Fluid-Surface Interaction of Vortices

Muhammad Abdullah Zafar Ghauri

5th April 2025

Abstract

This study analyzes the fluid-surface interaction of vortices produced by fluid at moving at speed of 0.2 ms^{-1} within the closed contour subjected to no-slip condition at the top, bottom, left and right side with dynamic viscosity of 0.01 mPa s and density of 1.0 g/L.

1 Introduction

Fluid–structure interaction, or commonly referred to as **FSI**, is the interaction of some movable or deformable structure with an internal or surrounding fluid flow [2], which can either be stable or oscillatory. In oscillatory interactions, the strain induced in the solid structure causes it to move such that the source of strain is reduced, and the structure returns to its former state only for the process to repeat. It is an important consideration in the design of many engineering systems,such as automobile, aircraft, spacecraft, engines and bridges. Failing to consider the effects of oscillatory interactions can be catastrophic, especially in structures comprising materials susceptible to fatigue. As far as analysis aspect of FSI is concerned, it is a complex and tedious procedure which cannot be evaluated analytically with ease, similar to multiphysics problems, hence the analysis can either be performed using numerical simulations or through experimentation. Research in the fields of computational fluid dynamics and computational structural dynamics is still ongoing but the maturity of these fields enables numerical simulation of fluid-structure interaction [8]. Two main approaches exist for the simulation of fluid–structure interaction problems:

- **Monolithic approach:** the equations governing the flow and the displacement of the structure are solved simultaneously, with a single solver
- **Partitioned approach:** the equations governing the flow and the displacement of the structure are solved separately, with two distinct solvers

