Enclosure A

User Defined Functions for ANSYS Fluent

DEFINE_SDOF_PROPERTIES

```
#include "udf.h"

DEFINE_SDOF_PROPERTIES(dof_udf, prop, dt, time, dtime)
{
        prop[SDOF_MASS] = 0.2160;       // mass of the plate
        prop[SDOF_IZZ] = 2.8872e-05;    // moment of inertia around z-axis
        prop[SDOF_LOAD_F_Y] = 0.7834;   // buoyancy force
}
```

DEFINE_CG_MOTION

Rotational Motion

```
#include "udf.h"

DEFINE_CG_MOTION(rotation_udf_45, dt, vel, omega, time, dtime)
{
        real pi = 3.141592;
        real omega_constant = 45*pi/180;
        omega[2] = omega_constant;
}
```

Translational Motion

```
#include "udf.h"

DEFINE_CG_MOTION(translation_udf, dt, vel, omega, time, dtime)
{
        real u = 0.5;
        real a = 5.0;
        real t1 = u/a;
        real t2 = t1+0.4;
        real t3 = t2+t1;


        if (time < t1) {
                vel[0] = a*time;
                }
        else if (time > t1 && time <= t2) {
```

```
                vel[0] = u;
                }
        else if (time > t2 && time <= t3) {
                vel[0] = u-a*(time-t2);
                }
        else {
                vel[0]=0;
                }
}
```

DEFINE_EXECUTE_AT_END

```
//UDF extracting the viscous and pressure forces in the x- and y-direction
    as well as the location of the centre of pressure

#include "udf.h"
#include "stdio.h"
#include "math.h"

DEFINE_EXECUTE_AT_END(save_results_udf)
{
        FILE *fp = NULL;
        FILE *fp2 = NULL;
        char filename[]="force_history.txt";
        real flow_time = RP_Get_Real("flow-time");
        double NV_VEC(viscous);
        double NV_VEC(pressure);
        double NV_VEC(force);
        double NV_VEC(total_force);
        double area = 0.;
        double NV_VEC(A);
        double NV_VEC(Wall_Shear_Force);
        double num_y = 0.;
        double num_x = 0.;
        double den_x = 0.;
        double den_y = 0.;
        double NV_VEC(cp);
        real x[ND_ND];
        int IDplate = 18; //Plate ID

        face_t f;
        Thread *f_thread;
        Domain *d;

        d = Get_Domain(1); //Fluid ID
        f_thread = Lookup_Thread(d, IDplate);
        fp = fopen(filename,"a");

        begin_f_loop(f,f_thread)
        // loop through the faces on the plate
        {
                F_AREA(A,f,f_thread);
                // calculate the area of the current face
                F_CENTROID(x,f,f_thread);
                // calculate the centre point of the current face
                Wall_Shear_Force[0] = -F_STORAGE_R_N3V(f,f_thread,
                    SV_WALL_SHEAR)[0];
```

```
            // wall shear stress in the x-direction for current face
            Wall_Shear_Force[1] = -F_STORAGE_R_N3V(f,f_thread,
                SV_WALL_SHEAR)[1];
            // wall shear stress in the x-direction for current face

            force[0] = F_P(f,f_thread) * A[0] + Wall_Shear_Force[0];
            // total force in the x-direction for current face
            force[1] = F_P(f,f_thread) * A[1] + Wall_Shear_Force[1];
            // total force in the y-direction for current face

            num_x += force[1] * x[0];
            // numerator for calculation of the centre of pressure x-
                direction
            den_x += force[1];
            // denominator for calculation of the centre of pressure x-
                direction

            num_y += force[0] * x[1];
            // numerator for calculation of the centre of pressure y-
                direction
            den_y += force[0];
            // denominator for calculation of the centre of pressure y-
                direction

            viscous[0] += Wall_Shear_Force[0];
            // summation of the viscous force in x-direction
            viscous[1] += Wall_Shear_Force[1];
            // summation of the viscous force in y-direction

            pressure[0] += F_P(f,f_thread) * A[0];
            // summation of the pressure force in the x-direction
            pressure[1] += F_P(f,f_thread) * A[1];
            // summation of the pressure force in the y-direction

            total_force[0] += force[0];
            // summation of the total force in the x-direction
            total_force[1] += force[1];
            // summation of the total force in the y-direction
        }
        end_f_loop(f,f_thread)
        cp[0] = num_x / den_x; // x-coordinate of the centre of pressure
        cp[1] = num_y / den_y; // y-coordinate of the centre of pressure
        fprintf(fp,"%0.4e\t%0.4e\t%0.4e\t%0.4e\t%0.4e\t%0.4e\t%0.4e\
            t%0.4e\n",flow_time,viscous[0],pressure[0],total_force[0],
            viscous[1],pressure[1],total_force[1],cp[0],cp[1]);
        // write to file
        fclose(fp);
}
```

