Simulations on Adapted Thomson Problem with Different Frames

Eleven (Shiyi) Chen sc7825@nyu.edu

May 19, 2021

Contents

1 Introduction

Thomson problem is determined the ground state of N electric charges on constrained in a sphere shell given the repeling effect given by the Coulomb's law[3]. By comparing the simulation result and continuum theory, M. Bowick et al. find out the crystalline structure in the ground state for the Thomson problem. In the meanwhile, Y. Levin and J. J. Arenzon generalized the Thomson problem replacing the sphere shell to a sphere[1]. They found that with small amount of charge, the charges are likely to go to the surface while for the large amount of charge, there are some charges stay in the middle of the shpere[2].

My work includes the two dimensional simulation on the Thomson problem and the three dimensional simulation on the generalized Thomson problem. While the original Thomson problem assumes on a sphere shell which is a two dimensional object, what if we change to other 2d shape, for example, the 2d square, will there still be any crystalline structure formation? Or follow the three dimensional by allowing charges moving freely within the sphere, how will the structure be like in the ground state, will there any charge stay inside of the shell?

1.1 Assumption of the model

In order to specify our problem, here I list the assumptions I made for my simulations:

- The frame has no charge and the each ball has equal amount of charge with random locations in the frame at time 0. And the frame is fixed which means it will not move as time goes by.
- The balls have volume and the collisions between ball and ball is imperfect collisions(obey the conservation of momentum but lose part of kinetic energy during collision) and it will exchange the electricity($q_{after} = (q1 + q2)/2$).
- The frame can absorb the electricity from the balls and distribute the it uniformly on the frame and the exchange of amount of charge will obey the law of conservation of charge, there will be a ratio introduced to quantify how much charge will be distributed to the frame and the ball.
- The shape of frame will be initially square, frame with other shapes may also be simulate(for example circle or ball in 3d).
- We consider the friction in the space as described in Section 2.4 except one ball in one frame case.
- No self-rotations for balls.
- No body force(Since we only consider the situation in space).

2 Equations of motion

2.1 Notations

The notations for this section is introduced in Table 2.1.

2.2 Coulomb Force

The Coulomb Force between the two balls is given in Equation 1. Similar to the law of the gravity, the magnitude of the Coulomb force is proportional to the $\frac{1}{r^2}$, while the direction of the force depends on what kind of charge the object has. In the Thomson problem, we have the homopolar charges, so the direction of the force repel two charges. I will includes more programming details in Section 3.

$$|F| = k\frac{|q_1 q_2|}{r^2} \quad k = 9 \times 10^9 \tag{1}$$

u, v	The speed of the ball a and ball b (m/s)
Cd	Drag coefficient
Dr	The new drag coefficient
$q1, q2$	Charge of the ball (C)
a	Recovery Coefficient
m	Mass of the ball (kg)
ρ	Density of the media (kg/m3)
A	Area of the ball (m2)
dt	Time step in the simulation (s)

Table 1: The notations used in this section

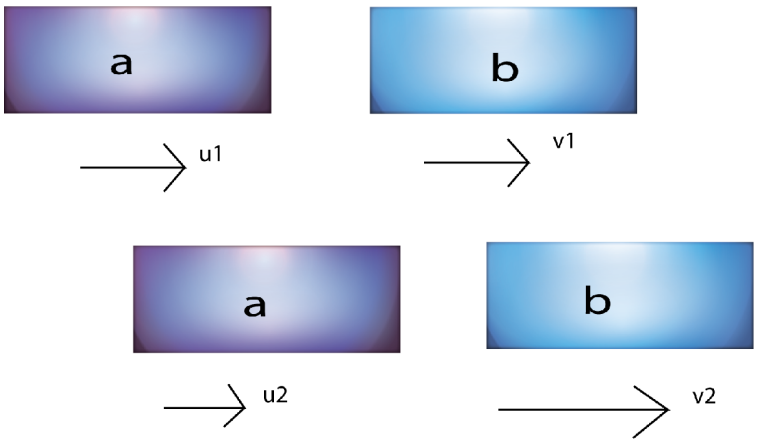

Figure 1: Schematic for the collision

2.3 Collision Equations

The collision equation in one dimension is listed in Equation 2. Since our assumption is that all the mass of the balls are the same, the mass drop out in the first equation. The second equation defines the recovery coefficient - a which ranges from 0 to 1. When $a = 0$, the collision is completely inelastic collision while $a = 1$ represents the elastic collision.

