Thomas Slagle (72602272)

Source Panel Project

A. Overview

This assignment provides the results for a Python Program that estimates the coefficient of pressure over the surface of a cylinder with radius 1 in a freestream flow. The source panel method is a basic Computational Fluid Dynamics (CFD) method for breaking down an arbitrary shape into smaller components, or panels, and estimating the aerodynamic quantities on each panel. The results of this code were also compared to the analytical solution which provides an exact solution to the source panel approximation.

B. Governing Equations

Start with (1), which defines the infinitesimal potential function along the shape. (2) integrates the function and defines the potential function for the entire shape.

$$d\phi = \frac{\lambda ds}{2\pi} ln(r) \tag{1}$$

$$\phi(x, y) = \int_a^b \frac{\lambda ds}{2\pi} \ln(r) \tag{2}$$

For the source panel method, the function (shape) is broken down into linearly varying panels. For the jth panel, where there are a finite n panels; the following equations hold:

$$\Delta\phi_j = \frac{\lambda_j}{2\pi} \int_j \ln(r_{ij})\, ds_j \tag{3}$$

(3) gives the contribution to the potential function at point P (center) from panel j.

$$V_{n_i} = \frac{\partial}{\partial n_i}[\phi(x_i, y_i)] = \frac{\lambda_i}{2} + \sum_{j=1}^{n} \frac{\lambda_j}{2\pi} \int_j \frac{\partial}{\partial n_i}(\ln(r_{ij})\, ds_j \tag{4}$$

(4) gives the normal velocity contribution at P. Since the body is a streamline, (5) holds.

$$V_{\infty n} + V_{n_i} = 0 \rightarrow \frac{\lambda_i}{2} + \sum_{j=1}^{n} \frac{\lambda_j}{2\pi} I_{ij} + V_\infty \cos(\beta_i) = 0 \tag{5}$$

$$V_t = \frac{\partial\phi}{\partial s} = \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{\lambda_j}{2\pi} \int_j \frac{\partial}{\partial s}(\ln(r_{ij})\, ds_j \tag{6}$$

(6) gives the tangent velocity contribution at P. The term for j=i is zero. The total surface velocity is (7).

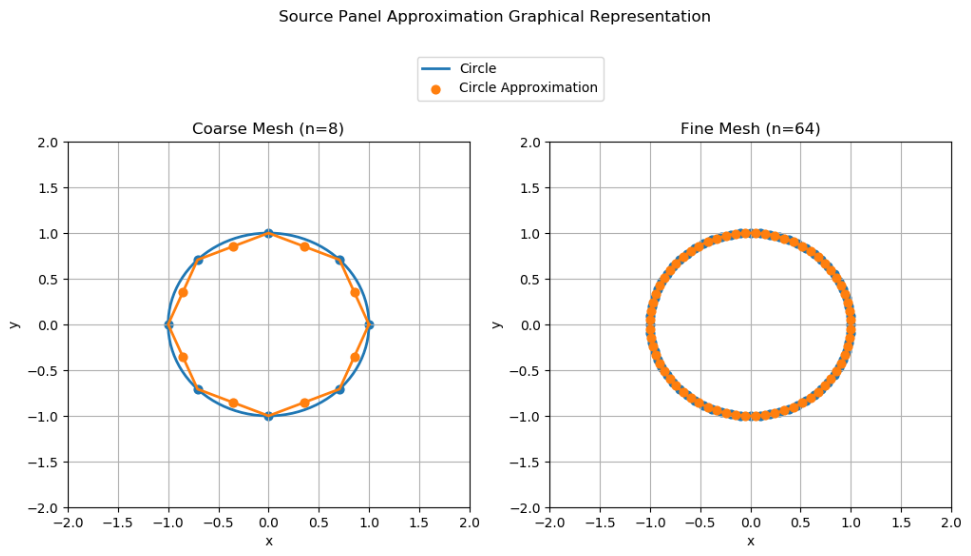
$$V_i = V_\infty \sin(\beta_i) + \sum_{j=1}^{n} \frac{\lambda_j}{2\pi} I_{ij} \tag{7}$$

