MEK 4420
Mandatory Assignment

Andreas Slyngstad

15. september 2015

Introduction of problem

In this assigment, the following exercise was presented:

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*a*x/(r*r) where a denotes the cylinder radius, r*r=x*x+y*y.

Theory

From our study of **Green's Theorem and Distribution of Singularities** in J.N. Newmans **Marine Hydrodynamics**, the following important result where presented.
By assuming potential theory, let two potentials ϕ and ψ be solutions of Laplace's equation for a fluid in a volume $\boldsymbol{V}$ within a closed surface $\boldsymbol{S}$. We now substitue ψ with the potential for a unit source G, where the source point $\xi = (\xi,\ \eta,\ \varsigma)$ is either located outside, on or withing the boundary $\boldsymbol{S}$. Now we assume a body moving within the fluid, expanding the volume $\boldsymbol{V}$ so the fluid can be seen as infinite and unbound. For a large spherical radius, the surfaceintegral of the boundary $\boldsymbol{S}$ at infinite can be neglected, hence we only need to consider the surface where the fluid is in contact with the body.

$$\iint\limits_S \Big(\phi\frac{\partial G}{\partial n} - G\frac{\partial \phi}{\partial n}\Big)dS = \begin{Bmatrix} 0 \\ -2\pi\phi(x,y,z) \\ -4\pi\phi(x,y,z) \end{Bmatrix}$$

$$G(x,y,z,\xi,\eta,\varsigma) = \frac{1}{r} + H(x,y,z,\xi,\eta,\varsigma)$$

Where the result represents if (x,y,z) is either outside, on or inside the boundary $\boldsymbol{S}$. Here the function G represents a ***Green Function***, while H is a solution of the Laplace equation, and can be used to suit different types of boundary conditions.
In our assignment we are supposed to study different types of geometry in 2D, the relation then yeilds

$$\iint\limits_S \Big(\phi\frac{\partial G}{\partial n} - G\frac{\partial \phi}{\partial n}\Big)dS = \begin{Bmatrix} 0 \\ -\pi\phi(x,y,z) \\ -2\pi\phi(x,y,z) \end{Bmatrix}$$

Where $\boldsymbol{S}$ now is the surface of our 2D geometry

Numerical Approach

We take a closer look at our line integral, where we want to evaluate (x,y) on the surface $\boldsymbol{S}$. Here we will take S to be a circle, which has been numerical implemented of several segments C_s

$$-\pi\phi(x_0) = \int_S \Big(\phi\frac{\partial\psi}{\partial n} - \psi\frac{\partial\phi}{\partial n}\Big)dS$$

Here ψ = ln r, which is the source potential in 2D. Furthermore, we consider ϕ and $\frac{\partial\phi}{\partial n}$ to be constant on each linesegment.

$$\pi\phi(X_0) + \sum_{n=1}^{N}\phi(X_n)\int_{Cs}\frac{\partial}{\partial n}lnr\ dS = \sum_{n=1}^{N}\frac{\partial\phi}{\partial n_X}\int_{Cs} lnr\ dS$$

Taking a closer look on our integral in the second term, we exploit a trick with complex numbers. Using the relations

$$\begin{aligned}
ln(z) &= ln(re^{i\theta}) = ln\ r + i\theta \\
ds &= \sqrt{dx^2+dy^2}\ ; n_2\ dS = dx\ ; -n_1 dS = dy \\
\int_{Cs}\frac{\partial}{\partial n_x}lnr\ dS &= Re\int_A^B \big(n_1\frac{\partial}{\partial x} + n_2\frac{\partial}{\partial y}\big)ln\ z\sqrt{dx^2+dy^2} = Re\int_A^B\frac{n_1+in_2}{z}\sqrt{dx^2+dy^2} \\
&= Re\int_A^B\frac{-dy+idx}{z}\sqrt{dx^2+dy^2} = Re\ i\int_A^B\frac{dz}{z} = Re\Big(i\ ln\ z\Big|_A^B\Big) \\
&= -\Big(\theta_B - \theta_A\Big)
\end{aligned}$$

