi_{\mu_{21}} & \phi_{\mu_{22}} & \cdots & \phi_{\mu_{2N}} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{\mu_{N1}} & \phi_{\mu_{N2}} & \cdots & \phi_{\mu_{NN}} \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{bmatrix} = \begin{bmatrix} -\sum_{j=1}^{N} \sigma_j \phi_{\sigma_{1j}} \\ -\sum_{j=1}^{N} \sigma_j \phi_{\sigma_{2j}} \\ \vdots \\ -\sum_{j=1}^{N} \sigma_j \phi_{\sigma_{Nj}} \end{bmatrix} \tag{35}$$

Solve this system using `numpy.linalg.solve`. The velocity on the surface of each panel is then given by

$$\mathbf{u}_i = \mathbf{u}_\infty + \nabla \mu \tag{36}$$

Implement this velocity calculation. Compare results with the previous two formulations, particularly in terms of the resultant doublet distribution. What do you notice? What advantages/disadvantages do you see in the Morino formulation versus the Neumann and source-free Dirichlet formulations?