The monolithic approach requires a code developed for this particular combination of physical problems whereas the partitioned approach preserves software modularity because an existing flow solver and structural solver are coupled. Moreover, the partitioned approach facilitates solution of the flow equations and the structural equations with different, possibly more efficient techniques which have been developed specifically for either flow equations or structural equations. Another promising method for solving FSI problems is through numerical solutions in which the Newton-Raphson method or fixed point iteration method can be used effectively, with the problems based on Newton-Raphson method are dealt through both partition [6, 5]and monolithic [3, 1, 4]approach. These methods solve the nonlinear flow equations and the structural equations in the entire fluid and solid domain with the Newton–Raphson method. The system of linear equations within the Newton–Raphson iteration can be solved without knowledge of the Jacobian with a matrix-free iterative method, using a finite difference approximation of the Jacobian-vector product. Whereas Newton–Raphson methods solve the flow and structural problem for the state in the entire fluid and solid domain, it is also possible to reformulate an FSI problem as a system with only the degrees of freedom in the interface's position as unknowns. This domain decomposition condenses the error of the FSI problem into a subspace related to the interface [7].The FSI problem can hence be written as either a root finding problem or a fixed point problem, with the interface's position as unknowns.

```
    close
clear
```

```
clc
%Specification of domain size and the cells in x,y direction with mesh
%resolution
xe=4.0; %Domain length in x direction
ye=2.0;  %Domain length in y direction
nx=200; %Cells in x direction
ny=100;  %Cells in y direction
nx_f=nx*2; %number of fine grid cells in x direction
ny_f=ny*2; %number of fine grid cells in y direction
dx=xe/nx; %Cell size in x direction
dy=ye/ny; %Cell size in y direction
dx_f=dx/2;  %Size of fine cell in x direction
dy_f=dy/2;  %Size of fine cell in y direction
xx=repmat(0:dx:xe,ny,1);
yy=repmat(0:dy:ye,nx,1);
%Defination of fluidic properties
rhoL=1.0;   %Density of fluid
mu=0.01;   %Dynamic viscosity of flluid
nu=mu/rhoL; %Kinematic viscosity of fluid
%Initialization of variables
u=zeros(ny+2,nx+2);  %Face centered velocity to the left side
v=zeros(ny+2,nx+2);  %Face centered velocity to the bottom side
p=zeros(ny+2,nx+2);  %Pressure cenetered towards cells
div=zeros(ny+2,nx+2);%Divergence extent cenetered towards cells
u_old = zeros(ny+2,nx+2);
v_old = zeros(ny+2,nx+2);
rho = zeros(ny+2,nx+2);
rho_f = zeros(ny_f+4,nx_f+4);
U = zeros(ny,nx);
V = zeros(ny,nx);
velMag = zeros(ny,nx);
vorticity = zeros(ny+2,nx+2);
Vorticity = zeros(ny,nx);
initialVelocity= 0.2; %Specifcation of initial velocity
dt=min(0.25*dx*dx/nu,4.0*nu/initialVelocity/initialVelocity); %Specification of Time
    Step
twfin=100*dt; %Termination criterion for simulation
%Initialization of immersed/rigid body
points=50;
radius=0.25;
Re=rhoL*initialVelocity*2*radius/mu; %Reynolds Number
xCent = xe/2.0;
yCent = ye/2.0;
rhoPsi = 1.0;
FSI_x = zeros(ny+2,nx+2);
FSI_y = zeros(ny+2,nx+2);
[X,X_t,X_c] = initIBM(points,radius,xCent,yCent,dx,dy); % Initialize my immersed
    boundary
points = length(X);
%Implementation of real grid boundary conditions
