Justine John "JJ" A. Serdoncillo
February 28, 2023

Homework 2

Number 1 c)

After determining the necessary complex potential function for this flow, the streamlines and potential lines are plotted as seen in the figure below. The solid lines correspond to the streamlines while the dashed, translucent lines correspond to the potential lines. The red points are the stagnation points and it can be seen that it does lie on the following required locations. The black arrows are the quiver plot for the velocity flow.

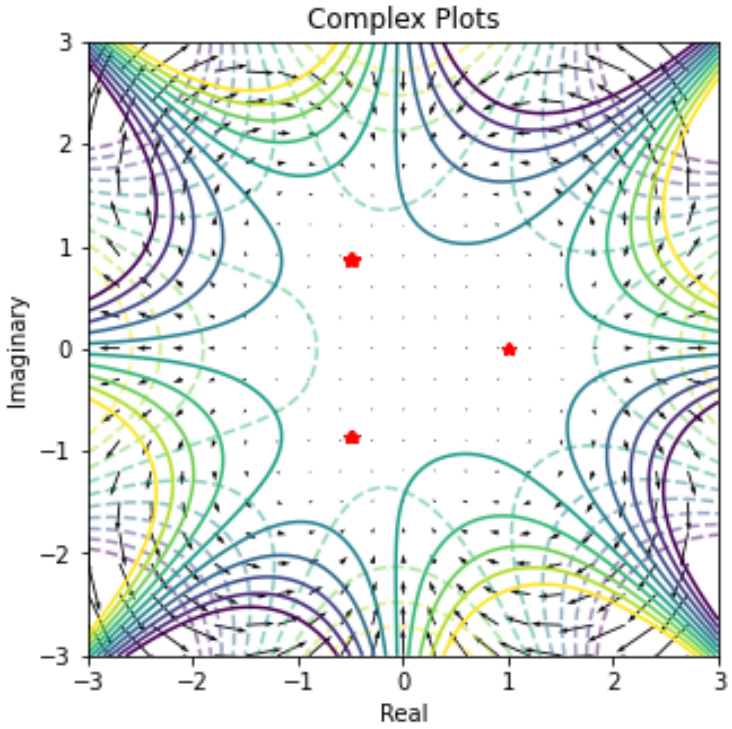

Figure 1: Complex Potential Flow for 3 Stagnation Points

Number 2 a)

After determining the necessary complex potential function for this flow, the streamlines are plotted as seen in the figure below. Similarly, the potential lines are also plotted as dashed translucent lines. It can be seen that the flow is acting on a flat plate and this is done using the method of images.

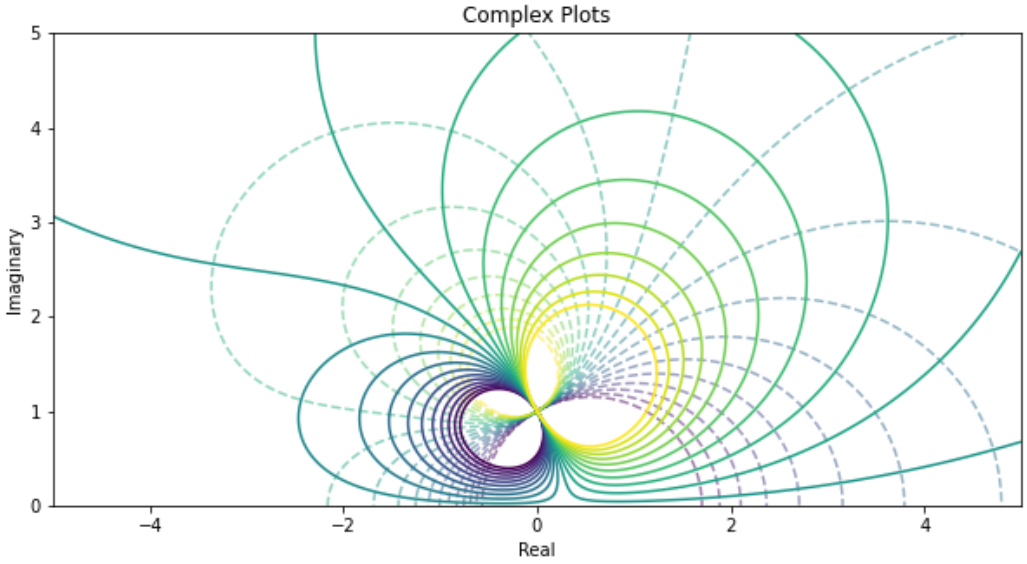

Figure 2: Complex Potential Flow for Dipole on Infinite Flat Plate

Number 3 d

The following streamlines are plotted for the two elliptical flows. The blue dots correspond to points that evaluate to the 0 streamline, while the cyan lines are with the 17cm streamline. It can be seen that at x=0, the cyan line is at around 60 which corresponds to the similar htop calculated of 11.67cm. The same can be said for the Prized Rose problem showing a smaller htop of around 3.33cm.

