Assignment Report - 3 (Revised Version 2)

Green's Boundary Integral Method for a Cylindrical Heaving Buoy

Aditya Narayanan (OE13S013) *

Abstract

Green's boundary integral method or panel method is used to obtain the added mass and radiation damping of a heaving cylinder in a small amplitude wave regime.

1 Problem Statement

Using the frequency-domain boundary integral method with $G = \frac{1}{r}$ numerically solve the heave radiation problem of a vertical circular cylinder.

1. Cylinder diameter, D = 1 m
2. Cylinder draft, d = 3 m
3. Water depth, h = 10 m
4. Water depth, h = 10 m
5. Frequency, σ from 0.1 to 5 [rad/s]

and also compute the heave added-mass μ_{zz} and damping λ_{zz} with total number of panels N=5000 and 10000.

2 Formulation

Since the cylinder is small compared to the incident wave's wavelength ($D/L << 1$), we neglect the scattering wave potential in this solution. We treat this as a linear problem, ie. it is assumed that the wave has a small enough amplitude to neglect non-linearities, which allows us to decompose the total fluid potential into separate components.

$$\phi = \phi^i + \phi^r$$

The radiation potential must satisfy the following:

$$\nabla^2 \Phi^r = 0, \text{ in } \forall$$

We apply the boundary conditions for the radiation potential at the body surface:

$$\frac{\partial \Phi^r}{\partial n} = V_n$$

$$\text{where, } V_n = \vec{U}.\hat{n} + \vec{\Omega}.(\vec{r} \times \hat{n})$$

*email: adityarn@gmail.com

The free surface boundary condition applied is:

$$\frac{\partial^2 \Phi^r}{\partial t^2} + g\frac{\partial \Phi^r}{\partial z} = 0, \text{ on } z = 0$$

The radiation potential can be computed in the frequency domain by separating the time function:

$$\Phi^r = \sum_{j=1}^{6} \phi_j^r = e^{-i\sigma t} \sum_{j=1}^{6} \varphi_j^r$$

If we define a displacement ξ_j as the displacement due to the j^{th} mode of oscillation, then:

$$\xi_j = A_j e^{-i\sigma t}$$

The velocity and acceleration would then be:

$$\frac{d\xi_j}{dt} = -i\sigma t A_j e^{-i\sigma t}$$

$$\frac{d^2\xi_j}{dt^2} = -\sigma^2 t A_j e^{-i\sigma t}$$

The radiation potential can be represented per unit displacement, as:

$$\varphi_j^r = A_j \psi_j^r$$

This radiation potential satisfies the following conditions:

$$\nabla^2 \psi_j^r = 0, \text{ in } \forall$$

$$\nabla \psi_j^r \to 0, \text{ as } z \to -\infty$$

Free surface condition:

$$-\sigma^2 \psi_j^r + g\frac{\partial \psi_j^r}{\partial z} = 0, \text{ on } z = 0$$

Sommerfeld radiation condition:

$$-i\sigma \psi_j^r + \frac{\sigma}{K}\frac{\partial \psi_j^r}{\partial R} = 0, \text{ at } R \to \infty$$

The force on the body, (obtained by integrating the dynamic pressure over the body surface):

$$F_{jk} = -\big[-\sigma^2 A_j e^{-i\sigma t} \mu_{jk} - i\sigma A_j e^{-i\sigma t} \lambda_{jk} \big]$$

The force per unit amplitude and in frequency domain will be:

$$f_{jk} = -\sigma^2 \mu_{jk} - i\sigma \lambda_{jk}$$

where,

$$f_{jk} = \rho \int_{S_0} \psi_j^r \frac{\partial \psi_k^r}{\partial n} dS_0$$

$$\mu_{jk} = \frac{\text{Real}(f_{jk})}{-\sigma^2} \text{ and } \lambda_{jk} = \frac{Imag(f_{jk})}{-\sigma}$$