ut=0.0; %Velocity at top wall
ub=0.0; %Velocity at bottom wall
vl=0.0; %Velocity at left wall
vr=0.0; %Velocty ar right wall
ul = initialVelocity;
ur = 0.0;
vt = 0.0;
vb = 0.0;
%Defination of gravity conditions (Neglecting gravitational effects in analysis)
gx = 0.0;
gy = 0.0;
%% Main loop
t = 0; % Start time for simulation
endTime = 10;
pldt = 0.05;
plt = 0.0;
fileNumber = 0;
while t < endTime
    [u,v,u_old,v_old] = boundaryConditions(u,v,u_old,v_old,ut,ub,ul,ur,vt,vb,vl,vr,nx,
    ny); % Set boundary conditions
    dt = stabilityCondition(u,v,nu,nx,ny,dx); % CFL condition
```

```
    [u,v,FSI_x,FSI_y,f_x,f_y] = ibmClassic(X_c,X,X_t,u,v,nx,ny,dx,dy);
    [u,v] = intermediateVelocity(u,v,u_old,v_old,FSI_x,FSI_y,mu,nx,ny,dx,dy,dt);
    [p,a_p] = pressureSolve(u,v,rhoL,nx,ny,dx,dy,dt); % Pressure iterative solver
    [u,v,u_old,v_old] = accel(p,u,v,rhoL,nx,ny,dx,dy,dt); % Advance time step
    X = updateFiberModel(X,u,v,dx,dy,dt); % Advance fiber model
    % Post
    if plt > pldt
        figure(1)
         plot(X(:,1),X(:,2),'ro')
         xlim([0 xe])
         ylim([0 ye])
         xlabel("x")
         ylabel("y")
         set(gcf,'Position',[200 200 1000,450])
        exportgraphics(gca,'comboExample.gif','Append',true)
        for i=2:nx+1
            for j=2:ny+1
                vorticity(j,i) = sqrt(((v(j,i) - v(j,i-1))/dx -(u(j,i) - u(j-1,i))/dy)
^2);
            end
        end
        for i=1:nx
            for j=1:ny
                Vorticity(j,i) = 0.5*(vorticity(j+1,i+1) + vorticity(j+1,i+2));
            end
        end
        figure(2)
        contourf(x,y,Vorticity,'LevelList',0:.2:3)
        hold on
        colorbar
        set(gcf,'Position',[800 400 1000,380])
        xlabel("x")
        ylabel("y")
        clim([0 3])
        exportgraphics(gca,'Re_10_vortexField.gif','Append',true)
        fileNumber = fileNumber + 1;
        plt = 0;
    end
    dragForce = 0.0;
    liftForce = 0.0;
    ds = sqrt((X(2,1) - X(1,1))^2 +(X(2,2) - X(1,2))^2);
    for k=1:points
        dragForce = dragForce + -f_x(k)*ds;
        liftForce = liftForce + f_y(k)*ds;
    end
     figure(3)
     plot(t,dragForce,'ro')
     hold on
     xlim([0 5])
     ylim([0 0.25])
     xlabel("Time(s)")
     ylabel('Drag force')
     set(gcf,'Position',[900 500 1000,380])
    exportgraphics(gca,'test1.gif','Append',true)
    figure(4)
     plot(t,liftForce,'bo')
     hold on
     xlim([0 5])
     ylim([-1.5e-3 1.5e-3])
    xlabel("Time(s)")
     ylabel('Lift force')
     set(gcf,'Position',[0 400 1000,380])
    exportgraphics(gca,'test2.gif','Append',true)
    t = t + dt;
    plt = plt + dt;
end
% Post processing
dragForce = 0.0;
ds = sqrt((X(2,1) - X(1,1))^2 +(X(2,2) - X(1,2))^2);
for k=1:points
    dragForce = dragForce + f_x(k)*ds;
```

```
end
for i=1:nx
    for j=1:ny
        U(j,i) = 0.5 *(u(j+1,i+1) + u(j+1,i+2));
        V(j,i) = 0.5 *(v(j+1,i+1) + v(j+2,i+1));
        velMag(j,i) = sqrt(U(j,i)^2 + V(j,i)^2);
    end
end
%Evaluation of extent of divergence and evaluating the maximum divergence
div = calcDiv(div,u,v,nx,ny,dx,dy);
[m,ii,jj] = maxDiv(div,nx,ny);
% Functions
function [dt] = stabilityCondition(u,v,nu,nx,ny,dx)
    maxVelMag = 0.0;
    for i=1:nx+2
        for j=1:ny+2
            velMag = sqrt(u(j,i)^2 + v(j,i)^2);
            if velMag > maxVelMag
                maxVelMag = velMag;
            end
        end
    end
    dt1 = min(0.25*dx*dx/nu,4.0*nu/maxVelMag/maxVelMag);
    dt2 = 0.3*dx/maxVelMag; % CFL stability condition