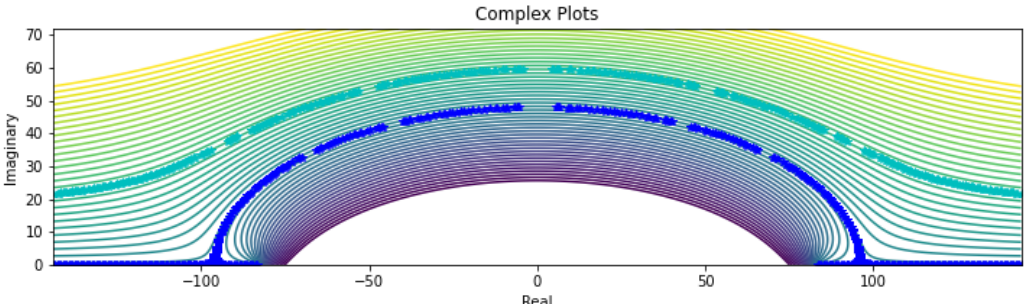

Figure 3: Streamlines for Elliptic Greenhouse

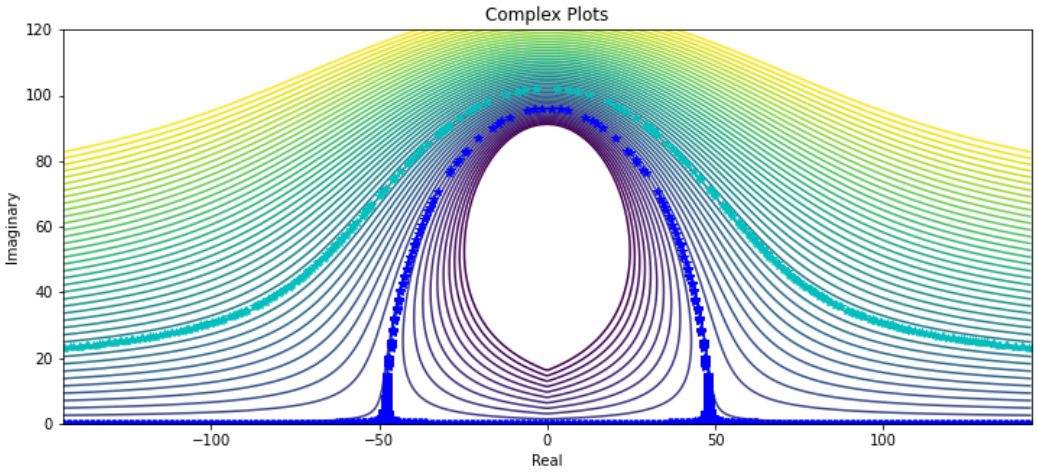

Figure 4: Streamlines for Elliptic Prized Rose

Using the BEM solver from the previous homework, the following figures are created as shown below.

