Assignment 01

AERO 455 - CFD for Aerospace Applications

Instructor: Dr. Ziad Boutanios

Concordia University - MIAE

Given: 29th January 2022

Due: 5pm EST on 14th February 2022

By: Christian Rowsell (40092183)

1 The Source Panel Method

# 1.1 Programming a source panel method

Write a 2D source panel code to calculate the solution for the Laplacian equation of the potential function *ϕ* which describes incompressible, inviscid and irrotational flow,

∇2*ϕ* = 0*.* (1.1)

You need to follow the procedure outlined in the source panel section of lecture 04, from slide 22 to slide 39, for the generic panel geometry in figure 8. The general program flow is the following,

In order to see all of the code, please refer to the attached files. However, best efforts will be made to include the appropriate snippets of code for each question. It may not be possible for all questions, due to the fact that the code was not written following the questions, and instead the entire source panel method was analyzed.

1. Using the Python programming language, program a function that programs equation 43, which is required to assemble the normal velocity boundary condition (equation 29) at the panel control points as induced by the potential function of another panel. *(20 points)*

# Imported packages  
import numpy as np  
import math as m  
# User defined functions  
  
# np.seterr('raise') # Raise floating point error when encountered  
  
  
# Define function to calculate geometric integral I and J  
def geometric\_integral(control, boundary, phi, length):  
  
 num\_panel = len(control) # Determine number of panels  
  
 # Initialize array for integrals  
 i\_int = np.zeros([num\_panel, num\_panel])  
 j\_int = np.zeros([num\_panel, num\_panel])  
  
 # Loop over panels to compute integrals  
 for i in range(num\_panel): # Loop over i panels  
 for j in range(num\_panel): # Loop over j panels  
 if j != i: # Calculate when i is not j  
 A = -(control[i][0] - boundary[j][0]) \* np.cos(phi[j]) - (control[i][1] - boundary[j][1]) \* np.sin(phi[j]) # A Term  
 B = (control[i][0] - boundary[j][0]) \*\* 2 + (control[i][1] - boundary[j][1]) \*\* 2 # B Term  
 Cn = np.sin(phi[i] - phi[j]) # Normal C Term  
 Dn = -(control[i][0] - boundary[j][0]) \* np.sin(phi[i]) + (control[i][1] - boundary[j][1]) \* np.cos(phi[i]) # Normal D Term  
 Ct = -np.cos(phi[i] - phi[j]) # Tangential C Term  
 Dt = (control[i][0] - boundary[j][0]) \* np.cos(phi[i]) + (control[i][1] - boundary[j][1]) \* np.sin(phi[i]) # Tangential D Term  
 E = np.sqrt(B - A \*\* 2) # E Term  
 # Set integral to 0 if E is 0 or complex or NAN or INF  
 if E == 0 or np.iscomplex(E) or np.isnan(E) or np.isinf(E):  
 i\_int[i, j] = 0  
 j\_int[i, j] = 0  
 else:  
 # Calculate I integral  
 i\_term1 = 0.5 \* Cn \* np.log((length[j] \*\* 2 + 2 \* A \* length[j] + B) / B)  
 i\_term2 = ((Dn - A \* Cn)/E) \* (m.atan2((length[j] + A), E) - m.atan2(A, E))  
 i\_int[i, j] = i\_term1 + i\_term2  
 # Calculate J integral  
 j\_term1 = 0.5 \* Ct \* np.log((length[j] \*\* 2 + 2 \* A \* length[j] + B) / B)  
 j\_term2 = ((Dt - A \* Ct) / E) \* (m.atan2((length[j] + A), E) - m.atan2(A, E))  
 j\_int[i, j] = j\_term1 + j\_term2  
  
 # Remove values that cause error  
 if np.iscomplex(i\_int[i, j]) or np.isnan(i\_int[i, j]) or np.isinf(i\_int[i, j]):  
 i\_int[i, j] = 0  
 if np.iscomplex(j\_int[i, j]) or np.isnan(j\_int[i, j]) or np.isinf(j\_int[i, j]):  
 j\_int[i, j] = 0  
  
 return i\_int, j\_int

1. This will allow you to assemble this boundary condition for all panels leading to *n* equations for *n λi* unknows, where *λi* is the source strength per unit length at a panel’s control point. *(5 points)*