    dt = min(dt1,dt2);
end
function [u,v,u_old,v_old] = boundaryConditions(~,~,u_old,v_old,ut,ub,ul,ur,vt,vb,vl,
    vr,nx,ny)
    u_old(:,2) = ul; % Left wall
    u_old(:,nx+2) = u_old(:,nx+1); % Right wall
    u_old(ny+2,:) = 2*ut + u_old(ny+1,:); % Top wall
    u_old(1,:) = 2*ub + u_old(2,:); % Bottom wall
    v_old(ny+2,:) = vt; % Top wall
    v_old(2,:) = vb; % Bottom wall
    v_old(:,1) = 2.0*vl - v_old(:,2); % Left wall
    v_old(:,nx+2) = 2.0*vr - v_old(:,nx+1); % Right wall;
    u = u_old;
    v = v_old;
end
function [X,X_t,X_c] = initIBM(points,radius,xCent,yCent,dx,dy)
    X = zeros(points,2); % x,y position of set of Lagangrian points
    X_t = zeros(points,2);
    X_c = zeros(points,points); % specified connectivity between springs
    dTheta = 2*pi/points;
    theta = 0.0;
    for j=1:points
        theta = theta + dTheta;
        x = radius*sin(theta);
        y = radius*cos(theta);
        X(j,1) = xCent + x;
        X(j,2) = yCent + y;
    end
    X_t = X;
    % Defination of connectivity matrix
    e = ones(points,1);
    X_c = spdiags([e 0*e e],-1:1,points,points);
    % If end points are connectead
    X_c(1,end) = 1;
    X_c(end,1) = 1;
end
function [f_x,f_y] = targetPoint(X,X_t,f_x,f_y)
    points = length(X(:,1));
    k_t = 100;
    for i=1:points
        f_x(i) = -k_t*(X(i,1) - X_t(i,1));
        f_y(i) = -k_t*(X(i,2) - X_t(i,2));
    end
end
function [f_x,f_y] = springsHookean(X_c,X,f_x,f_y)
    points = length(X(:,1));
    k_s = 20; % spring stiffness(N/m)
```

```
    r_l = 0.1; % spring resting length(m)
    for j=1:points
        for i=1:points
            % Connection at point
            if X_c(j,i) == 1
                x_r = X(i,1);
                x_l = X(j,1);
                y_r = X(i,2);
                y_l = X(j,2);
                tempF_x = k_s*(sqrt((x_r - x_l)^2 +(y_r - y_l)^2) - r_l)*(x_r - x_l);
                tempF_y = k_s*(sqrt((x_r - x_l)^2 +(y_r - y_l)^2) - r_l)*(y_r - y_l);

                f_x(j) = f_x(j) + tempF_x;
                f_y(j) = f_y(j) + tempF_y;
            end
        end
    end
end
function [f_x,f_y] = springsTorsional(X,f_x,f_y)
    points = length(X(:,1));
    k_b = 20; % bending stiffness(N/m)
    theta = 20.0*pi/180; % desired angle
    % the master is i+1
    for i=2:points-1
        X_l = X(i-1,1);
        X_m = X(i,1);
        X_r = X(i+1,1);
        Y_l = X(i-1,2);
        Y_m = X(i,2);
        Y_r = X(i+1,2);
        d_lm = sqrt((X_l - X_m)^2 +(Y_l - Y_m)^2);
        d_rm = sqrt((X_r - X_m)^2 +(Y_r - Y_m)^2);
        C = d_lm*d_rm*sin(theta);
        f_x(i) = k_b*((X_r - X_m)*(Y_m - Y_l) -(Y_r - Y_m)*(X_m - X_l) - C)*((Y_m -
    Y_l) +(Y_r - Y_m));
        f_y(i) = k_b*((X_r - X_m)*(Y_m - Y_l) -(Y_r - Y_m)*(X_m - X_l) - C)*(-(X_r -
    X_m) -(X_m - X_l));
    end
end
function [u,v,FSI_x,FSI_y,f_x,f_y] = ibmClassic(X_c,X,X_t,u,v,nx,ny,dx,dy)
    points = length(X(:,1));
    ds = dx;
    E_x = zeros(points,1);
    E_y = zeros(points,1);
    f_x = zeros(points,1);
    f_y = zeros(points,1);
    FSI_x = zeros(ny+2,nx+2);
    FSI_y = zeros(ny+2,nx+2);
    [f_x,f_y] = targetPoint(X,X_t,f_x,f_y);
    % [f_x,f_y] = springsHookean(X_c,X,f_x,f_y);
    % [f_x,f_y] = springsTorsional(X,f_x,f_y);
    for i=2:nx+1
        for j=2:ny+1
            for k=1:points
                tx_X = dx*(i-2);
                ty_X = dy*(j-2) + 0.5*dy;

                tx_Y = dx*(i-2) + 0.5*dx;
                ty_Y = dy*(j-2);