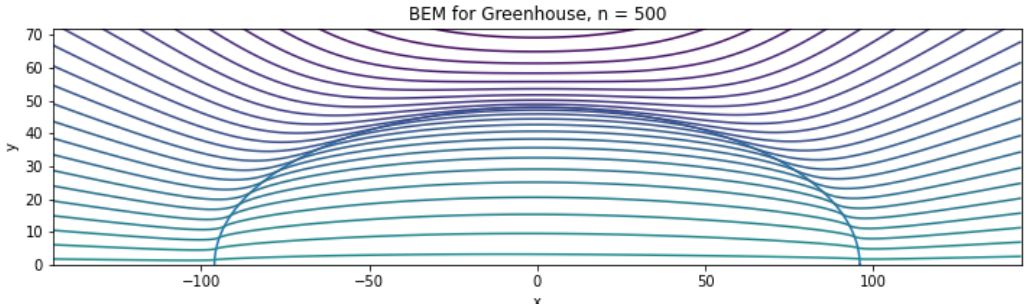

Figure 5: Streamlines from BEM Solver for Greenhouse

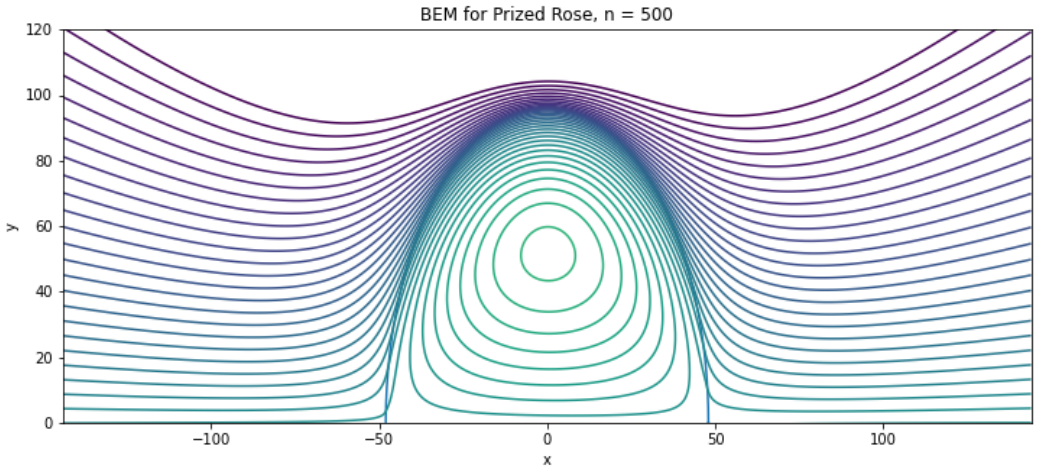

Figure 6: Streamlines from BEM Solver for Prized Rose

It can be seen that there is a clear difference between the analytical solution and the BEM method. It can be seen that from using the BEM method, the streamlines are somewhat going up instead of going down. At

close streamlines to the surface this may be an accurate representation, however, close to infinity, it doesn't show a proper uniform flow.

Appendix)