Hence the integral can be interpreted as the angle between the node points where each potential is located. It is important to understand that each potential is seen as the average between the node points, and therefore located as ghost point between the node points. For our last integral we look at

$$\sum_{n=1}^{N}\frac{\partial\phi}{\partial n_X}\int_{Cs} lnr\ dS$$

x

Again it is important to notice that we operate with two reference systems. Firstly the term $\frac{\partial\phi}{\partial n_X}$ is the normal component of ϕ at the line segment, where the normal vector is on that line segment. On the other hand the coordinate $r = \sqrt{(x-x')^2+(y-y')^2}$, we let the coordinates (x, y) be fixed, and move around the geometry with (x', y'). Hence each integral the $ln\ r$ part is taken with respect from the ϕ beeing calculated on that perticular line segment. We are only interested in lateral and with rotaton in 2D. We use the relation

$$\begin{aligned}
\frac{\partial\phi_i}{\partial n} &= n_i \quad i = 1,2; \qquad \frac{\partial\phi_i}{\partial n} = (\mathbf{r'}\times\mathbf{n})_{i-3} \quad i = 6 \\
\frac{\partial\phi_1}{\partial n} &= n_1 = cos\theta = \frac{x}{r} \qquad \frac{\partial\phi_6}{\partial n} = (\mathbf{r'}\times\mathbf{n})_3
\end{aligned}$$

Our goal now is to end up linear system on the following form

$$\begin{Bmatrix} \pi & (\theta_1-\theta_2) & (\theta_2-\theta_3)\cdots \\ (\theta_{N-1}-\theta_N) & \pi & (\theta_1-\theta_2)\cdots \\ (\theta_{N-2}-\theta_{N-1}) & (\theta_{N-1}-\theta_N) & \pi\cdots \\ & \vdots & \end{Bmatrix} \begin{Bmatrix} \phi(x_0) \\ \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_N) \end{Bmatrix} = \begin{Bmatrix} \frac{\partial\phi}{\partial n}\int\limits_{C1} lnr_1\, dS \\ \frac{\partial\phi}{\partial n}\int\limits_{C2} lnr_2\, dS \\ \vdots \\ \frac{\partial\phi}{\partial n}\int\limits_{CN} lnr_N\, dS \end{Bmatrix}$$

By exploiting the relations presented above into the integrals, we are able to calculate the *added mass tensor*

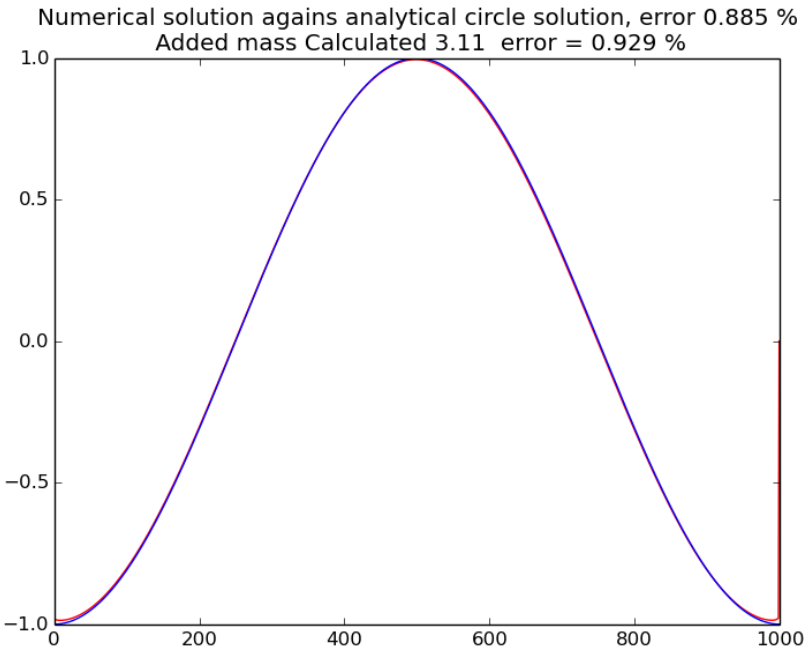
$$\begin{Bmatrix} m_{11} & m_{12} & m_{16} \\ m_{21} & m_{22} & m_{26} \\ m_{61} & m_{62} & m_{66} \end{Bmatrix}$$

We can argue that because of lateral movement

$$\begin{Bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & 0 \\ 0 & 0 & m_{66} \end{Bmatrix}$$

Results

Numerical approximation of m11, Circle N=1000



For a circle, the numerical results are very good. We end up with a relative error of 0.885 % for the velocity potential on the boundary of the circle. For the added mass in lateral direction m_{11}, we end up with a relative error of 0.929 %. These results gives a good indication that the numerical estimation for the circle is pretty accurate. For the yaw m_{66}, we execpt the analytical solution to be 0, due to the cross product $\mathbf{r'} \times \mathbf{n}$ is zero for a circle. This is verified from the numerical calculation

```
Exact value: 0.00000, Calculated mass: 0.00000
```