                r_x_X = abs(X(k,1) - tx_X) / dx;
                r_y_X = abs(X(k,2) - ty_X) / dy;

                r_x_Y = abs(X(k,1) - tx_Y) / dx;
                r_y_Y = abs(X(k,2) - ty_Y) / dy;

                if r_x_X <= 2
                    phi_x = delta1(r_x_X);
                else
                    phi_x = 0.0;
                end
```

```
                if r_y_X <= 2
                    phi_y = delta1(r_y_X);
                else
                    phi_y = 0.0;
                end

                delta = phi_x*phi_y/dx/dy;
                FSI_x(j,i) = FSI_x(j,i) + f_x(k)*delta*ds;

                if r_x_Y <= 2
                    phi_x = delta1(r_x_Y);
                else
                    phi_x = 0.0;
                end

                if r_y_Y <= 2
                    phi_y = delta1(r_y_Y);
                else
                    phi_y = 0.0;
                end

                delta = phi_x*phi_y/dx/dy;
                FSI_y(j,i) = FSI_y(j,i) + f_y(k)*delta*ds;
            end
        end
    end
end
function [d] = delta1(a)
    d = 0.25*(1 + cos(pi*a/2));
end
function [X] = updateFiberModel(X,u,v,dx,dy,dt)
    points = length(X(:,1));
    for k=1:points
       Interoilation of velocity for fibres
        x_left = round(X(k,1)/dx - 2,'TieBreaker','minusinf');
        x_right = round(X(k,1)/dx + 2,'TieBreaker','plusinf');
        y_bot = round(X(k,2)/dy - 2,'TieBreaker','minusinf');
        y_top = round(X(k,2)/dy + 2,'TieBreaker','plusinf');
        u_fiber = 0.0;
        v_fiber = 0.0;
        for i=x_left:x_right
            for j=y_bot:y_top

                tx_X = dx*(i-2);
                ty_X = dy*(j-2) + 0.5*dy;

                tx_Y = dx*(i-2) + 0.5*dx;
                ty_Y = dy*(j-2);

                r_x_X = abs(X(k,1) - tx_X) / dx;
                r_y_X = abs(X(k,2) - ty_X) / dy;

                r_x_Y = abs(X(k,1) - tx_Y) / dx;
                r_y_Y = abs(X(k,2) - ty_Y) / dy;

                if r_x_X <= 2
                    phi_x = 0.25*(1 + cos(pi*r_x_X/2));
                else
                    phi_x = 0.0;
                end

                if r_y_X <= 2
                    phi_y = 0.25*(1 + cos(pi*r_y_X/2));
                else
                    phi_y = 0.0;
                end

                delta = phi_x*phi_y/dx/dy;
                u_fiber = u_fiber + u(j,i)*delta*dx*dy;
```

```
                if r_x_Y <= 2
                    phi_x = 0.25*(1 + cos(pi*r_x_Y/2));
                else
                    phi_x = 0.0;
                end

                if r_y_Y <= 2
                    phi_y = 0.25*(1 + cos(pi*r_y_Y/2));
                else
                    phi_y = 0.0;
                end

                delta = phi_x*phi_y/dx/dy;
                v_fiber = v_fiber + v(j,i)*delta*dx*dy;

            end
        end
        X(k,1) = X(k,1) + u_fiber*dt;
        X(k,2) = X(k,2) + v_fiber*dt;
    end
end
function [u,v] = intermediateVelocity(u,v,u_old,v_old,FSI_x,FSI_y,mu,nx,ny,dx,dy,dt)
    for i=3:nx+1
        for j=2:ny+1
            % Interpolating velocities
            u_e = 0.5*(u_old(j,i) + u_old(j,i+1));
            u_w = 0.5*(u_old(j,i-1) + u_old(j,i));
            u_n = 0.5*(u_old(j,i) + u_old(j+1,i));
            u_s = 0.5*(u_old(j,i) + u_old(j-1,i));

            v_n = 0.5*(v_old(j+1,i-1) + v_old(j+1,i));
            v_s = 0.5*(v_old(j,i-1) + v_old(j,i));

            % Solving div(rho*u*u) and div(tau)