Python Code for Problems 1,2,3

```
# Imported from CFD Project
import numpy as np
import time
from scipy.stats import norm
import matplotlib.pyplot as plt
##################
def HW4(problem='blah',let='a',num=10,num2=None,
        plotzero=False,plotstag=False,func='nope',s=5,pot=True,stream=False):
    g = 500
    X = np.linspace(-s,s,g)
    Y = np.linspace(-s,s,g)

    if problem == 'Problem 1':
        fig, ax = plt.subplots(figsize=(5,5))
    elif problem == 'Problem 2':
        fig, ax = plt.subplots(figsize=(10,5))
    elif problem == 'Problem 3':
        if let == 'b':
            fig, ax = plt.subplots(figsize=(12,3))
        if let == 'c':
            fig, ax = plt.subplots(figsize=(12,5))
    if problem == 'blah':
        fig, ax = plt.subplots(figsize=(20,20))
    #ax.grid()
    ax.set_title('Complex Plots')
    ax.set_xlabel('Real')
    ax.set_ylabel('Imaginary')
    ax.set_xlim([-s,s])
    ax.set_ylim([-s,s])

    if problem == 'Problem 1':
        pass
    elif problem == 'Problem 2':
        ax.set_xlim([-s,s])
        ax.set_ylim([ 0,s])
    elif problem == 'Problem 3':
        if let == 'b':
            X = np.linspace(-144,144,g)
            Y = np.linspace(   0, 72,g)
            ax.set_xlim([-144,144])
            ax.set_ylim([   0, 72])
        if let == 'c':
            X = np.linspace(-144,144,g)
            Y = np.linspace(   0,120,g)
            ax.set_xlim([-144,144])
            ax.set_ylim([   0,120])

    x,y = np.meshgrid(X,Y)
    r = np.arctan2(y,x)
    t = np.sqrt(x**2+y**2)
    z = x + 1j*y
    ''' Input Function Here '''

    if problem == 'Problem 1':
        Fz = z**4/4 - z
        w = z**3 - 1
        u = w.real
        v = w.imag
    elif problem == 'Problem 2':
        M = 1
        Fz = -M*(np.sqrt(3)+z) / (z**2+1)
        w = (z**2+2*np.sqrt(3)*z-1)/ (z**2+1)**2
        u = w.real
        v = w.imag
    elif problem == 'Problem 3':
        if let == 'b':
            a = 72
```

```
            b = 24 * np.sqrt(3)
        if let == 'c':
            a = 72
            b = 24 * np.sqrt(3) * 1j
        l = z/2 + np.sign(z.real) * np.sqrt((z/2)**2-b**2)
        Fz = l + a**2/l
    else:
        Fz = -M*(np.sqrt(3)+z) / (z**2+1)

    phi = Fz.real
    psi = Fz.imag

    if func == 'lmao':
        if pot == True:
            mean = np.mean(phi)
            std = np.std(phi)
            levels = np.linspace(norm(mean,std).ppf(0.1), norm(mean,std).ppf(0.9), num)
            ax.contour(x, y, phi, levels=levels, linestyles='dashed',alpha=0.5)
        mean = np.mean(psi)
        std = np.std(psi)
        levels = np.linspace(norm(mean,std).ppf(0.1), norm(mean,std).ppf(0.9), num)
        ax.contour(x, y, psi, levels=levels)
    else:
        if pot == True:
            ax.contour(x, y, phi, num, linestyles='dashed',alpha=0.5)
        ax.contour(x, y, psi, num)

    #### Plot Zero Streamline ####
    if plotzero == True:
        zero = np.argwhere(np.around(psi,1) == 0)
        zeros = np.zeros((2,zero.shape[0]))
        for i in range(zero.shape[0]):
            zeros[0,i] = x[zero[i,0],zero[i,1]]
            zeros[1,i] = y[zero[i,0],zero[i,1]]
        ax.plot(zeros[0],zeros[1],'b*')

    if stream != False:
        zero = np.argwhere(np.around(psi,1) == stream)
        zeros = np.zeros((2,zero.shape[0]))
        for i in range(zero.shape[0]):
            zeros[0,i] = x[zero[i,0],zero[i,1]]
            zeros[1,i] = y[zero[i,0],zero[i,1]]
        ax.plot(zeros[0],zeros[1],'c*')

    if 'u' and 'v' in locals():
        mag = np.sqrt(u**2+v**2)
        #### Plot Stagnation Points ####
        if plotstag == True:
            zero = np.argwhere(np.around(mag,1) == 0)
            zeros = np.zeros((2,zero.shape[0]))
            for i in range(zero.shape[0]):
                zeros[0,i] = x[zero[i,0],zero[i,1]]
                zeros[1,i] = y[zero[i,0],zero[i,1]]
            ax.plot(zeros[0],zeros[1],'r*')
        if num2 != None:
            ll = int(g/num2)
            skip = (slice(None, None, ll), slice(None, None, ll))
            ax.quiver(x[skip],y[skip],u[skip],v[skip])
    if problem == 'Problem 1':
        name = 'first'
    elif problem == 'Problem 2':
        name = 'second'
    elif problem == 'Problem 3':
        if let == 'b':
            name = 'third'
        if let == 'c':
            name = 'fourth'
    if 'name' in locals():
        gg = 'images/' + name + '.png'
        plt.savefig(gg)

if __name__ == "__main__":
```

```
    #HW4('Problem 1',func='lmao',num=10,plotstag=True,num2=20,s=3)
    HW4('Problem 2',func='lmao',num=20,s=5,pot=True)# num2=50, plotstag=True # weird
    stagnations stuff
    HW4('Problem 3','b',func='lmao',pot=False,num=50,plotzero=True,stream=17) # good now tbh
    HW4('Problem 3','c',func='lmao',pot=False,num=50,plotzero=True,stream=17) # also good
    now tbh
    #HW4(func='lmao',pot=False,num=50,s=5,num2=51)
```

Python Code for Problem 3 BEM Solver

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 10 10:06:00 2023

@author: jjser
"""
# Imported from CFD Project
import numpy as np
import matplotlib.pyplot as plt
import math
from numpy import linalg as la
from numpy import asarray
import pandas
import time
from scipy.stats import norm

