Marine Hydrodynamics
Assignment 1

Sebastian Gjertsen

14. september 2015

Assignment

For specified (see below) two-dimensional geometries, assuming potential theory in unbounded fluid, and use of Green's second identity, calculate the velocity potential along the body and the added mass forces, for a circle, an ellipse, a square and a rectangle, moving laterally, and with rotation. Find also the cross coupling added mass coefficients. For the circle, the reference solution is: $\phi = -a^2/(x^2)$ where a denotes the cylinder radius, $r^2 = x^2 + y^2$.

Theory, numerics and program

In this assignment we use the method of panels. By splitting the geometry into N equal parts, and assuming that ϕ, $\frac{\partial \phi}{\partial n}$ is constant every segment.
From Newman chapter 4 we have (79):

$$\int_C \phi \frac{\partial G}{\partial n} - G \frac{\partial \phi}{\partial n} = -\pi \phi(x, y)$$

where $G = ln(r)$, a source in 2D, and C is the circumference of the geometry

$$-\pi \phi(x_0) + \int_C \phi \frac{\partial}{\partial n} ln(r) dl = \int_C ln(r) \frac{\partial \phi}{\partial n} dl$$

where x_0 states a point on our geometry.

We got a trick from the lectures, turning it into:

$$-\pi phi(x_0) + \sum_{n=1}^{N} \phi(x_n)(\theta_a - \theta_b) = \int_C ln(r) \frac{\partial \phi}{\partial n} dl$$

$$\begin{pmatrix} -\pi & (\theta_a - \theta_b)_0 & (\theta_a - \theta_b)_1 & ... \\ (\theta_a - \theta_b)_N & -\pi & (\theta_a - \theta_b)_0 & ... \\ (\theta_a - \theta_b)_{N-1} & (\theta_a - \theta_b)_N & -\pi & ... \\ ... & ... & .... & ... \end{pmatrix} \begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ ... \\ \phi_N \end{pmatrix} = \begin{pmatrix} \int ln(r_0) \frac{\partial \phi}{\partial n} dl \\ \int ln(r_1) \frac{\partial \phi}{\partial n} dl \\ \int ln(r_2) \frac{\partial \phi}{\partial n} dl \\ ... \end{pmatrix}$$

To calculate the ϕ values we created a fictional point between the x_N points, where we stand in this ghostpoint and calculate the $\Delta\theta$ to the x_N values. This is done since we have assumed ϕ to be constant. To evaluate the integrals we used the trapezoidal method. Every line of the matrix on the right hand side is calculated by going around the geometry once.

The normal vector is defined as:

$$\bar{n} = \frac{\frac{x}{a^2}\bar{i} + \frac{y}{b^2}\bar{j}}{(\frac{x}{a^2})^2 + (\frac{y}{b^2})^2}$$

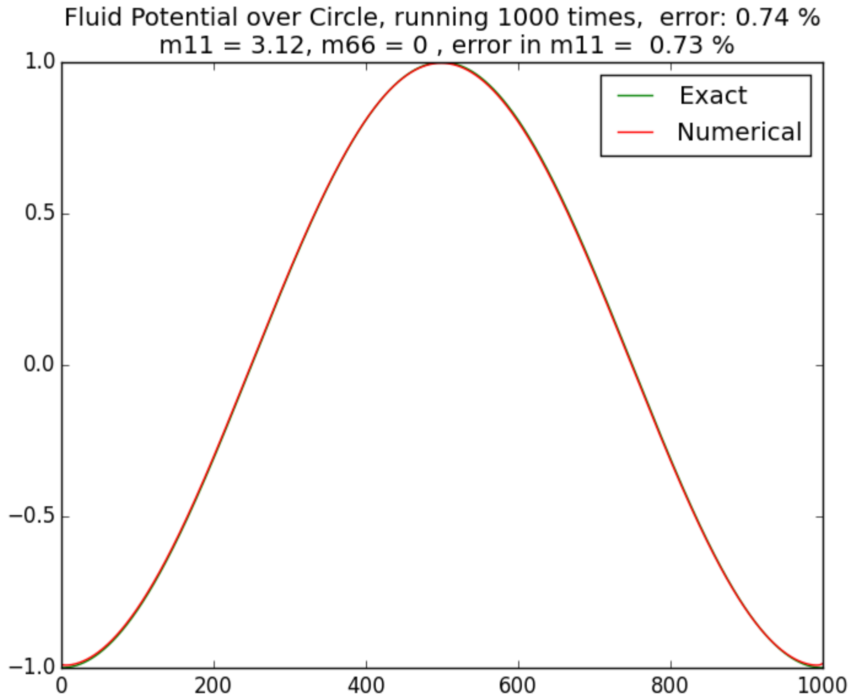
which is used for ellipse and for circle when $a = b$.