            convection = -(u_e*u_e - u_w*u_w)/dx -(v_n*u_n - v_s*u_s)/dy;
            diffusion = mu*(u_old(j,i-1) - 2.0*u_old(j,i) + u_old(j,i+1))/dx/dx + mu*(
    u_old(j+1,i) - 2.0*u_old(j,i) + u_old(j-1,i))/dy/dy;

            % Calculate intermediate u velocity
            u(j,i) = u_old(j,i) + dt*(diffusion + convection + FSI_x(j,i));
        end
    end
    for i=2:nx+1
        for j=3:ny+1
            % Interpolating velocities
            v_e = 0.5*(v_old(j,i) + v_old(j,i+1));
            v_w = 0.5*(v_old(j,i) + v_old(j,i-1));
            v_n = 0.5*(v_old(j,i) + v_old(j+1,i));
            v_s = 0.5*(v_old(j,i) + v_old(j-1,i));

            u_e = 0.5*(u_old(j-1,i+1) + u_old(j,i+1));
            u_w = 0.5*(u_old(j-1,i) + u_old(j,i));

            % Solving div(rho*u*u) and div(tau)
            convection = -(v_e*u_e - v_w*u_w)/dx -(v_n*v_n - v_s*v_s)/dy;
            diffusion = mu*(v_old(j,i-1) - 2*v_old(j,i) + v_old(j,i+1))/dx/dx + mu*(
    v_old(j+1,i) - 2*v_old(j,i) + v_old(j-1,i))/dy/dy;

            % Calculate intermediate v velocity
            v(j,i) = v_old(j,i) + dt*(diffusion + convection + FSI_y(j,i));
        end
    end
end
function [p,a_p] = pressureSolve(u,v,rho,nx,ny,dx,dy,dt)
    [p,a_p] = pressureOptimized2(u,v,rho,nx,ny,dx,dy,dt);
end
function [p,a_p] = pressureOptimized2(u,v,rho,nx,ny,dx,dy,dt)
    p = zeros(ny+2,nx+2);
    for i=2:nx+1
        for j=2:ny+1
            rhs(j,i) =((u(j,i+1) - u(j,i))/dx +(v(j+1,i) - v(j,i))/dy)/dt;
```

```
        end
    end
    e = ones(nx,1);
    Ax = spdiags([e 0*e e],-1:1,ny,ny)/dx/dx;
    Ay = spdiags([e 0*e e],-1:1,nx,nx)/dy/dy;
    Ix=speye(ny,ny);
    Iy=speye(nx,nx);
    Iy3 = speye(nx,nx);
    Iy3(1:nx-1,1:nx-1) = 0;
    a1 = kron(Iy,Ax); % This is correct
    a2 = kron(Ay,Ix); % this is correct
    A3 = a1 + a2;
    A4 = A3;
    middleCoeff = diag(sum(A4));
    A5 = A4 - middleCoeff;
    % Now add in boundary conditiosn
    Ax2 = spdiags([0*e -e 0*e ],-1:1,ny,ny)/dx/dx;
    Ax2(1,1) = -e(1)/dx/dx;
    Ax2(ny,ny) = -e(ny)/dx/dx;
    Iy2 = speye(nx,nx);
    Iy2(2:nx,2:nx) = 0;
    Ax22 = spdiags([0*e -e 0*e ],-1:1,ny,ny)/dx/dx;
    Ax22(ny,ny) = -e(ny)/dx/dx;

    a3 = kron(Iy2,Ax2)/rho;
    a4 = kron(Iy3,Ax22)/rho;
    A6 = A5;
    A6 = A5 + a3 + a4;
    b = zeros(ny,nx);
    for i=2:nx+1
        for j=2:ny+1
            b(j-1,i-1) = rhs(j,i);
        end
    end
    b2 = reshape(b,[ny*nx,1]);

    x2 = A6 \ b2;
    x = reshape(x2,[ny,nx]);

    for i=2:nx+1
        for j=2:ny+1
            p(j,i) = x(j-1,i-1);
        end
    end
    a_p = 1;
end
function [u,v,u_old,v_old] = accel(p,u,v,rho,nx,ny,dx,dy,dt)
    for i=3:nx+1
        for j=2:ny+1
            % Update u velocity
            u(j,i) = rho*u(j,i) - dt*(p(j,i) - p(j,i-1))/dx;

            u(j,i) = u(j,i) / rho;
        end
    end

    for i=2:nx+1
        for j=3:ny+1
            % Update v velocity
            v(j,i) = rho*v(j,i) - dt*(p(j,i) - p(j-1,i))/dy;

            v(j,i) = v(j,i) / rho;
        end
    end
    u_old = u;
    v_old = v;
end
%% Utility functions

function div = calcDiv(div,u,v,nx,ny,dx,dy)
    for i=2:nx-1
```

```
        for j=2:ny-1
            div(j,i) =(u(j,i+1) - u(j,i))/dx +(v(j+1,i) - v(j,i))/dy;
        end
    end
end

function [max,ii,jj] = maxDiv(div,nx,ny)
    max = 0.0;
    for i=2:nx-1
        for j=2:ny-1
            if div(j,i) > max
                max = div(j,i);
                ii = i;
                jj = j;
            end
        end
    end
end
```