$$\begin{cases} u_1 + v_1 = u_2 + v_2 \\ a\,(u_1 - v_1) = u_2 - v_2 \end{cases} \tag{2}$$

2.4 Drag

In the typical Thomson problem, no drag is included in the system. I include the drag due to the defect of the Euler's Method which I will explain more in Section 3.1. Equation **??** shows how the drag is calculated. So that the speed change due to the drag can be written as

$$\frac{D \times dt}{m}$$

while D is the drag force calculated with

$$D = \mathrm{Cd} \times \frac{\rho \times A \times v^2}{2}.$$

Thus we can define a ew drag coefficient as

$$\mathrm{Dr} = \mathrm{Cd} \times \rho \times A \times \frac{\mathrm{dt}}{2m}.$$

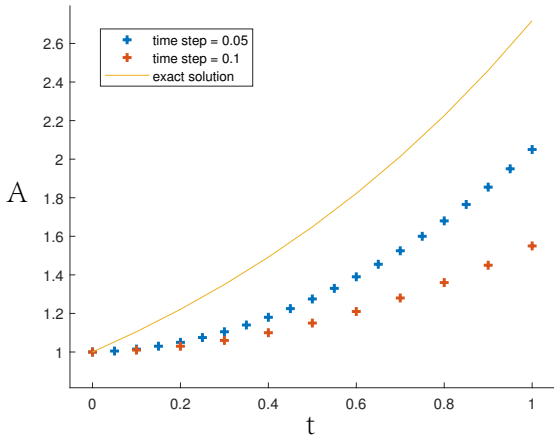

Figure 2: The numerical approximation in the example

So that

$$\frac{D \times dt}{m} = Dr \times v^2.$$

3 Numerical method

The numerical implementation in this project based on the Euler's method. Without the damping force to the system, the velocities of balls increases as time goes by. Although we can make the time step smaller to get a better result, to get a better result, I decide to simulate the process when the damping exist in the space.

3.1 Euler's Method

The Euler's method is a typical method to solve a differential equation. Considering the ODE below:

$$\frac{dA}{dt} = A \quad \text{with initial condition } A(0) = 1.$$

We now the exact solution is $A = e^x$, while for more complex ODE system, we may not be also to find the exact solution. In those kind of situations, a numerical method can be employed to approximate the exact solution. Euler's method is a typical numerical method which can help use solve the problem. In the ODE system above, we can approximate the exact solution by:

$$A(t + dt) = A(t) + Adt \quad \text{while } dt \text{ represents the time step}$$

By setting smaller dt, one can get a more accurate approximation. For the example above, the numerical solution is shown in Figure 2.

3.2 Newton's Second law and its Euler solution

Newton's Second law state that the force on the object is proportional to the product of its acceleration and mass, i.e.

$$F = ma = m\frac{dv}{dt}.$$

Applying to our model, the equations for the movement can described as:

$$\begin{cases} \dfrac{d\vec{x}}{dt} = \vec{v} \\ \dfrac{d\vec{v}}{dt} = \dfrac{\vec{F}}{m}. \end{cases} \tag{3}$$

$d\vec{x}$ and $d\vec{v}$ are the locations and speeds in cartesian cordinates. In 2d, they are 2d vectors while 3d vectors in 3d space.

For the program, I create a variable called `balls` which contains all the information of n balls. I also use a for loop to update the information every dt seconds. For the 2d simulation, the `balls` contains 7 rows. The last row is about the charge for each balls and the first six rows contains the information for locations, velocities, and accelerations in x and y directions.

The program terminates when the time step reached the larges time step. In each iteration, the program update the accelerations by calling two functions - `ball2ball` and `board2ball` which will be discussed in Section 4.1 and 4.2 respectively. Then using the new accelerations for each balls, the velocities, then locations will be updated sequentially. If any collision happens, the charge row will also be updated.