For a circle we have an exact solution for $\phi = -\frac{a^2 x}{r^2}$, which is $\phi = -acos(\theta)$ and an exact solution for added mass $m_{11} = \rho r_a^2 \pi$
For an ellipse we only have the exact solution for added mass $m_{11} = \rho r_a^2 \pi$, $m_{22} = \rho r_b^2 \pi$, $m66 = \frac{\pi}{8}\rho(r_a^2 - r_b^2)^2$.
Next page is the program code in Python

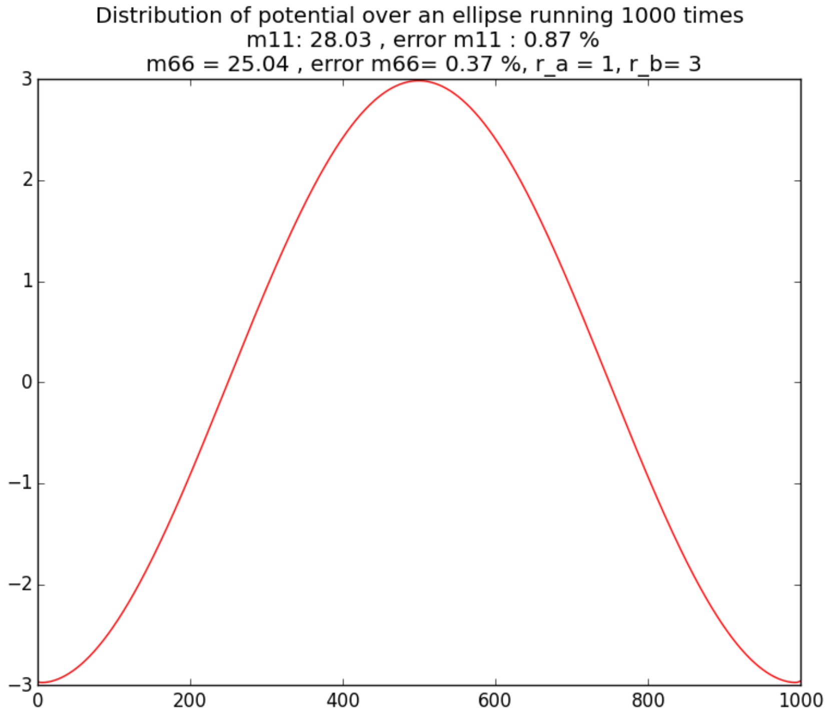
```
from matplotlib.pylab import *
import numpy as np
import math as math


def make_points(num_points,r_a, r_b):
        dx = 2*pi/num_points
        x = np.zeros(num_points*3)
        y = np.zeros(num_points*3)
        for i in range(num_points*2+1):
                x[i] = r_a*np.cos(i*dx)
                y[i] = r_b*np.sin(i*dx)
        return x,y

def make_angle(N,r_a,r_b):
        angle = np.zeros(N-1)
        x,y = make_points(N,r_a,r_b)
        matrix_ = np.zeros((N,N))
        for i in range(N):
                x_first = (x[i]+x[i+1])/2.0
                y_first = (y[i]+y[i+1])/2.0
                for k in range (N):
                        if i == k:
                                matrix_[i][k] = -np.pi
                        else:
                                xa = x[k] - x_first
                                xb = x[k+1] - x_first
                                ya = y[k] - y_first
                                yb = y[k+1] - y_first
                                matrix_[i][k] = -np.arccos((xa*xb + ya*yb)/(np.sq
        return matrix_


def integral_1(N,r_a,r_b,direction1):
        x,y = make_points(N,r_a,r_b)
        traps1 = 0
        traps2 = 0
        Matrix_B = np.zeros(N)
        for i in range(N):
                integral_x = 0
                integral_66 = 0
                x0 = 0.5*(x[i]+x[i+1])
                y0 = 0.5*(y[i]+y[i+1])
                for j in range(N-1):
                        rad_1 = np.sqrt((x0-x[j])**2 + (y0-y[j])**2)
                        rad_2 = np.sqrt((x0-x[j+1])**2 + (y0-y[j+1])**2)

                        if direction1 == 11:
                                traps1x = -((x[j]+x[j+1])/(2*r_a**2)) / np.sqrt(
((x[j]+x[j+1])/(2*r_a**2))**2 +  ((y[j]+y[j+1])/(2*r_b**2))**2  ) * np.log(rad_1)
                                traps2x = -((x[j+1]+x[j+2])/(2*r_a**2)) / np.sqrt
((x[j+1]+x[j+2])/(2*r_a**2))**2 + ((y[j+1]+y[j+2])/(2*r_b**2))**2  )* np.log(rad_
                                ds = np.sqrt((x[j+1]-x[j])**2 + (y[j+1] - y[j])**
                                integral_x = integral_x + (traps1x + traps2x) * d
* 0.5
```

```