Enclosure B

Revised Model

```
%%%%%%%%   Revised 2D model of a flat plate in free fall.      %%%%%%%%

% PROJECT TITLE:     Extending the Existing Modelling Framework for
%                    Non-Spherical Particles to Include Flat Plates in Free
%                    Fall - An Experimental and Numerical Investigation of
%                    the Unsteady Aerodynamics of Flat Plates
% AUTHOTS:           Anna Lyhne Jensen and Jakob Hærvig
% DETAILS:           Master Thesis project 1st February to 3rd June 2014
% CONTACT:           Email: jakob.haervig(at)gmail.com

clc; clear all; close all    %clear command windows, workspace and figures

% LOAD FILES FOR UNSTEADY LIFT AND DRAG REGRESSIONS
coefficient_matrix = importdata('coefficient_matrix.mat');
alpha_start_matrix = importdata('alpha_start_matrix.mat');
alpha_end_matrix = importdata('alpha_end_matrix.mat');

% OPERATORS USED TO TURN VECTOR 90 DEGRESS
T90cw = [cosd(90) sind(90);-sind(90) cosd(90)]; %turns vector 90 deg cw
T90ccw = [cosd(90) -sind(90);sind(90) cosd(90)];%turns vector 90 deg ccw

% PARAMETERS DEFINED BY USER
rho_f = 998.2;               %density of fluid [kg/m^3]
mu = 0.001003;               %dynamic viscosity of fluid [Pa*s]
rho_p = 2700;                %density of plate [kg/m^3]
g = 9.81;                    %gravitational acceleration [m/s^2]
beta = 1/20;                 %aspect ratio of plate [-]
L = 0.04;                    %length of plate [m]
h = L*beta;                  %height of plate [m]
V = L*h;                     %volume of plate[m^2]
m = V*rho_p;                 %mass of plate [kg/m]
m_f = V*rho_f;               %mass of fluid occupied by plate [kg/m]
Iz = 1/12*m*(h^2+L^2);       %moment of inertia for plate [kg*m]
Iz_f = 1/12*m_f*(h^2+L^2);   %moment of inertia for fluid [kg*m]
dt = 0.00025;                %time step size [s]
t_start = 0;                 %start time [s]
t_end = 3;                   %end time [s]

c1 = pi/2;                   %constant to calculate angle of attack [rad]
c2 = pi;                     %constant to calculate angle of attack [rad]
dalpha_dt_matrix = [0:5:90];%vector giving dalpha/dt simulations

pos = [0 0];                 %initial position of plate [m]
u = [0 -0.0002];             %initial velocity of plate [m/s]
```

```
w = [0 0];                  %velocity of fluid surround the plate [m/s]
v = w-u;                    %initial relative fluid velocity [m/s]
theta = -30*pi/180;         %initial orientational angle [deg]
omega = 0;                  %initial angular velocity of plate [rad/s]
domega = 0;                 %initial angular acceleration of plate [rad/s]
dalpha_dt = 0;              %intial change in angle of attack [rad/s]

% INITIAL CALCULATIONS
phi = atan(v(2)/v(1));      %relative fluid velocity angle [m/s]
alpha = 2*c1/pi*atan(cot(pi/c2*theta-pi/2-phi));
                            %angle of attack [rad]

t = t_start;                %initialise time as start time [s]
i = 1;                      %set loop counter
tic
while t<t_end               %start main loop

    T_to_local = [cos(theta) sin(theta); -sin(theta) cos(theta)];
                                        %trans. to local coordinates
    T_to_global = [cos(theta) -sin(theta); sin(theta) cos(theta)];
                                        %trans. to global coordinates
    v = w-u;                            %relative fluid velocity

    % CALCULATE DRAG AND LIFT COEFFICIENTS

    Re = norm(v)*L*rho_f/mu;            %update reynolds number
    if Re <= 5
        CD90 = 1.75+5.0/(0.20*Re^1.18);     %CD(90) for Re<=5 (extrapol.)
    elseif Re > 5 && Re <= 75
        CD90 = 1.75+5.0/(0.20*Re^1.18);     %CD(90) for 5<Re<=75
    elseif Re > 75 && Re <= 1280
        CD90 = 1.63*10^-9*Re^3-5.17*10^-6*Re^2+...
            5.41*10^-3*Re+1.54;             %CD(90) for 75<Re<=1280
    elseif Re > 1280 && Re <= 20000
        CD90 = (3.05+5.0/(0.045*Re^0.8));   %CD(90) for 1280<Re<=20000
    else
        CD90 = (3.05+5.0/(0.045*Re^0.8));   %CD(90) for Re>20000 (extrapol.)
        end

    CD0 = (0.023+5.45/(-0.8+Re^0.58));      %CD(0) for 5<Re20000
    CD = CD0+abs((CD90-CD0)*sin(alpha).^3);%blending function by Rosendahl

    if abs(dalpha_dt) > pi/2
        dalpha_dt = pi/2;                   %limit dalpha/dt to 90 deg/s
    end

    a1 = ((pi/2)/abs(dalpha_dt))^0.5;       %freq. paramter for unsteady CD
    a2 = (pi/2)/abs(dalpha_dt);             %amp. parameter for unsteady CD

    CD_unsteady = (pi/a2+CD0)-CD0/a2*cos(a1*abs(alpha)+pi/2).^2-CD90/...
        a2*sin(a1*abs(alpha)+pi/2).^6;      %approx of unsteady drag
    CD_steady = CD0+abs((CD90-CD0)*sin(alpha).^3);
                                            %steady drag
    if CD_steady<CD_unsteady
        CD = CD_unsteady;                   %use unsteady drag coefficient
    else
        CD = CD_steady;                     %use steady drag coefficient
    end
```

```
dalpha_dt_rounded = round(abs(dalpha_dt)*180/pi/5)*5;
                                        %round dalpha/dt
dalpha_dt_vec_index = find(dalpha_dt_matrix == dalpha_dt_rounded);