3 Methodology

The Green's mixed distribution method is used, and by using the appropriate boundary conditions for the radiation potential, we get the following

$$\begin{aligned} 2\pi\varphi(P) + \int_{S_0} \varphi\Big(\frac{\partial}{\partial n}\frac{1}{r} dS_0\Big) \quad &+ \quad \int_{F_0} \varphi\Big(\frac{\partial}{\partial n}\frac{1}{r} - \frac{1}{r}\frac{\sigma^2}{g}\Big) dF_0 + \int_B \varphi\Big(\frac{\partial}{\partial n}\frac{1}{r}\Big) dB \\ &+ \quad \int_\Sigma \varphi\Big(\frac{\partial}{\partial n}\frac{1}{r} - \frac{1}{r} ik\Big) d\Sigma \;= \int_{S_0} \frac{1}{r} V_n dS_0 \end{aligned} \tag{1}$$

Where, S_0 is the body's surface, F_0 is the free surface, B is the bottom surface, and Σ is the far-field surface.

For the j'th mode of operation,

$$V_n = \frac{\partial \varphi_j^r}{\partial n} = -i\sigma n_j$$

The above system of equations can be represented as:

$$C_{ij}\varphi_j = b_j$$

The system when solved gives us the radiation potential at each panel on all the four surfaces. The forces when integrated over the surface of the oscillating body (using the potentials obtained by solving the equations) give us the added mass and damping.

The mesh used in the solution is shown in fig 1.

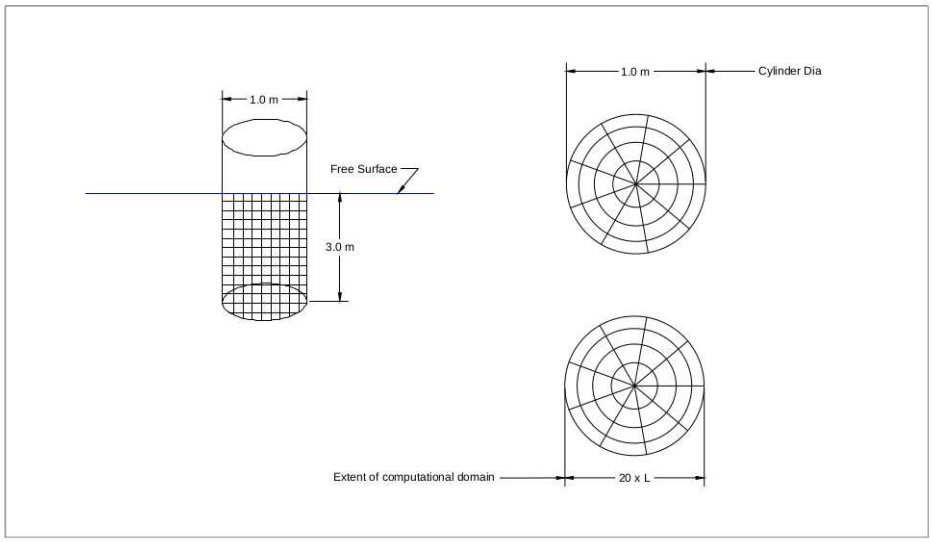

Figure 1: Mesh for cylinder, free surface, far-field, and bottom surface

4 Results

The table shows the final results obtained. As is evident, the damping is very low, and the added mass is about 1.02 to 1.03.

$$\text{(Mass displaced by buoy)} = \rho \times \pi \times 1^2 \times 3.0 = 9660.4Kg$$

$$\text{Added Mass coefficient}, C_a = \frac{\text{(added mass)}}{\text{(added mass + mass displaced)}}$$

Table 1: Results - Mu and Damping coefficients in a non-dimensionalized form

Total no of panels	**no of body panels**	**Sigma**	**Mu**	**Damping**
6191	1250	0.628	1.0306243785	1.0000068937
6191	1250	5	1.0249887263	1.0000433711

5 Program Listing

5.1 C++ Program to Generate Coordinates

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<iomanip>
#include<time.h>
#include <vector>

using namespace std;

//Class with all panel properties (x,y,z,nx,ny,nz,area)
class Panel
{
public:
  double coord[3],norm[3],area;

  int set_area(double x, double y)
  {
    area = x*y;
    return 1;
  }

  double rij(double coordj[3])
  {
    return sqrt(pow(coord[0]-coordj[0],2)+pow(coord[1]-coordj[1],2)+pow(coord[2]-
        coordj[2],2));
  }
};

class Green
{

public:

  // Constructor to initialise values
  // Time_period is that of incoming wave
  Green(double Time_Period, double Depth, double Draft, double Dia, int Nbody,
      int Nf, int Nb, int Nff)
  {
    Time_period = Time_Period;

    // n_body is the number of panels on the surface of the body
    // Ensure that Ns is 2 x (number with exact square root)
    n_body = Nbody;

    // n_fs is the number of panels on the free surface
    n_fs = Nf;

    // n_bottom is the number of panels on the bottom of the domain (flat sea bed
        )
```

```
    n_bottom = Nb;

    // n_on_ff is the number of far field panels
    n_ff = Nff;

    n_total = Nbody + Nf + Nb + Nff;

    // xyz.coord[] holds x,y,z coordinates of control points of each panel
    xyz.resize(n_total);

    draft = Draft;
    pi = 4*atan(1.0);
    rad = Dia*0.5;

    sigma = 2*pi/Time_period;

    depth = Depth;
  }

  int Solve()
  {
    clock_t t1;
    t1 = clock();

    iterate_L();
    coord_set_body();
    coord_set_fs();
    coord_set_bb();
    coord_set_ff();

    write_coord();
    cout<<"\nAll body coords set!\n";

    return 1;
  }

protected:
  double pi, rad, draft, domain_rad, sigma, L, depth, Time_period;
  int n_body, n_fs, n_bottom, n_ff, n_total;
  vector <Panel> xyz;

  int iterate_L()
  {
    double Lo, tL;

    Lo = 1.56 * pow(Time_period,2);
    L = Lo;
    tL = 0;

    while( abs(L - tL) >= 1e-8)
      {
        tL = L;
```

```
      L = Lo * tanh(2*pi/tL * depth);
    }

  cout<<"\nLo = "<<Lo<<"\nL = "<<L<<"\n";
  return 1;
}

// Function which sets the control points
// coordinate to x[], y[], z[]
int coord_set_body()
{
  double dy, dtheta, dr, denom;
  int ndtheta, ndy, ndr, c;

  c = 0;
  ndy = int(pow(n_body*.5,0.5));
  ndtheta = ndy;
  dtheta = 2*pi/ndtheta;
  dy = draft/ndy;

  for(int i=0; i<ndy; i++)
    {
      for(int j=0; j<ndtheta; j++)
        {
          xyz[c].coord[0] = rad * cos(0.5*dtheta + j*dtheta);
          xyz[c].coord[1] = rad * sin(0.5*dtheta + j*dtheta);
          xyz[c].coord[2] = -(dy * 0.5 + i * dy);
          xyz[c].set_area(rad * dtheta, dy);
          denom = sqrt(4*pow(xyz[c].coord[0],2) + 4 * pow(xyz[c].coord[1],2));
          xyz[c].norm[0] = -2 * xyz[c].coord[0] / denom;
          xyz[c].norm[1] = -2 * xyz[c].coord[1] / denom;
          xyz[c].norm[2] = 0.0;
          c++;
        }
    }

  if(c != ndy * ndtheta)
    {
      cout<<"Check body discretization! c .ne. (ndy x ndtheta), c = "<<c<<"\n";
      //      return 0;
    }

  // Bottom (of body) discretization
  ndr = ndy;
  dr = rad / ndr;
  cout<<"\ndr = "<<dr<<"\nndr = "<<ndr<<"\n";
  for(int i=0; i<ndr; i++)
    {
      for(int j=0; j<ndtheta; j++)
        {
          xyz[c].coord[0] = (0.5*dr + i*dr) * cos(0.5*dtheta + j*dtheta);
          xyz[c].coord[1] = (0.5*dr + i*dr) * sin(0.5*dtheta + j*dtheta);
```

```
            xyz[c].coord[2] = -draft;
            xyz[c].set_area((0.5*dr + i*dr)*dtheta,dr);
            xyz[c].norm[0] = 0.0;
            xyz[c].norm[1] = 0.0;
            xyz[c].norm[2] = 1.0;
            c++;
          }
      }

    return 1;