# Calculate I and J integral  
I, J = geometric\_integral(control\_data, data, phi, panel\_length)  
  
# Create A matrix  
A = np.zeros([panel\_num, panel\_num])  
for i in range(panel\_num):  
 for j in range(panel\_num):  
 if i == j:  
 A[i, j] = np.pi  
 else:  
 A[i, j] = I[i, j]  
# Create b array in Ax + b = 0  
b = np.zeros(panel\_num)  
for i in range(panel\_num):  
 b[i] = - V\_inf \* 2 \* np.pi \* np.cos(beta[i])

1. This well-defined system must be solved using the appropriate methods from the Python libraries. *(15 points)*

lam = np.linalg.solve(A, b)

1. At this point you should program another function for the accuracy check of equation 35. *(5 points)*
2. print(str(num) + ' Panels')  
   print("Sum of L: ", sum(lam\*panel\_length))  
     
   # Calculate velocities and pressure coefficient  
   vt = np.zeros(panel\_num)  
   cp = np.zeros(panel\_num)  
     
   for i in range(panel\_num):  
    cntr = 0  
    for j in range(panel\_num):  
    cntr = cntr + (lam[j] / (2 \* np.pi)) \* J[i, j]  
     
    vt[i] = V\_inf \* np.sin(beta[i]) + cntr  
    cp[i] = 1 - (vt[i] / V\_inf)\*\*2  
     
   # Calculate analytical value for pressure coefficient and velocity  
   angle = np.linspace(0, 360, 360)  
   cp\_analytical = 2\*np.cos(2\*angle\*np.pi/180)-1  
   u\_theta = V\_inf \* 2 \* np.sin(angle \* np.pi / 180)  
     
   residual = sum(lam\*panel\_length)
3. Then program another function that allows you to assemble the velocity integral in equation 44. *(20 points)*

This is again the same part of code that is included for question 1.

1. Based on equation 44 and equation 32, program a function that calculates the surface velocity at every panel control point. *(5 points)*

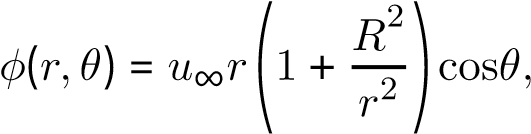
This is the same part of the code that is included for question 4.

1. Then program the last function to calculate the pressure coefficient at every control point using Bernouilli’s law as given by equation 34. *(5 points)*
2. This is the same part of the code that is included for question 4.

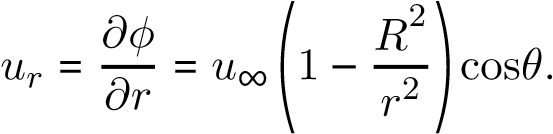
# 1.2 Validation of the source panel method code

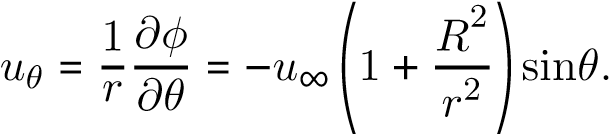
Validate the code you wrote in section 1.1 by repeating the 2D cylinder example in slides 40 to 44.

1. Use 4, 6, 8 and 10 panels to discretize the cylinder surface and compare the accuracy of each number of panels. *(10 points)*
2. Keep increasing the number of panels. How far can you go and why? Plot the residual against the number of panels. *(5 points)*
3. The analytical solution in polar coordinates for the potential function of a uniform flow around a circular cylinder is,

 (1.2)

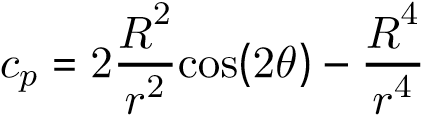
where *u*∞ is the freestream velocity magnitude. From equation 1.2 one can extract the velocity components in polar coordinates,

 (1.3)

 (1.4)

Compare the velocity magnitude obtained from the panel code to the one from the analytical solution for all discretizations. *(5 points)*

1. Plot the pressure coefficient at the surface of the cylinder against the angle *θ* for all discretizations, and compare to the analytical solution given by,

 (1.5)

*(5 points)*