##################
def G(r):
    return np.log(r)/(2*np.pi)
def dGdn(R,N,r):
    return R @ N / (2*np.pi*r)

def prob(a,b,n,num):
    # 'a' is the length of the major axis
    # 'b' is the length of the minor axis
    # 'n' is the number of elements in the boundary element method
    # 'num' is the number of streamlines
    if a > b:
        shape = 'Greenhouse'
        prob = 1
        fig0, ax0 = plt.subplots(figsize=(12,3))
    elif b > a:
        shape = 'Prized Rose'
        prob = 2
        fig0, ax0 = plt.subplots(figsize=(12,5))
    else:
        shape = 'bro wrong problem'
        prob = False

    dtheta = 2*np.pi / n
    x0 = np.zeros(n+1)
    y0 = np.zeros(n+1)
    theta = np.zeros(n+1)

    # Creating the actual circle first
    for i in range(n+1):
        theta[i] = i * dtheta
        x0[i] = a*np.cos(theta[i])
        y0[i] = b*np.sin(theta[i])
    # Applying BEM
    A = np.zeros((n,n))
    B = np.zeros((n,n))
    area = np.zeros(n)
    cx = np.zeros(n)
    cy = np.zeros(n)
    N = np.zeros((n,2))

    #fig0, ax0 = plt.subplots(figsize=(5,5))
    #fig0, ax0 = plt.subplots()
    title = 'BEM for ' + shape + ', n = ' + str(n)
    ax0.set_title(title)
    ax0.set_xlabel('x')
    ax0.set_ylabel('y')
    s = 1.5*max(a,b)
    if a > b:
        ax0.set_xlim([-144,144])
        ax0.set_ylim([   0, 72])
    elif b > a:
        ax0.set_xlim([-144,144])
        ax0.set_ylim([   0,120])
```

```
    for i in range(n):
        area[i] = math.sqrt((x0[i+1]-x0[i])**2+(y0[i+1]-y0[i])**2)
        cx[i] = 0.5 *  (x0[i+1]+x0[i])
        cy[i] = 0.5 *  (y0[i+1]+y0[i])
        N[i][0] = -(y0[i+1]-y0[i])
        N[i][1] =  (x0[i+1]-x0[i])
        N[i] = N[i] / la.norm(N[i])
    for i in range(n):
        for j in range(n):
            if i==j:
                B[i,j] = 0
                A[i,j] = -0.5
            else:
                R = [cx[j]-cx[i],cy[j]-cy[i]]
                r = la.norm(R)
                R = R/r
                A[i,j] = dGdn(R,N[j],r) * area[j]
                B[i,j] = G(r) * area[j]
    ax0.plot(x0,y0)

    # Solve for psi1 and dpsi1dn
    psi1 = np.zeros(n)
    for i in range(n):
        psi1[i] = -y0[i]
    dpsi1dn = la.inv(B) @ A @ psi1.T

    # Plotting the Cylinder
    g = 501
    x = np.linspace(-s,s,g)
    y = np.linspace(-s,s,g)
    X,Y = np.meshgrid(x,y)
    psi = np.zeros((g,g))

    # Streamfunction Contours
    for k in range(n):
        rx = cx[k] - X
        ry = cy[k] - Y
        rmag = np.sqrt(rx**2+ry**2)
        psi += (psi1[k]*(rx*N[k][0]+ry*N[k][1])/(2*np.pi*rmag)-G(rmag)*dpsi1dn[k])*area[k]
    psi += Y

    mean = np.mean(psi)
    std = np.std(psi)
    levels = np.linspace(norm(mean,std).ppf(0.1), norm(mean,std).ppf(0.9), num)
    ax0.contour(x, y, psi, levels=levels)

    if prob != False:
        name = 'images/BEM' + str(prob) + '.png'
        fig0.savefig(name)
    '''
    zero = np.argwhere(np.around(psi,1) == 0)
    zeros = np.zeros((2,zero.shape[0]))
    for i in range(zero.shape[0]):
        zeros[0,i] = X[zero[i,0],zero[i,1]]
        zeros[1,i] = Y[zero[i,0],zero[i,1]]
    ax0.plot(zeros[0],zeros[1],'b*')

    zero = np.argwhere(np.around(psi,1) == 17)
    zeros = np.zeros((2,zero.shape[0]))
    for i in range(zero.shape[0]):
        zeros[0,i] = X[zero[i,0],zero[i,1]]
        zeros[1,i] = Y[zero[i,0],zero[i,1]]
    ax0.plot(zeros[0],zeros[1],'c*')
    '''

##### BEM for a 4:1 ellipse
#prob(4,1,500,15)
prob(96,48,500,50)
prob(48,96,500,50)
start_time = time.time()
print("--- %10s seconds ---" % np.round((time.time() - start_time),4))
```