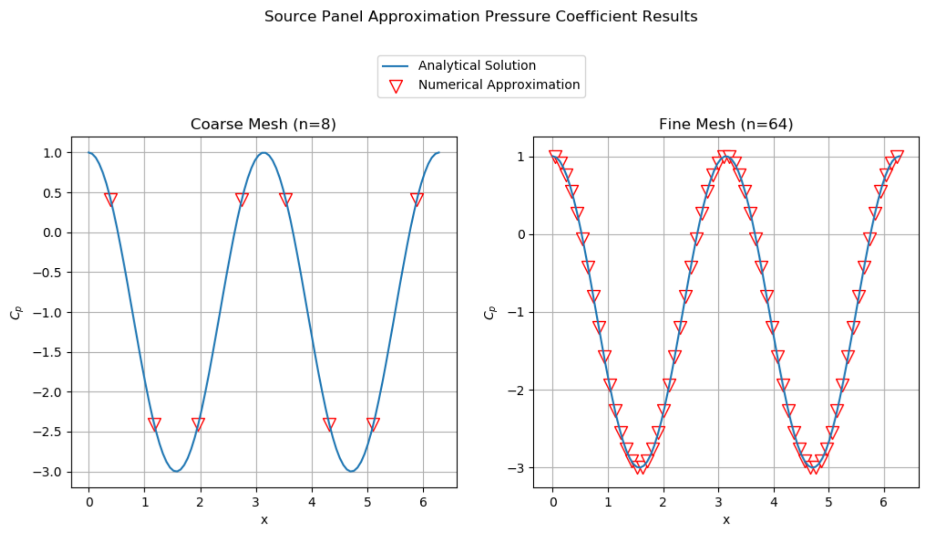
$$C_{p,i} = 1 - \left(\frac{V_i}{V_\infty}\right)^2 \tag{8}$$

$$\sum_{j=1}^{n} \lambda_j S_j = 0 \tag{9}$$

The values of lambda must obey equation (9).

C. Results





D. Conclusions

1. As the number of panels increased, the results resembled the analytical solution more. For the first geometry graph, the panels more closely resembled the shape of the circle. For the Cp graph, the numerical result lead to a more smoothed out cosine curve.
2. The code compared very well to the analytical solution. As mentioned in my answer to question 1, the numerical approximation more closely resembled the analytical solution as the number of panels increased.
3. In the real world, we would expect the results to resemble the general shape of the numerical results however, there will be some deviation in the graph. Our model makes several assumptions and neglects factors that influence Cp in the real word such as skin friction, the effects of the boundary layer, and the effects of backflow/turbulence.

E. Code

The methods for defining the panel method and computing the integrals for the source panel method are based on the work from the barbagroup/Aeropython, available here: Barba, Lorena A., and Mesnard, Olivier (2019). Aero Python: classical aerodynamics of potential flow using Python. Journal of Open Source Education, 2(15), 45, https://doi.org/10.21105/jose.00045

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import integrate

#Defining Panel Method
class Panel:
    def __init__(self, xa, ya, xb, yb):
        self.xa, self.ya=xa,ya
        self.xb, self.yb=xa,yb

        self.xc, self.yc = (xa+xb)/2,(ya+yb)/2
        self.length=np.sqrt((xb-xa)**2+(yb-ya)**2)

        if (xb-xa) <= 0.0:
            self.beta=np.arccos((yb-ya)/self.length)
        elif (xb-xa) > 0.0:
            self.beta=np.pi+np.arccos(-(yb-ya)/self.length)

        self.lambda_ = 0.0
        self.vt = 0.0
        self.cp = 0.0

#Integral of panel contribution at the center-point in the normal direction
def normalIntegral(p_i,p_j):
    def integrand(sourcesheet):
        return (((p_i.xc - (p_j.xa-
np.sin(p_j.beta)*sourcesheet))*np.cos(p_i.beta) +
                 (p_i.yc - (p_j.ya+np.cos(p_j.beta)*sourcesheet))*np.sin(p_i.beta
))/
                ((p_i.xc - (p_j.xa-np.sin(p_j.beta)*sourcesheet))**2 +
                 (p_i.yc - (p_j.ya+np.cos(p_j.beta)*sourcesheet))**2))
    return (integrate.quad(integrand,0.0,p_j.length)[0])