Below illustrates the main loop in my programs:

for i = 1:clockMax

$$\textbf{update balls} = \begin{matrix} \textbf{loc}_x \\ \textbf{loc}_y \\ \textbf{vel}_x \\ \textbf{vel}_y \\ \textbf{acc}_x \\ \textbf{acc}_y \\ \textbf{charge} \end{matrix} \begin{bmatrix} \textbf{ball}_1 & \textbf{ball}_2 & \textbf{ball}_3 & \dots & \dots & \textbf{ball}_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \textbf{ball}_1 & \textbf{ball}_2 & \textbf{ball}_3 & \dots & \dots & \textbf{ball}_n \end{bmatrix};$$

end

4 Implementation in the code

This project mainly coded in Matlab with some calculation done in Mathmatica. In this section, more programming implementation about the method will be provided.

4.1 Forces between ball and ball

The Matlab function `dist` is used to calculate the distances between the balls. The output of the `dist` function is the of the form:

$$\text{Dists} = \begin{bmatrix} 0 & d_{12} & \cdots & d_{1n} \\ d_{21} & 0 & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{\text{n1}} & d_{\text{n2}} & \cdots & 0 \end{bmatrix}.$$

This is equivalent to:

```
d = sum((x-y).^2).^0.5
```

while `x` and `y` represents the x coordinates and y coordinates of the balls.

The function `ball2ball` do the calculation for this and is listed in Appendix B show the entire function for the 3d simulation. Notice that we do not use any for loop in our code in order to enhance the runtime of the program.

4.2 Forces between board and ball

For the 3d sphere, we don't need to consider the force from the frame to the balls because the charge is uniformly distributed according to our assumptions. Since the uniform sphere shell doesn't exert any forces on the objects inside the sphere, we can skip this section for the 3d simulation. For the 2d simulation, we calculate the force to each ball one by one by doing the integration. The entire expression is calculated by Mathmatica and can be found in Appendix C.

Now, only consider the force from the bottom part of the frame and the ball locates at the position (a, b). The force can be calculated as

$$\int_{-\frac{L}{2}}^{\frac{L}{2}} k\frac{\lambda q}{(x-a)^2+b^2}\mathrm{dx} \quad \text{while } \lambda = \frac{q_{total}}{4L}. \tag{4}$$

By adding the four components from the wall, we can get the total force from the frame.

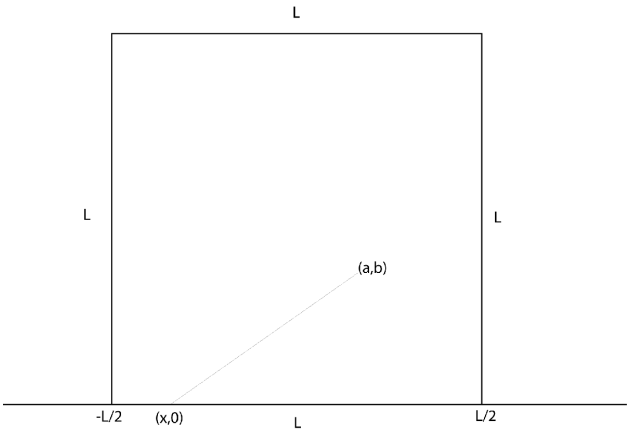

Figure 3: schematic diagram explaining the integration

4.3 Collision

The law of collision in one dimension is listed in Section 2.3, for two or three dimension, we only need to apply the same rules on each dimensions as shown in the code below.

```
% ball with ball collision
balls(3,I(i)) = (ux+vx)/2-a*(ux-vx)/2;
balls(3,J(i)) = (ux+vx)/2+a*(ux-vx)/2;
balls(4,I(i)) = (uy+vy)/2-a*(uy-vy)/2;
balls(4,J(i)) = (uy+vy)/2+a*(uy-vy)/2;

% electron merge
q = (balls(7,I(i))+balls(7,J(i)))/2;
```

```
balls(7,I(i)) = q;
balls(7,J(i)) = q;
```

