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.

(1)

(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:

(3)

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

(4)

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

(5)

(6)

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

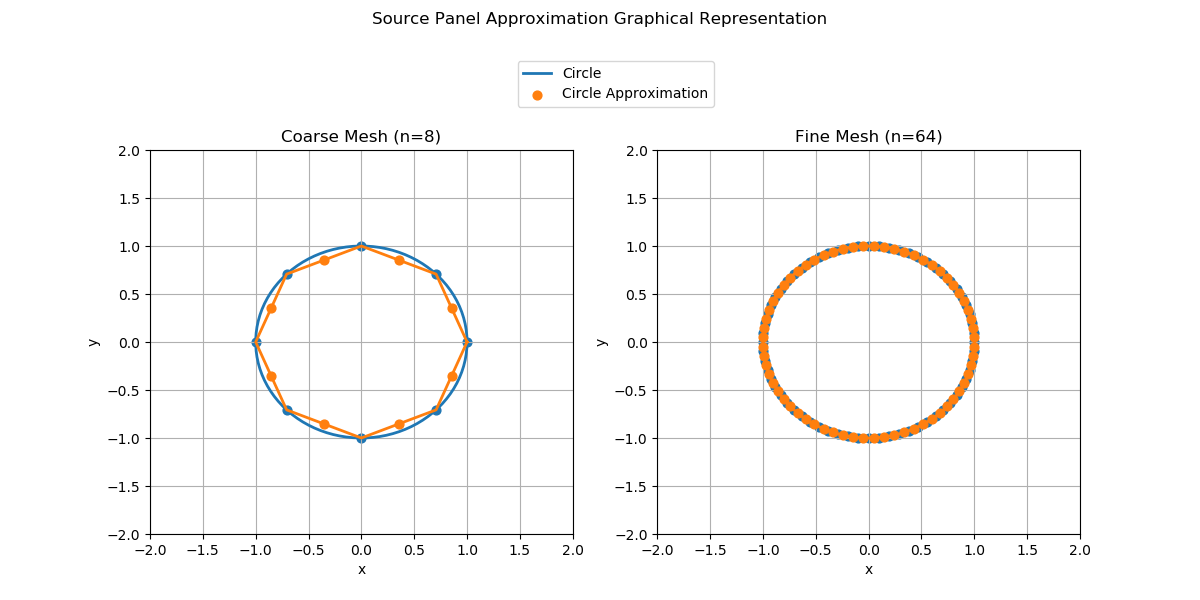
(7)

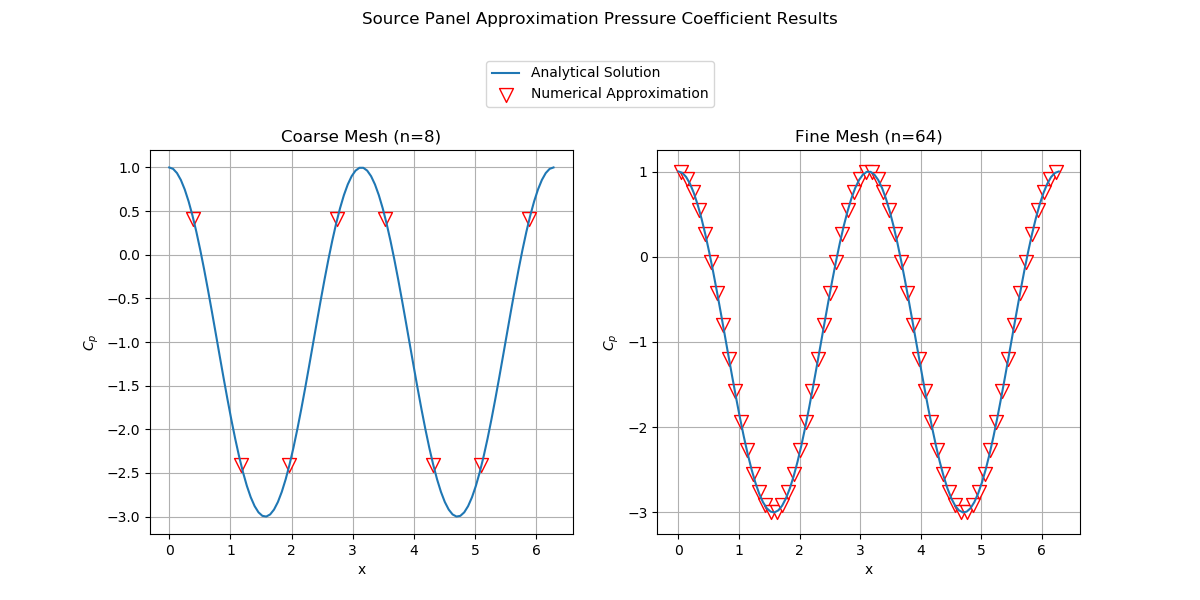
(8)

(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.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])

*#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

*#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',edgecolors='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',edgecolors='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')