}

int coord_set_fs()
{
    double dtheta, dr, ndtheta, ndr;
    int c = n_body;

    ndr = sqrt(n_fs);
    ndtheta = ndr;

    dtheta = 2*pi/ndtheta;
    dr = (10*L) / ndr;

    for(int i= 0; i < ndtheta; i++)
      {
        for(int j=0; j<ndr; j++)
          {
            xyz[c].coord[0] = (0.5 * dr + j*dr) * cos(0.5*dtheta + i*dtheta);
            xyz[c].coord[1] = (0.5 * dr + j*dr) * sin(0.5*dtheta + i*dtheta);
            xyz[c].coord[2] = 0;

            xyz[c].set_area( (0.5 * dr + j*dr) * dtheta , dr);

            xyz[c].norm[0] = 0;
            xyz[c].norm[1] = 0;
            xyz[c].norm[1] = 1.0;

            c++;
          }
      }
    return 1;
}

int coord_set_bb()
{
    double dtheta, dr, ndtheta, ndr;
    int c = n_body + n_fs;

    ndr = sqrt(n_fs);
    ndtheta = ndr;
```

```
  dtheta = 2*pi/ndtheta;
  dr = (10*L) / ndr;

  for(int i= 0; i < ndtheta; i++)
    {
      for(int j=0; j<ndr; j++)
        {
          xyz[c].coord[0] = (0.5 * dr + j*dr) * cos(0.5*dtheta + i*dtheta);
          xyz[c].coord[1] = (0.5 * dr + j*dr) * sin(0.5*dtheta + i*dtheta);
          xyz[c].coord[2] = -depth;

          xyz[c].set_area( (0.5 * dr + j*dr) * dtheta , dr);
          xyz[c].norm[0] = 0;
          xyz[c].norm[1] = 0;
          xyz[c].norm[1] = -1.0;

          c++;
        }
    }
  return 1;
}

int coord_set_ff()
{
  double dtheta, dy, ndtheta, ndy, denom;
  int c = n_body + n_fs + n_bottom;

  ndy = int(pow(n_ff , 0.5));
  ndtheta = ndy;
  dtheta = 2*pi/ndtheta;
  dy = depth/ndy;

  for(int i=0; i<ndy; i++)
    {
      for(int j=0; j<ndtheta; j++)
        {
          xyz[c].coord[0] = 10 * L * cos(0.5*dtheta + j*dtheta);
          xyz[c].coord[1] = 10 * L * sin(0.5*dtheta + j*dtheta);
          xyz[c].coord[2] = - (dy * 0.5 + i * dy);

          xyz[c].set_area(L/10 * dtheta, dy);
          denom = sqrt(4*pow(xyz[c].coord[0],2) + 4 * pow(xyz[c].coord[1],2));

          xyz[c].norm[0] = 2 * xyz[c].coord[0] / denom;
          xyz[c].norm[1] = 2 * xyz[c].coord[1] / denom;
          xyz[c].norm[2] = 0.0;
          c++;
        }
    }

  if( (c - n_body - n_fs - n_bottom) != ndy * ndtheta - 1)
    {
```

```
        cout<<"Check far field discretization! c .ne. (ndy x ndtheta - 1), c =
            "<<c<<"\n";
        //      return 0;
      }

    return 1;
  }

  void write_coord()
  {
    ofstream coord;
    coord.open("xyz.txt");
    coord<<Time_period<<" "<<L<<" "<<n_total<<" "<<n_body<<" "<<n_fs<<" "<<
        n_bottom<<" "<<n_ff<<"\n";
    for(int i=0; i<n_total; i++)
      coord<<xyz[i].coord[0]<<" "<<xyz[i].coord[1]<<" "<<xyz[i].coord[2]<<" "<<
          xyz[i].norm[0]<<" "<<xyz[i].norm[1]<<" "<<xyz[i].norm[2]<<" "<<xyz[i].
          area<<"\n";
  }

};

int main()
{

  int f=0;
  //  Green(double Time_period, double Depth, double Draft, double Dia, int Nbody
      , int Nf, int Nb, int Nff)
  // Nbody = 2 x (perfect square)
  // Nf = Nb = Nff = (perfect square)
  Green A(10.0, 30, 3.0, 1.0, 5000, 1681, 1681, 1681);

  A.Solve();

  return 1;
}
```

5.2 Python Code to Solve the Green's Mixed Distribution Equations