For the collision between the walls and the balls, we just do the reflections, i.e.,

```
% collision with the verticle boundary
balls(4,I) = -a*balls(4,I);

% collision with the horz boundary
balls(3,J) = -a*balls(3,J);
```

5 Validations

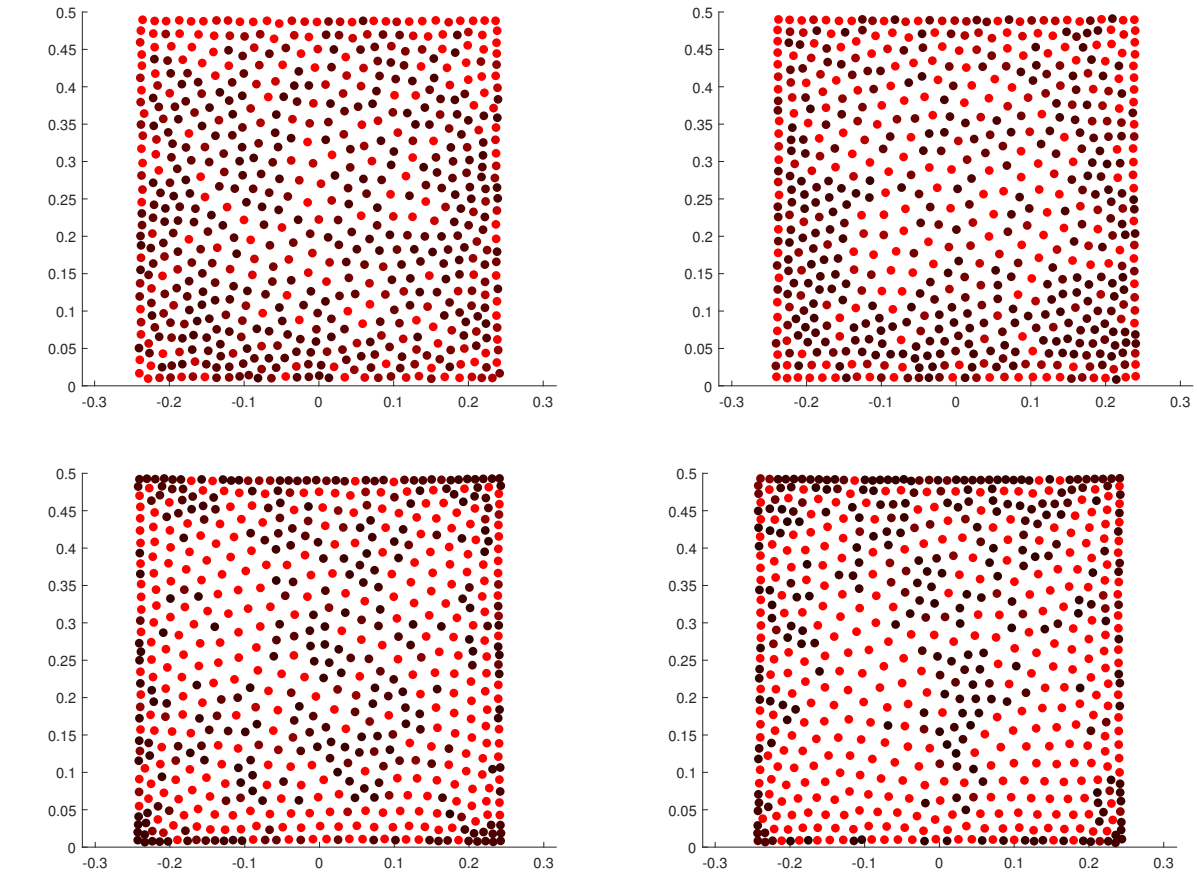

Figure 4: When Dr = 0.05, 0.1, 0.2, 0.8 at T = 100s

5.1 validation for Section 4.1

To validate the force between ball and ball, I fix 100 by 100 balls with equal amount of charge in the a one by one space. Since each ball has equal amount of charge, the force in the middle should be small in the sense that the forces were cancel out by the charges around them as shown in the left side of Figure 5.