                                        %find index of nearest ang. vel.

if isempty(dalpha_dt_vec_index) == 1
    dalpha_dt_vec_index = 11;           %limit dalpha/dt to 90 deg/s
end

if abs(alpha) < alpha_start_matrix(dalpha_dt_vec_index)*pi/180
    CL = abs(alpha)*180/pi*0.1263;      %linear regression
elseif  (abs(alpha) > alpha_start_matrix(dalpha_dt_vec_index)*pi/180
    &&...
        abs(alpha) < alpha_end_matrix(dalpha_dt_vec_index)*pi/180)
    CL = coefficient_matrix(1,dalpha_dt_vec_index)*abs(alpha)^3+...
        coefficient_matrix(2,dalpha_dt_vec_index)*abs(alpha)^2+...
        coefficient_matrix(3,dalpha_dt_vec_index)*abs(alpha)+...
        coefficient_matrix(4,dalpha_dt_vec_index);
                                        %dalpha/dt dependent CL
else
    CL = 7.0763*abs(alpha)^5-28.9422*abs(alpha)^4+...
        41.3903*abs(alpha)^3-25.7446*abs(alpha)^2+...
        7.3899*abs(alpha)+0.0375;       %dalpha/dt for steady CL
end

% CALCULATE FORCES, ACCELERATION, AND UPDATE POSITION

Fb = [0 V*rho_f*g];                     %buoyancy force vector
Fg = [0 -g*m];                          %gravity force vector

Fd_abs = CD*1/2*rho_f*L*norm(v)^2;      %absolute drag force
Fl_abs = CL*1/2*rho_f*L*norm(v)^2;      %absolute lift force
Fd = v/norm(v)*Fd_abs;                  %drag force vector
    if alpha<0
    Fl = (T90cw*(v/norm(v)*Fl_abs)')';
                                        %turn lift clock-wise
else
    Fl = (T90ccw*(v/norm(v)*Fl_abs)')';
                                        %turn lift counter-close-wise
end

C = [0 0; 0 0];                         %added mass coefficient matrix
Ma = C.*rho_f.*V;                       %added mass matrix
M = eye(2)*m;                           %total mass matrix
a_local = inv(M+Ma)*T_to_local*(Fd'+Fl'+Fg'+Fb');
                                        %accele. in local coordinates
a_global = T_to_global*a_local;         %accele. in global coordinates
a = a_global';                          %update acceleration

u = u+a*dt;                             %update translational velocity
v = w-u;                                %update relative fluid velocity
dpos = u*dt+1/2*a*dt^2;                 %find change in position
pos = pos+dpos;                         %update position

phi = atan(v(2)/v(1));                  %find rel. fluid velocity angle
alpha_old = alpha;                      %save old angle of attack
alpha = 2*c1/pi*atan(cot(pi/c2*theta-pi/2-phi));
```

```
                                        %update angle of attack
    dalpha_dt = (alpha-alpha_old)/dt;   %find change in angle of attack

    % CALCULATE CENTRE OF PRESSURE LOCATION

    cp_abs = L*0.015*(1-sin(abs(alpha))^3);
                                        %find centre of pressure dist.

    sign_cp = sign_cp_fcn(theta,v);     %call fun. to find sign of cp
        cp = [cp_abs*abs(cos(theta))*sign_cp(1)...
        cp_abs*abs(sin(theta))*sign_cp(2)];%find cp vector


        T_offset = det([cp; Fd])+det([cp; Fl]);
                                        %calc. torque aerodyn. forces
        T_resist = -rho_f*2*omega*abs(omega)*(1/4)*(L/2)^4;
                                        %calculate torque resistance
    M66 = 0*Iz_f;                       %added moment of inertia
    T_total = T_offset+T_resist;        %total torque on plate
    domega = T_total/(Iz+M66);          %find angular acceleration

    omega = omega+domega*dt;            %update angular velocity
    dtheta = omega*dt+1/2*domega*dt^2;  %find change in orient. angle
    theta = theta+dtheta;               %update orientation angle

    % SAVE RESULTS TO ARRAYS FOR LATER PLOTTING

    pos_vec(i,:) = pos;                 %position
    u_vec(i,:) = u;                     %trans. velocity
    u_mag_vec(i) = norm(u);             %trans. velocity magnitude
    a_vec(i,:) = a;                     %trans. acceleration
    a_vec_mag(i) = norm(a);             %trans. acceleration magnitude
    theta_vec(i) = theta;               %orientation angle
    alpha_vec(i) = alpha;               %angle of attack
    omega_vec(i) = omega;               %angular velocity
    domega_vec(i) = domega;             %angular acceleration
    Fd_vec(i,:) = Fd;                   %drag force
    Fl_vec(i,:) = Fl;                   %lift force
    cp_abs_vec(i) = cp_abs;             %centre of pressure magnitude
    t_vec(i) = t;                       % time

    % UPDATE LOOP COUNTER AND TIME

    i = i+1;                            %update loop counter
    t = t+dt;                           %update time
end

%%
toc

% PLOTTING FREE FALL TRAJECTORY WITH ANIMATION
figure(1)
plot(pos_vec(:,1)/L,pos_vec(:,2)/L,'k'); hold on
for i = 1:50:length(t_vec)
    xplate(1) = cos(theta_vec(i))*L/2;
    yplate(1) = sin(theta_vec(i))*L/2;
    xplate(2) = -xplate(1);
    yplate(2) = -yplate(1);
```

```
    plot((pos_vec(i,1)+xplate)/L,(pos_vec(i,2)+yplate)/L,'k','Linewidth',2)
    hold on
    xlabel('Normalised distance X/L [-]')
    ylabel('Normalised distance Y/L [-]')
    axis equal
    pause(0.01)
end
```