#Integral of panel contribution at the center-point in the tangential direction
def tangentialIntegral(p_i,p_j):
    def integrand(sourcesheet):
        return ((-(p_i.xc - (p_j.xa-
np.sin(p_j.beta)*sourcesheet))*np.sin(p_i.beta) +
                  (p_i.yc - (p_j.ya+np.cos(p_j.beta)*sourcesheet))*np.cos(p_i.bet
a))/
                 ((p_i.xc - (p_j.xa-np.sin(p_j.beta)*sourcesheet))**2 +
                  (p_i.yc - (p_j.ya+np.cos(p_j.beta)*sourcesheet))**2))
    return (integrate.quad(integrand,0.0,p_j.length)[0])

#Freestream
U_inf = 1

#Definition of Geometry put in Freestream flow
#Cylinder
#delta theta
dtheta=100
#Radius
R = 1
#Center location
x_0, y_0 = 0.0, 0.0
#Theta array
theta_a = np.linspace(0.0,2*np.pi,dtheta)
X_cylinder, Y_cylinder = (x_0 + R*np.cos(theta_a),
                          y_0 + R*np.sin(theta_a))

#Plot of Cylinder
x_figsize = 6
y_figsize = 12
figure_1, (ax1, ax2) = plt.subplots(1,2, figsize=(y_figsize,x_figsize))
figure_1.suptitle('Source Panel Approximation Graphical Representation')
plt.subplots_adjust(top=0.75)
ax1.grid()
ax1.set_title('Coarse Mesh (n=8)')
ax1.set(xlabel='x',ylabel='y')
ax1.plot(X_cylinder,Y_cylinder,linewidth=2)
ax1.set_xlim([-2,2])
ax1.set_ylim([-2,2])

ax2.set_title('Fine Mesh (n=64)')
ax2.grid()
ax2.set(xlabel='x',ylabel='y')
ax2.plot(X_cylinder,Y_cylinder,linewidth=2, label='Circle')
ax2.set_xlim([-2,2])
ax2.set_ylim([-2,2])

### Course ###
#Break Geometry into Panels
number_panels = 8
```

```python
#Panel endpoints
x_ends = R*np.cos(np.linspace(0.0,2*np.pi,number_panels+1))
y_ends = R*np.sin(np.linspace(0.0,2*np.pi,number_panels+1))
#Create Panels
panels=np.empty(number_panels,dtype=object)
for i in range(number_panels):
    panels[i]=Panel(x_ends[i],y_ends[i],x_ends[i+1],y_ends[i+1])
#Plot of panels
ax1.plot(x_ends,y_ends,linewidth=2)
ax1.scatter([p.xa for p in panels], [p.ya for p in panels],s=40)
ax1.scatter([p.xc for p in panels], [p.yc for p in panels],s=40,zorder=3)


#Source Influence Matrix
Matrix_A = np.empty((number_panels, number_panels), dtype=float)
np.fill_diagonal(Matrix_A, 0.5)
for i, p_i in enumerate(panels):
    for j, p_j in enumerate(panels):
        if i != j:
            Matrix_A[i, j] = 0.5/np.pi * normalIntegral(p_i,p_j)
#RHS of system
b = -U_inf * np.cos([p.beta for p in panels])

lambda_ = np.linalg.solve(Matrix_A, b)
for i, panel in enumerate(panels):
    panel.lambda_ = lambda_[i]

#Source Influence Matrix 2
Matrix_A = np.empty((number_panels, number_panels), dtype=float)
np.fill_diagonal(Matrix_A, 0.0)
for i, p_i in enumerate(panels):
    for j, p_j in enumerate(panels):
        if i != j:
            Matrix_A[i, j] = 0.5/np.pi * tangentialIntegral(p_i,p_j)
#RHS of system
b = -U_inf * np.sin([panel.beta for panel in panels])
#tangent velocity
V_tangential = np.dot(Matrix_A, lambda_) + b
for i, panel in enumerate(panels):
    panel.vt = V_tangential[i]

#Numerical Pressure Coefficient
for panel in panels:
    panel.cp = 1.0 - (panel.vt/U_inf)**2
```