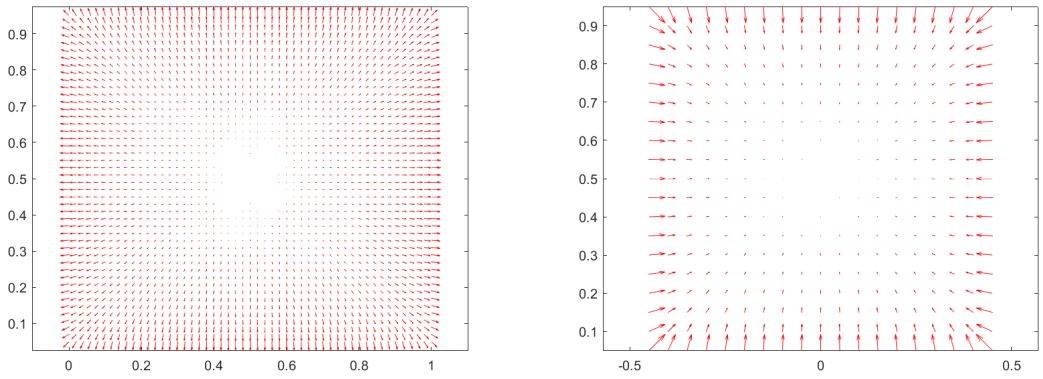

Figure 5: The forces and the electricity field

5.2 validation for Section 4.2

To validate the force from the frame to balls, the electricity field was plotted out as shown in the right side of Figure 5. In the middle part has a weaker field strength because the it cancels out from those in four different directions.

5.3 validation for the main program

The validation for the main loop in the program checks the one ball one frame simulation and look at the trajectory of the ball with different time step. Starting from $(0.1, 0.1)$, the orbits of the ball is shown in Figure 6.

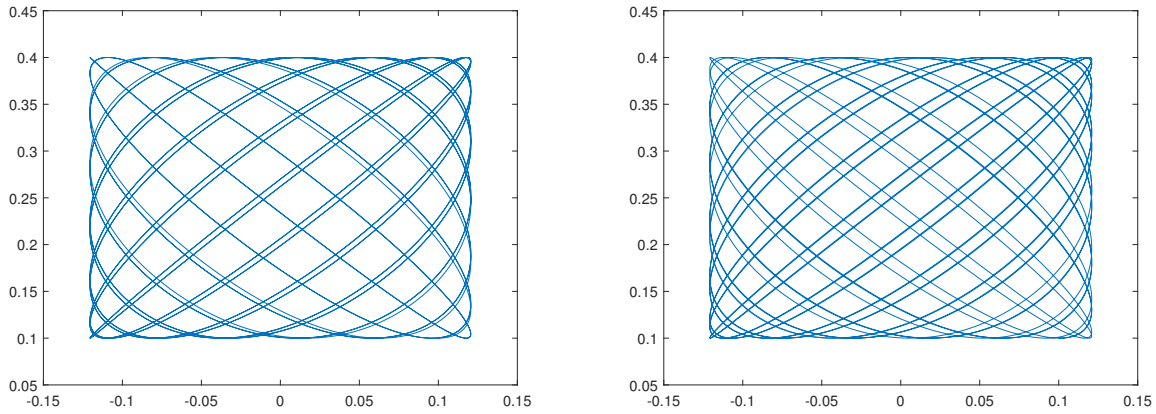

Figure 6: track of the balls with time step 0.01(left) and 0.001(right)

6 Results

6.1 2D simulation

By varying the new drag coefficient Dr, the patterns those balls form are as shown in Figure 4 when the simulation time is long enough. The balls were plotted based on how much charge the have, red represents more

charge while black represents little charge. At the final stage, balls are oscillating instead of being stationary.

6.2 3D simulation

For the collision, I only consider the collision between the balls and the frame without considering those between balls and balls. Other than that, the simulation is almost the same with the 2 d version. Figure 7 shows the result of the simulation for 1,000 balls. As shown from the Figure 7, most balls goes to the surface of the frame while there are few of them stay in the middle. As for small number of balls, Figure 8 shows the result. To show the geometry in 3d space, I connect the balls near by.