For the rest of the numerical calculations the results will be compared up against the added mass tensor, due to the lack of an exact analytical solution for the velocity potensial.
For an ellipse

$$(\frac{x}{a})^2 + (\frac{y}{b})^2 = 1$$

I have chosen $a = 1, b = 3$, and as for the circle, N = 1000 elements have been chosen for good precition. The analytical solution of the *mass tensor* components are presented in newman as

- $m_{11} = \rho\pi b^2$
- $m_{66} = \frac{1}{8}\rho\pi(b^2 - a^2)^2$

A run of the numerical program gives the following result for the tensor components

```
Exact value: 28.27433, Calculated mass: 27.94521, error: 1.16402 %
Exact value: 25.13274, Calculated mass: 25.03932, error: 0.37172 %
```

With a relative error at the most $\approx 1\%$, these are some satisfying results, pointing in the direction of a decent numerical implementation.

Conclusion

With help from the analytical solutions, I conclude that the implementation is decent for the presented problem. The relative error is small, which conclude a valid implementation. I take criticism in implementing to many for-loops, which results in long runtime of the program for large N. I will try to implement a vectorized implementation for my next assignment.

```
import numpy as np
import matplotlib.pyplot as plt

def points(N, a, b):
        if a == b:
                o = 2.0*np.pi*a
        else:
                o = 2*np.pi

        dx = o/N
        x = np.zeros(N)
        y = np.zeros(N)
        for i in range(N):
                x[i] = a*np.cos(dx*i)
                y[i] = b*np.sin(dx*i)
        return x, y

def theta(N, a, b, j):
        x, y = points(N, a, b)
        theta = np.zeros(N)
        initx = (x[j+1] + x[j])*0.5
        inity = (y[j+1] + y[j])*0.5
        for i in range(len(theta)):
                if i == len(theta)-1:
                        xa = x[i] - initx; xb = x[0] - initx
                        ya = y[i] - inity; yb = y[0] - inity
                        theta[i] = np.arccos((xa*xb + ya*yb)/(np.sqrt(xa**2 + ya*
                else:
                        xa = x[i] - initx; xb = x[i+1] - initx
                        ya = y[i] - inity; yb = y[i+1] - inity
                        test = (xa*xb + ya*yb)/(np.sqrt(xa**2 + ya**2) * np.sqrt(
                        if test < -1.0: #Due too round off error
                                theta[i] = np.arccos(-1.0)
                        #if test > 1.0: #Due too round off error
                        #       theta[i] = np.arccos(1.0)
                        else:
                                theta[i] = np.arccos(test)
        return theta

def integrate(N, eq, direction,a,b):
        area = 0
        x, y = eq
        integral = np.zeros(N)
        if direction == 11:
                num = x
        if direction == 22:
                num = y
        for i in range(N-1):
                x_0 = 0.5*(x[i]+x[i+1])
                y_0 = 0.5*(y[i]+y[i+1])

                for j in range(N-2):
                        ds = np.sqrt((x[j+1]-x[j])**2 + (y[j+1]-y[j])**2)
                        #if x_0 == x[j] and y_0 == y[j] or x_0 == x[j+1] and y_0
                        #       continue
```

```
                        radius_a = np.sqrt((x_0 -x[j])**2 + (y_0 - y[j])**2)
                        radius_b = np.sqrt((x_0 -x[j+1])**2 + (y_0 - y[j+1])**2)

                        if direction == 66:
                                n1_a = -((x[j]+x[j+1])/(2*a**2))/np.sqrt(((x[j+1]
                                n2_a = -((y[j]+y[j+1])/(2*b**2))/np.sqrt(((x[j+1]
                                rx_a = 0.5*(x[j]+x[j+1]); ry_a = 0.5*(y[j]+y[j+1]
                                n_a = np.cross([n1_a,n2_a],[rx_a,ry_a])

                                n1_b = -((x[j+1]+x[j+2])/(2*a**2))/np.sqrt(((x[j+
                                n2_b = -((y[j+1]+y[j+2])/(2*b**2))/np.sqrt(((x[j+
                                rx_b = 0.5*(x[j+1]+x[j+2]); ry_b = 0.5*(y[j+1]+y[
                                n_b = np.cross([n1_b,n2_b],[rx_b,ry_b])

                                f_a = np.log(radius_a) * n_a