Listing 1: MATLAB code for analysis of Fluid-Surface Interaction of fluidic vorticea at Re

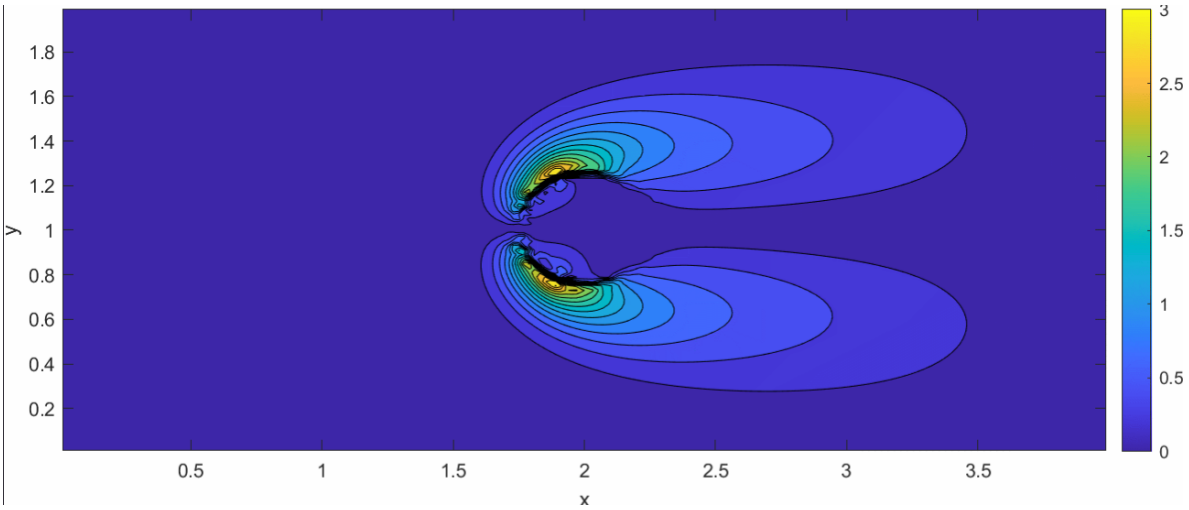
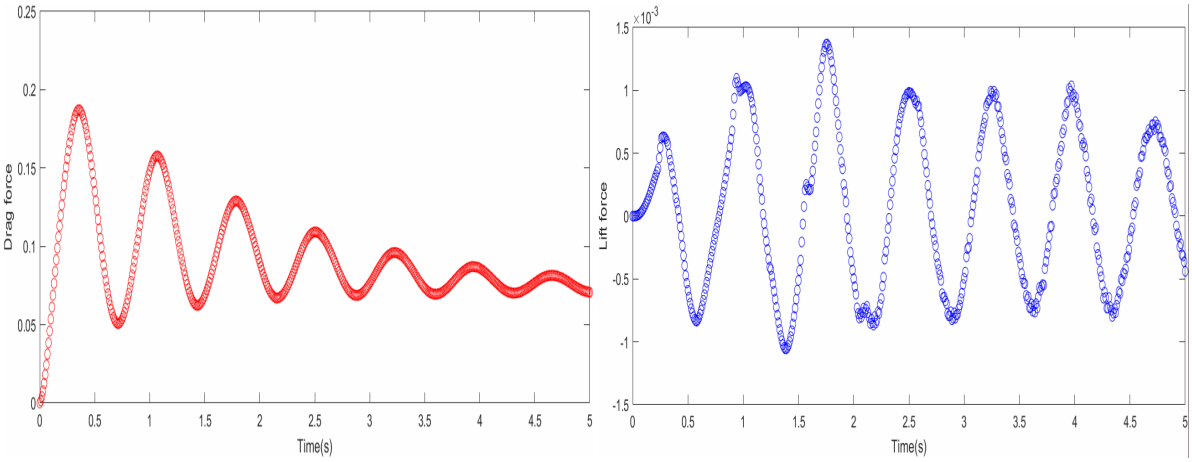