                        else :
                                r1a = (x[j] + x[j+1])*0.5 ; r2a =0.5*(y[j] + y[j+
                                n1a = -((x[j]+x[j+1])/ (2*r_a**2)) / np.sqrt(
((x[j]+x[j+1])/(2*r_a**2))**2 + ((y[j]+y[j+1])/(2*r_b**2))**2  )
                                n2a = -((y[j]+y[j+1])/(2*r_b**2))/np.sqrt(
((x[j]+x[j+1])/(2*r_a**2))**2 + ((y[j]+y[j+1])/(2*r_b**2))**2  )
                                crosses1 = r1a*n2a - r2a*n1a

                                r1b = (x[j+1] + x[j+2])*0.5 ; r2b =0.5*(y[j+1] + 
                                n1b = -((x[j+1]+x[j+2])/(2*r_a**2))/np.sqrt(
((x[j+1]+x[j+2])/(2*r_a**2))**2 + ((y[j+1]+y[j+2])/(2*r_b**2))**2  )
                                n2b = -((y[j+1]+y[j+2])/(2*r_b**2))/np.sqrt(
((x[j+1]+x[j+2])/(2*r_a**2))**2 + ((y[j+1]+y[j+2])/(2*r_b**2))**2  )
                                crosses2 = r1b*n2b - r2b*n1b

                                traps3 = np.log(rad_1) * crosses1
                                traps4 = np.log(rad_2) * crosses2
                                ds = np.sqrt((x[j+1]-x[j])**2 + (y[j+1] - y[j])**
                                integral_66 = integral_66 + (traps3 +traps4)* ds
* 0.5
                if direction1  == 11:
                        Matrix_B[i] = integral_x
                else :
                        Matrix_B[i] = integral_66
        return Matrix_B


def solver1(N,r_a,r_b,direction1):
        return np.linalg.solve(make_angle(N,r_a,r_b),integral_1(N,r_a,r_b,directi

def added_mass(N,r_a,r_b):
        phix = solver1(N,r_a,r_b,direction1 =11)
        phi6 = solver1(N,r_a,r_b,direction1 =66)
        x,y = make_points(N,r_a,r_b)
        add_m11 = 0
        add_m22 = 0
        add_m66 = 0
        for i in range(N-2):
                rad_1 = np.sqrt(  ((x[i]+x[i+1])/(2*r_a**2))**2 +  ((y[i] +y[i+1]
                rad_2 = np.sqrt(  ((x[i+1]+x[i+2])/(2*r_a**2))**2 +
((y[i+1]+y[i+2])/(2*r_b**2))**2)
                ds = np.sqrt((x[i+1]-x[i])**2 + (y[i+1] - y[i])**2)

                traps1 = phix[i] * -(x[i]+x[i+1])/(2*r_a**2)/rad_1
                traps2 = phix[i+1] * -(x[i+1]+x[i+2])/(2*r_a**2)/rad_2

                r1x = (x[i] + x[i+1])*0.5 ; r2x = 0.5*(y[i] + y[i+1])
                n1x = -(x[i] + x[i+1])/(2*r_a**2)/rad_1
                n2x = -(y[i] + y[i+1])/(2*r_b**2)/rad_1
                crosses1 = r1x*n2x - r2x*n1x

                r1y = (x[i+1] + x[i+2])*0.5 ; r2y = 0.5*(y[i+1] + y[i+2])
                n1y = -(x[i+1] + x[i+2])/(2*r_a**2)/rad_2
                n2y = -(y[i+1] + y[i+2])/(2*r_b**2)/rad_2