Figure 7: 3D simulation with 1,000 balls with Dr = 0.1 r=0.5m, the right side is the scaled version of the left side(some points on the sphere were cut off when zoomed in)

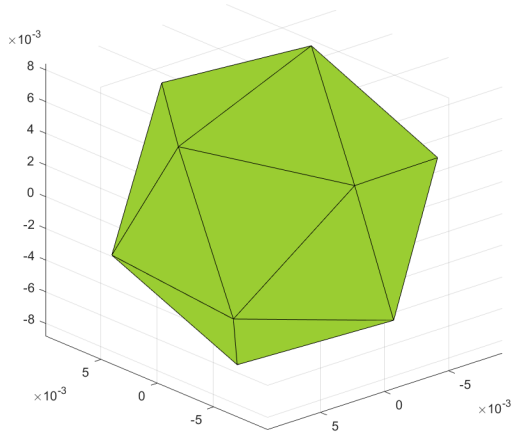

Figure 8: 3D simulation with 12 balls

7 Summary and Conclusions

In this project, adapted Thomson problem was simulated with 2d square frame and the 3d sphere frame. For the 2d simulation, different patterns was observed at T=100s. For the 3d simulation, with the number of balls being small, the balls will go to the surface while for the large amount of balls, this could be wrong.

References

[1] Mark Bowick et al. "Crystalline order on a sphere and the generalized Thomson problem". In: *Physical Review Letters* 89.18 (2002), p. 185502.

[2] Yan Levin and Jeferson J Arenzon. "Why charges go to the surface: A generalized Thomson problem". In: *EPL (Europhysics Letters)* 63.3 (2003), p. 415.

[3] Joseph John Thomson. "XXIV. On the structure of the atom: an investigation of the stability and periods of oscillation of a number of corpuscles arranged at equal intervals around the circumference of a circle; with application of the results to the theory of atomic structure". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 7.39 (1904), pp. 237–265.

Appendices

A Default variable values

Variable	value(in SI)	description
r	0.005	radius of the ball
L	0.5	length of the frame
m	0.002	mass of the ball
T	100	simulation time
dt	0.01	time step
q0	1e-8	initial charge for each ball
a	0.1	recovery coefficient
N	300	charge capacity of the frame
n	600	number of the balls
Dr	0.025	new drag coefficient
Plot	1	whether to plot
Xrn	L/2	half of the x length of frame
Yrn	L	y length of the frame
q1	0	initial charge on the frame

Table 2: parameters for the 2d simulation

B `ball2ball.m`

```
function [forces_x,forces_y,forces_z] = ball2ball(loc,q)
n = size(loc,1);
if size(q,1) ~= n
    forces_x= NaN;
    forces_y = NaN;
    forces_z= NaN;
    return
end

qq = q*q';
k = 9e9;
Dists = dist(loc');
% the matrix describe the reletive distance in x y z direction
Dxs = loc(:,1)'-loc(:,1);
Dys = loc(:,2)'-loc(:,2);
Dzs = loc(:,3)'-loc(:,3);

% The force in x y z direction
Forces_x = k.*qq.*Dxs./(abs(Dists).^3);
Forces_y = k.*qq.*Dys./(abs(Dists).^3);
Forces_z = k.*qq.*Dzs./(abs(Dists).^3);

% make the diagonal elements equal to 0
Forces_x(isnan(Forces_x)) = 0;
Forces_y(isnan(Forces_y)) = 0;
Forces_z(isnan(Forces_z)) = 0;

forces_x = sum(Forces_x);
forces_y = sum(Forces_y);
forces_z = sum(Forces_z);

end
```