coefficient_matrix.mat

```
% coefficient matrix for unsteady lift approximation
coefficient_matrix = [
         0         0         0         0
 -357.1757   86.8499    0.7730    0.1468
 -339.6383  124.0477   -7.0129    0.5075
 -321.5198  151.9937  -15.5799    1.0821
 -305.1881  175.2423  -24.9560    1.8921
 -273.7018  183.2955  -32.3156    2.7558
 -231.0180  175.2214  -35.9001    3.4559
 -186.1378  156.3123  -35.7169    3.8684
 -144.3193  131.6641  -32.4764    3.9553
 -109.2020  106.3660  -27.4888    3.7768
  -80.9141   82.6345  -21.5865    3.3881
  -65.3751   70.8033  -19.5711    3.4626
  -50.9844   56.5638  -15.2833    3.1031
  -44.3648   51.5779  -14.8134    3.2965
  -33.3199   37.1812   -8.5775    2.4114
   -9.1105   -3.2495   13.9542   -1.7400
   -0.9539  -17.3068   22.3751   -3.4478
    7.8310  -33.9494   33.2012   -5.8047
   15.8961  -50.5052   44.8213   -8.5169]';
```

alpha_start_matrix.mat

```
alpha_start_matrix = [0 1.6 3.2 9.0 7.5 9.0 11.0 13.0 14.0 15.0 17.0 18.0
    18.0 20.0 20.0 21.0 24.5 25.6 25.0];
```

alpha_end_matrix.mat

```
alpha_end_matrix = [0 12.0 15.7 18.9 21.8 24.7 27.6 30.6 33.7 36.8 39.8
    42.7 45.3 47.6 49.7 52.7 55.0 57.8 61.0];
```