                crosses2 = r1y*n2y - r2y*n1y
```

```
                traps3 = phi6[i] * crosses1
                traps4 = phi6[i+1] * crosses2

                add_m11 = add_m11 + (traps1 + traps2)*ds * 0.5
                add_m66 = add_m66 + (traps3 + traps4)*ds *0.5
        return add_m11, add_m66



N = 1000
r_a = 1
r_b = 3
equal = r_a - r_b


if equal == 0:
        a_m11,a_m66 = added_mass(N,r_a,r_b)
        error1 = ((((r_a**2*np.pi)-float(a_m11))/(r_a**2*np.pi))*100)
        print "m11= %.3f m66= %.f,m11 error in percent: %.2f  %% " %(a_m11,a_m66,
        thet = np.zeros(N)
        om = 2.0*pi*r_a
        dx = 2*pi/N
        for i in range(N):
                thet[i] = i*dx
        solution = solver1(N,r_a,r_b,direction1 = 11)
        exact_1 = -np.cos(thet)
        diff = max(exact_1 - solution)
        plot(exact_1,"g")
        plot(solution, "r")
        plt.legend(['Exact ', 'Numerical'])
        title("Fluid Potential over Circle, running %.d times,  error: %.2f %%
\n m11 = %.2f, m66 = %.f , error in m11 =  %.2f %% " %(N,100*diff/max(exact_1), a
        show()

else :
        a_m11,a_m66 =  added_mass(N,r_a,r_b)
        error1 = (((r_b**2*np.pi)-a_m11)/(r_b**2*np.pi))*100
        exact6 = (np.pi/8)*(r_a**2-r_b**2)**2
        error6 = (((exact6-a_m66 ) / (exact6)))*100
        print "m11: " ,a_m11 ,"error m11 : %.2f  %%, error m66 %.2f %%
running %.d times" %(error1,error6, N)
        plot(solver1(N,r_a,r_b,direction1 = 11), "r")
        title("Distribution of potential over an ellipse running %.d times \n m11
" %(N,a_m11,error1,a_m66,error6,r_a,r_b))
        show()
```



This shows a plot of ϕ over a circle, with the added mass and errors calculated.
We can see that the error of ϕ compared to the exact solution is 0.74% and the added mass m_{66} has an error of 0.73% . The added mass m_{66} is of course zero because we have a circle. With these small error i am convinced that the program works for a circle. Next up an ellipse:



This plot is over an ellipse. Again we can see that the errors are very small 0.87% and 0.37% With both of our geometries having so small error compared to the analytics, I am convinced that the program is correct and the potential and added masses are correctly calculated. I have not calculated the added mass m_{22}, m_{12}, m_{21} as these will all be zero since the geometry only travels laterally.
Since i am not very good in Latex , the code does not look very good copied in. Please let me know if you want the Python code.