C board2ball.m

```
function [x,y] = board2ball(a,b,L,q,lam)
k = 9e9;

x = k.*(a.^4+2.*a.^2.*(b+(-1).*L).*(b+L)+(b.^2+L.^2).^2).^(-1/2).*(( ...
  -1).*(b.^2+(a+(-1).*L).^2).^(1/2)+(b.^2+(a+L).^2).^(1/2)).*lam.*q;
y = b.^(-1).*k.*(a.*((-1).*(b.^2+(a+(-1).*L).^2).^(-1/2)+(b.^2+(a+L) ...
  .^2).^(-1/2))+L.*((b.^2+(a+(-1).*L).^2).^(-1/2)+(b.^2+(a+L).^2).^( ...
  -1/2))).*lam.*q;

%y = k.*lam.*q.*a.*(1./r+(L-a)./sqrt(b.^2+(L-a).^2))./b;
x(max((a<-L),(a>L))) = 0;
x(max((b<0),(b>L))) = 0;
y(max((a<-L),(a>L))) = 0;
y(max((b<0),(b>L))) = 0;
end
```

D 2d simulation

```
% simulation_4
% Consider the size of the ball and consider the collisions
% between balls
% add the side walls
% consider the damping force without video recorder

% % parameters
r = 0.005;
L = 0.5;
m = 0.002;
T = 100;
dt = 0.001;
q0 = 1e-8;
a = 0.1;
N = 300;
% Cd = 0.5;
DEN = 1.2;
if ~exist('n','var'); n = 600;      end
if ~exist('Dr','var'); Dr = 0.05;   end
if ~exist('Plot','var'); Plot = 1; end
if ~exist('Xrn','var'); Xrn = L/2;  end
if ~exist('Yrn','var'); Yrn = L;   end
if ~exist('q1','var'); q1 = 0;      end
q0
Dr
% initial state of the balls
if ~exist('balls','var')
    balls = [2*Xrn*rand(1,n)-Xrn;Yrn*rand(1,n)];
    [f_x,f_y] = ball2ball(balls',q0*ones(n,1));
    balls = [balls; zeros(2,n);f_x./m;f_y./m;q0*ones(1,n)];
end
t = 0;
Q = [];
E =[];
xs = [];
ys = [];

% plotting part
if Plot ==1
    n = size(balls,2);
    fig = scatter(balls(1,:),balls(2,:), [],balls(7,:)'.* [balls(7,:)'./q0^2,ones(n,2)], '
    filled');
    xlim([-Xrn Xrn])
    ylim([0 Yrn])
end