```
#Convert Numerical Approximation Results to Polar Coordinates
center_tup=zip([p.xc for p in panels],[p.yc for p in panels])
delta_list=[]
for each_center in (list(center_tup)):
    if each_center[1]>0:
        delta=np.arctan2(each_center[1],each_center[0])
    elif each_center[1]<=0:
        delta=(2*np.pi + np.arctan2(each_center[1],each_center[0]))
    delta_list.append(delta)

#Analytical Pressure Coefficient
c_p = 1.0 - 4*(Y_cylinder/R)**2
figure_2, (axx1, axx2) = plt.subplots(1,2, figsize=(y_figsize,x_figsize))
plt.subplots_adjust(top=0.75)
figure_2.suptitle('Source Panel Approximation Pressure Coefficient Results')
axx1.grid()
axx1.set_title('Coarse Mesh (n=8)')
axx1.set(xlabel='x',ylabel='$C_p$')
axx1.plot(theta_a, c_p)
axx1.scatter(delta_list, [p.cp for p in panels],color='white',marker='v',edgecolo
rs='r',s=100,zorder=2)

### Fine ###
#Break Geometry into Panels
number_panels = 64
#Panel endpoints
x_ends = R*np.cos(np.linspace(0.0,2*np.pi,number_panels+1))
y_ends = R*np.sin(np.linspace(0.0,2*np.pi,number_panels+1))
#Create Panels
panels=np.empty(number_panels,dtype=object)
for i in range(number_panels):
    panels[i]=Panel(x_ends[i],y_ends[i],x_ends[i+1],y_ends[i+1])
#Plot of panels
ax2.plot(x_ends,y_ends,linewidth=2)
ax2.scatter([p.xa for p in panels], [p.ya for p in panels],s=40)
ax2.scatter([p.xc for p in panels], [p.yc for p in panels],s=40,zorder=3, label='
Circle Approximation')
ax2.legend(loc='upper right', bbox_to_anchor=(0.15,1.25))

#Source Influence Matrix
Matrix_A = np.empty((number_panels, number_panels), dtype=float)
np.fill_diagonal(Matrix_A, 0.5)
for i, p_i in enumerate(panels):
    for j, p_j in enumerate(panels):
        if i != j:
            Matrix_A[i, j] = 0.5/np.pi * normalIntegral(p_i,p_j)
#RHS of system
b = -U_inf * np.cos([p.beta for p in panels])

lambda_ = np.linalg.solve(Matrix_A, b)
for i, panel in enumerate(panels):
    panel.lambda_ = lambda_[i]

#Source Influence Matrix 2
Matrix_A = np.empty((number_panels, number_panels), dtype=float)
np.fill_diagonal(Matrix_A, 0.0)
for i, p_i in enumerate(panels):
    for j, p_j in enumerate(panels):
        if i != j:
            Matrix_A[i, j] = 0.5/np.pi * tangentialIntegral(p_i,p_j)
#RHS of system
b = -U_inf * np.sin([panel.beta for panel in panels])
#tangent velocity
V_tangential = np.dot(Matrix_A, lambda_) + b
for i, panel in enumerate(panels):
    panel.vt = V_tangential[i]

#Numerical Pressure Coefficient
for panel in panels:
    panel.cp = 1.0 - (panel.vt/U_inf)**2

#Convert Numerical Approximation Results to Polar Coordinates
center_tup=zip([p.xc for p in panels],[p.yc for p in panels])
delta_list=[]
for each_center in (list(center_tup)):
    if each_center[1]>0:
        delta=np.arctan2(each_center[1],each_center[0])
    elif each_center[1]<=0:
        delta=(2*np.pi + np.arctan2(each_center[1],each_center[0]))
    delta_list.append(delta)

#Analytical Pressure Coefficient
c_p = 1.0 - 4*(Y_cylinder/R)**2
axx2.grid()
axx2.set_title('Fine Mesh (n=64)')
axx2.set(xlabel='x',ylabel='$C_p$')
axx2.plot(theta_a, c_p, label='Analytical Solution')
axx2.scatter(delta_list, [p.cp for p in panels],color='white',marker='v',edgecolo
rs='r',s=100,zorder=2, label='Numerical Approximation')
axx2.legend(loc='upper right', bbox_to_anchor=(0.15,1.25))
```

```
figure_1.savefig('Source Panel Circle Plot')
figure_2.savefig('Source Panel Cp')
```