                                f_b = np.log(radius_b) * n_b

                        else:
                                na = -(num[j]+num[j+1])/(2*a**2)/np.sqrt(((x[j]+x
                                nb = -(num[j+1]+num[j+2])/(2*a**2)/np.sqrt(((x[j+
                                f_a = np.log(radius_a) * na
                                f_b = np.log(radius_b) * nb

                        area = area + 0.5*(f_a + f_b)*ds
                integral[i] = area
                area = 0
        return integral

def matrix(N, a, b):
        mat = np.zeros((N,N))
        for i in range(N):
                if i == N-1:
                        mat[i][:] = -theta(N, a, b, -1)
                else:
                        mat[i][:] = -theta(N, a, b, i)
        return mat

def added(sol, N, a,b, direction):
        x, y = points(N,a,b)
        area = 0
        if direction == 11:
                num = x*a**-2
                for i in range(N-2):
                        na = -0.5*(num[i]+num[i+1])/np.sqrt((((x[i+1]+x[i])/(2*a*
                        nb = -0.5*(num[i+1]+num[i+2])/np.sqrt((((x[i+2]+x[i+1])/(
                        ds = np.sqrt((x[i+1]-x[i])**2 + (y[i+1]-y[i])**2)
                        area = area + 0.5*(sol[i]*na+sol[i+1]*nb)*ds
                return area
        if direction == 22:
                numerator = y*b**-2
                for i in range(N-1):
                        n1 = -numerator[i]/np.sqrt((x[i+1]**2+y[i+1]**2))
                        ds = np.sqrt((x[i+1]-x[i])**2 + (y[i+1]-y[i])**2)
                        area = area + 0.5*(sol[i]*n1+sol[i+1]*n1)*ds
                return area
```

```
        if direction == 66:
                for i in range(N-2):
                        n1a = -((x[i]+x[i+1])/(2*a**2))/np.sqrt(((x[i+1]+x[i])/(2
                        n2a = -((y[i]+y[i+1])/(2*b**2))/np.sqrt(((x[i+1]+x[i])/(2
                        rxa = 0.5*(x[i]+x[i+1]); rya = 0.5*(y[i]+y[i+1]);
                        na = np.cross([n1a,n2a],[rxa,rya])

                        n1b = -((x[i+1]+x[i+2])/(2*a**2))/np.sqrt(((x[i+2]+x[i+1]
                        n2b = -((y[i+1]+y[i+2])/(2*b**2))/np.sqrt(((x[i+2]+x[i+1]
                        rxb = 0.5*(x[i+1]+x[i+2]); ryb = 0.5*(y[i+1]+y[i+2]);
                        nb = np.cross([n1b,n2b],[rxb,ryb])


                        ds1 = np.sqrt((x[i+1]-x[i])**2 + (y[i+1]-y[i])**2)
                        ds2 = np.sqrt((x[i+2]-x[i+1])**2 + (y[i+2]-y[i+1])**2)
                        area = area + 0.5*(sol[i]*na*ds1 + sol[i+1]*nb*ds2)
                return area


def result(N, a, b, direction):
         return np.linalg.solve(matrix(N,a,b), integrate(N, points(N,a,b),directio

def main():
        N = 500
        a = 4.0
        b = 4.0
        angle = np.zeros(N)
        direction = 11

        solve = result(N, a, b, direction)
        mass =  added(solve, N,a,b, direction)
        #exact = 4.753 square

        if a == b:
                if direction == 66:
                        exact = 0
                        print "Exact value: %.5f, Calculated mass: %.5f" % (exact
                else:
                        for i in range(N):
                                angle[i] = i*2*np.pi/N
                        error = max(-np.cos(angle)-solve)/max(-np.cos(angle))*100
                        masse = float((a**2*np.pi-mass)/np.pi * 100)
                        plt.plot(solve, 'r')
                        plt.plot(-np.cos(angle))
                        plt.title("Numerical solution agains analytical circle so
error = %1.3f %% " %(error, mass, masse))
                        plt.show()

        else:
                if direction == 66:
                        exact = (np.pi/8.0)*(a**2-b**2)**2
                        masse = float(exact - mass)/(exact) * 100.0
                        print "Exact value: %.5f, Calculated mass: %.5f, error: %
                else :
                        exact = np.pi*b**2
                        masse = float((np.pi*b**2 - mass)/(np.pi*b**2) * 100.0)
```

```
                    print "Exact value: %.5f, Calculated mass: %.5f, error: %

if __name__ == "__main__":
        main()
```