```
import numpy as np
import linecache

'''Reading From Input File "xyz.txt"
which contains all coordinates, normals
and area of each panel. This file is
generated with the C++ code "coord_gen.cpp" '''

c = 0
line = linecache.getline("xyz.txt",c+1)
line_arr = line.split()

Time_period = float(line_arr[0])
L = float(line_arr[1])
```

```
n_total = int(line_arr[2])
n_body = int(line_arr[3])
n_fs = int(line_arr[4])
n_bottom = int(line_arr[5])
n_ff = int(line_arr[6])

Kl = 2*np.pi/L
sigma = 2*np.pi/Time_period

x = np.zeros(n_total)
y = np.zeros(n_total)
z = np.zeros(n_total)
nx = np.zeros(n_total)
ny = np.zeros(n_total)
nz = np.zeros(n_total)
ar = np.zeros(n_total)
a = np.zeros((n_total,n_total),dtype=complex)
b = np.zeros(n_total,dtype=complex)
phi = np.zeros(n_total,dtype=complex)

for c in range(n_total):
    line = linecache.getline("xyz.txt",c+2)
    line_arr = line.split()
    x[c] = float(line_arr[0])
    y[c] = float(line_arr[1])
    z[c] = float(line_arr[2])
    nx[c] = float(line_arr[3])
    ny[c] = float(line_arr[4])
    nz[c] = float(line_arr[5])
    ar[c] = float(line_arr[6])

'''Green's mixed Distribution coefficients and RHS matrix are
computed here'''
for i in range(n_total):
    for k in range(n_body):
        if(i!=k):
            r = (x[i]-x[k])**2 + (y[i]-y[k])**2 + (z[i]-z[k])**2
            numer = -sigma * nz[k]* ar[k]
            denom = r**0.5
            b[i] += complex(0,numer/denom)

    for k in range(n_body):
        if(i != k):
            r = (x[i]-x[k])**2 + (y[i]-y[k])**2 + (z[i]-z[k])**2
            numer = ((x[i]-x[k])*nx[k] + (y[i]-y[k])*ny[k] + (z[i]-z[k])*nz[k])*
                ar[k]
            denom = r**1.5
            a[i][k] = complex(numer/denom,0)
        else:
            a[i][k] = complex(2*np.pi,0)

    for k in range(n_body,n_body+n_fs,1):
```

```
        if(i != k):
            r = (x[i]-x[k])**2 + (y[i]-y[k])**2 + (z[i]-z[k])**2
            numer = (x[i]-x[k])*nx[k] + (y[i]-y[k])*ny[k] + (z[i]-z[k])*nz[k]
            denom = r**1.5
            numer2 = sigma**2
            denom2 = 9.81 * r**0.5
            a[i][k] = complex((numer/denom - numer2/denom2)*ar[k],0)
        else:
            a[i][k] = complex(2*np.pi,0)

    for k in range(n_body+n_fs,n_body+n_fs+n_bottom,1):
        if(i != k):
            r = (x[i]-x[k])**2 + (y[i]-y[k])**2 + (z[i]-z[k])**2
            numer = ((x[i]-x[k])*nx[k] + (y[i]-y[k])*ny[k] + (z[i]-z[k])*nz[k])*
                ar[k]
            denom = r**1.5
            a[i][k] = complex(numer/denom,0)
        else:
            a[i][k] = complex(2*np.pi,0)

    for k in range(n_body+n_fs+n_bottom,n_total,1):
        if(i != k):
            r = (x[i]-x[k])**2 + (y[i]-y[k])**2 + (z[i]-z[k])**2
            numer = ((x[i]-x[k])*nx[k] + (y[i]-y[k])*ny[k] + (z[i]-z[k])*nz[k])*
                ar[k]
            denom = r**1.5
            numer2 = Kl/r
            a[i][k] = complex((numer/denom*ar[k]), -numer2*ar[k])
        else:
            a[i][k] = 2*np.pi

phi = np.linalg.solve(a,b)

fjk = complex(0.0,0.0)
for i in range(n_body):
    fjk +=  phi[i] * nz[i] * ar[i]

fjk = fjk * 1025. * sigma * complex(0,-1)

mu = fjk.real /(-sigma**2)
damping = fjk.imag/(-sigma)
```