Enclosure C

Torque due to Buoyancy

```
clear all; close all; clc

L=0.04;                          %Length of plate [m]
n=40;                           %Number of discretisations on each side [-]
beta = 1/20;                    %Aspect ratio [-]
h=L*beta;                       %Height of plate [m]

k=1;
for theta = -pi/2 : pi/32 : pi/2
T = [cos(theta) sin(theta); -sin(theta) cos(theta)];
                                %Transformation matrix [-]
g = 9.81;                       %Gravitational acceleration [m/s^2]
rho_f = 998.2;                  %Density of fluid [kg/m^3]
pos = [0 -1];                   %Position of plate [m]
P0 = 100000;                    %Reference pressure [Pa]

% CALCULATE POSITION OF PLATE CORNERS, LENGTH OF SIDES, AND UNIT VECTORS
% ALONG THE SIDES

endplate_low = [1/2*L*cos(theta)+pos(1) 1/2*L*sin(theta)+pos(2)];
                        %Position of midpoint of small edge of plates
endplate_high = [-1/2*L*cos(theta)+pos(1) -1/2*L*sin(theta)+pos(2)];
                        %Position of midpoint of small edge of plate
vec = [-1/2*h*sin(theta) 1/2*h*cos(theta)];
                        %Vector from midpoint to corner
corner(1,:) = endplate_high+vec;      %Location of 1st corner
corner(2,:) = endplate_low+vec;       %Location of 2nd corner
corner(3,:) = endplate_low-vec;       %Location of 3rd corner
corner(4,:) = endplate_high-vec;      %Location of 4th corner

xy_length = [corner(2,1)-corner(1,1) corner(2,2)-corner(1,2)];
                        %Vector for length
xy_height = [corner(3,1)-corner(2,1) corner(3,2)-corner(2,2)];
                        %Vector for height

unit_vec(1,:) = xy_length/norm(xy_length);    %Unit vector length
unit_vec(2,:) = xy_height/norm(xy_height);    %Unit vector height
unit_vec(3,:) = -unit_vec(1,:);               %- Unit vector length
unit_vec(4,:) = -unit_vec(2,:);               %- Unit vector height


for i = [1 2 3 4]                             %Loop through four edges
        x = corner(i,1);                          %x-position of current
```

```
        corner
    y = corner(i,2);                        %y-position of current
        corner


    if i==1 || i==3
    dx = L/n;                               %Cell length
    X = L;                                  %Length of edge
else
    dx = h/n;                               %Cell length
    X = h;                                  %Length of edge
    end

num_x = 0;                                  %Reset numerator in cpx
den_x = 0;                                  %Reset denominator in cpx
num_y = 0;                                  %Reset numerator in cpy
den_y = 0;                                  %Reset denominator in cpy

    for q = 1/2*dx : dx : X-1/2*dx
    pos_point = [x+unit_vec(i,1)*q y+unit_vec(i,2)*q];
                                            %Position of current point
    P = P0+rho_f*(-g)*pos_point(2);         %Pressure at current point
    normal = [unit_vec(i,2) -unit_vec(i,1)];
                                            %Area normal vector for
                                            %current point

    Fp = dx*normal*P;                       %Force vector at current point
    Fp_local = T*Fp';                       %Transform to local
        coordinates

    pos_point_local = T*(pos_point-pos)';%Position on location coords.

    if i == 1 || i == 3
        num_x = num_x+pos_point_local(1)*Fp_local(2);
                                            %Numerator in cpx
        den_x = den_x+Fp_local(2);          %Denominator in cpx
        num_y = 0;                          %Numerator in cpy
        den_y = 0;                          %Denominator in cpy
    else
        num_x = 0;                          %Numerator in cpx
        den_x = 0;                          %Denominator in cpx
        num_y = num_y+pos_point_local(2)*Fp_local(1);
                                            %Numerator in cpy
        den_y = den_y+Fp_local(1);          %Denominator in cpy
    end
    Fp_vec(i,:) = Fp;                       %Save force vector

end
    cp(i,:) = [num_x/den_x num_y/den_y];    %Calculate cp
end

    cp(isnan(cp)) = 0;
    cp_vec(k) = sum(cp(:,1))/4;
    theta_vec(k) = theta;
    k = k+1;
end
```

```
plot(theta_vec*180/pi,cp_vec/L,'k'); hold on;
xlim([-90 90])
set(gca,'xtick',-90:15:90)

ylabel('Centre of pressure cp_{x´}/L [-]')
xlabel('Orientation angle \theta [deg]')
grid on
```