Figure 1: Fluid Surface Interaction Contour at Re=10



(a) Drag Force Distribution about Contour

(b) Lift Force Distribution about Contour

Figure 2: Curves for Drag and Lift Force within the Contour

12

[2]This code is excerpts of Code with necessary amendments made as per requirements.

Conclusion

The present study offers a comprehensive investigation into the fluid–structure interaction (FSI) phenomena associated with vortex formation and evolution around a flexible structure immersed in a laminar crossflow.One important paarmeter of analysis was the variation in lift and drag coefficients whihc were extensively deliberated upon to visualize the vortex shedding mechanisms and the effect of structural flexibility. From the Figure 1, a well-developed von Kármán vortex street was observed, yet its amplitude modulation suggests a two-way coupling between the fluid forces and the structure's oscillatory motion.The drag coefficient plots highlight predominantly unidirectional trend with superimposed oscillatory components. These oscillations arise from the intermittent enlargement and downstream advection of wake vortices, which induce pressure differentials and transient shear layers along the structure's surface. The drag amplitude's sensitivity to both flow velocity and structural stiffness underscores the role of wake-induced resonance phenomena, a key contributor to unsteady force magnitudes in real-world applications such as aeroelastic flutter and vortex-induced vibration (VIV). Notably, the drag curve demonstrates greater irregularity in its frequency spectrum compared to lift, suggesting that longitudinal structural displacements also participate in the feedback mechanism, albeit less dominantly. The formation and detachment of coherent vortex structures were visualized in sequential time frames, and their propagation trajectories exhibited clear signs of interaction with the deformed surface geometry. These surface modulations not only influenced the shedding frequency but also altered the near-wake topology, particularly in the generation of secondary vortices and the widening of the recirculation zone. As a result, the dynamic pressure field adapts continuously to the instantaneous shape of the structure, introducing nonlinearity into the aerodynamic loading and fostering complex modal interactions. This is a hallmark of strong FSI behavior where neither the fluid nor the structure can be considered passive. Moreover, the study confirms that the lift-to-drag ratio experiences significant variations due to structural compliance. While a rigid body would yield relatively steady aerodynamic performance, the flexible structure studied here benefits from transient energy harvesting opportunities at specific oscillation phases, as evidenced by peaks in the instantaneous lift-to-drag ratio. Such observations are especially relevant for energy-harvesting devices and bio-inspired propulsion systems, where synchronization between structural motion and vortex timing is critical to efficiency.

References

[1] Bathe, K.-J., and Zhang, H. Finite element developments for general fluid flows with structural interactions. *International Journal for Numerical Methods in Engineering 60*, 1 (2004), 213–232.

[2] Bungartz, H.-J., and Schäfer, M., Eds. *Fluid-Structure Interaction: Modelling, Simulation, Optimization*. Springer-Verlag, 2006.

[3] Heil, M. An efficient solver for the fully coupled solution of large-displacement fluid-structure interaction problems. *Computer Methods in Applied Mechanics and Engineering 193*, 1–2 (2004), 1–23.

[4] Hron, J., and Turek, S. A monolithic fem/multigrid solver for ale formulation of fluid-structure interaction with application in biomechanics. In *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, H.-J. Bungartz and M. Schäfer, Eds., Lecture Notes in Computational Science and Engineering. Springer, 2006, pp. 146–170.

[5] Matthies, H., Niekamp, R., and Steindorf, J. Algorithms for strong coupling procedures. *Computer Methods in Applied Mechanics and Engineering 195*, 17–18 (2006), 2028–2049.

[6] Matthies, H., and Steindorf, J. Partitioned strong coupling algorithms for fluid-structure interaction. *Computers and Structures 81*, 8–11 (2003), 805–812.

[7] Michler, C., van Brummelen, E., and de Borst, R. Error-amplification analysis of subiteration-preconditioned gmres for fluid-structure interaction. *Computer Methods in Applied Mechanics and Engineering 195*, 17–18 (2006), 2124–2148.

[8] Sigrist, J. *Fluid-Structure Interaction: An Introduction to Finite Element Coupling*. Wiley, 2015.