```

```
while t<T
    % Merge the electron of the balls
    Dists = dist(balls(1:2,:));
    Dists = triu(Dists);
    [I,J] = find(Dists<2*r&Dists~=0);
    xs = [xs, balls(1,1)];
    ys = [ys, balls(2,1)];

    % consider the collision of the balls
    if ~isempty(I)
        for i = 1:length(I)
            % Move the balls because of the round off error
            Dist = Dists(I(i),J(i));
            xlen = 2*r*(balls(1,I(i))-balls(1,J(i)))/Dist;
            ylen = 2*r*(balls(2,I(i))-balls(2,J(i)))/Dist;
            x_mid = (balls(1,I(i))+balls(1,J(i)))/2;
            y_mid = (balls(2,I(i))+balls(2,J(i)))/2;

            if abs(Dist)>=r
                balls(1,I(i)) = x_mid+xlen/2;
                balls(2,I(i)) = y_mid+ylen/2;
                balls(1,J(i)) = x_mid-xlen/2;
                balls(2,J(i)) = y_mid-ylen/2;
            else
                balls(1,I(i)) = x_mid+xlen/2+r;
                balls(2,I(i)) = y_mid+ylen/2+r;
                balls(1,J(i)) = x_mid-xlen/2-r;
                balls(2,J(i)) = y_mid-ylen/2-r;
            end

            % electron merge
            q = (balls(7,I(i))+balls(7,J(i)))/2;
            balls(7,I(i)) = q;
            balls(7,J(i)) = q;

            % retrieve the speeds
            ux = balls(3,I(i));
            uy = balls(4,I(i));
            vx = balls(3,J(i));
            vy = balls(4,J(i));

            % recalculate the speeds
            balls(3,I(i)) = (ux+vx)/2-a*(ux-vx)/2;
            balls(4,I(i)) = (uy+vy)/2-a*(uy-vy)/2;
            balls(3,J(i)) = (ux+vx)/2+a*(ux-vx)/2;
            balls(4,J(i)) = (uy+vy)/2+a*(uy-vy)/2;
        end
    end

    % Merge the ball with the board
    I = find(max(balls(2,:)<r,balls(2,:)>Yrn-r));
    if ~isempty(I)
        q = sum([balls(7,I),q1])./(N+length(balls(7,I)));
        balls(7,I) = q;
        q1 = N*q;
        balls(4,I) = -a*balls(4,I);
    end

    J = find(balls(1,:)<-Xrn+r|balls(1,:)>Xrn-r);
    if ~isempty(J)
        q = sum([balls(7,J),q1])./(N+length(balls(7,J)));
        balls(7,J) = q;
        q1 = N*q;
        balls(3,J) = -a*balls(3,J);
    end
    [fx,fy] = force(balls,Xrn,Yrn,q1,m);
    balls(5,:) = fx;
    balls(6,:) = fy;
```

```
114
115     % the damping force propotional to the speed
116     balls(3,:) = balls(3,:) -Dr*balls(3,:).^2.*sign(balls(3,:));
117     balls(4,:) = balls(4,:) - Dr*balls(4,:).^2.*sign(balls(4,:));
118
119     %update the velocity
120     balls(3,:) = balls(5,:).*dt+balls(3,:);
121     balls(4,:) = balls(6,:).*dt+balls(4,:);
122
123     % If touch the walls, do the reflection
124     balls(1,balls(1,:)<r-Xrn) = 2*r-2*Xrn-balls(1,balls(1,:)<r-Xrn);
125     balls(1,balls(1,:)>Xrn-r) = 2*Xrn-2*r-balls(1,balls(1,:)>Xrn-r);
126     balls(2,balls(2,:)>Yrn-r) = 2*Yrn-2*r-balls(2,balls(2,:)>Yrn-r);
127     balls(2,balls(2,:)<r) = 2*r-balls(2,balls(2,:)<r);
128
129     % If the ball is too far, delete it
130     q1 = q1+ sum(balls(7,balls(1,:)<-Xrn|balls(1,:)>Xrn|balls(2,:)<0|balls(2,:)>Yrn));
131     balls(:,balls(1,:)<-Xrn|balls(1,:)>Xrn|balls(2,:)<0|balls(2,:)>Yrn)=[];
132     disp(t/T)
133
134     % update the position
135     balls(1,:) = balls(3,:).*dt+balls(1,:);
136     balls(2,:) = balls(4,:).*dt+balls(2,:);
137
138     %plotting
139    if Plot == 1
140         clf
141         n = size(balls,2);
142         scatter(balls(1,:),balls(2,:), [], balls(7,:)'.*[balls(7,:)'./q0^2,ones(n,2)], '
     filled');
143         axis([-Xrn, Xrn,0,Yrn])
144         axis equal
145         drawnow
146    end
147
148     t = t+dt;
149     Q = [Q sum([balls(7,:),q1])];
150     E = [E sum(balls(3,:).^2+balls(4,:).^2)];
151 end
```

E 3d simulation

```
1  % For the spherical frame work
2
3  % initial set up for the cluster
4  clear
5  T = 50;
6  dt = 0.01;
7  N = floor(T/dt);
8  R = 0.03;
9  a = 1.5*R;
10 q0 = 1e-7;
11 n = 300;
12 mass = 1e2;
13 ratio = 100;
14 qc = 0;
15 col = 1;
16
17 % location of the balls
18 x = zeros(1,n);
19 y = zeros(1,n);
20 z = zeros(1,n);
21 r = zeros(1,n);
22 E = zeros(1,N);
23 q = q0*ones(n,1);
24 Dr = 0.1;
25
```

```
for i=1:n
    r(i)=2*R;

    % put the stars in the sphere with radius R
    while r(i) > R
        x(i) = R*2*(rand-0.5);
        y(i) = R*2*(rand-0.5);
        z(i) = R*2*(rand-0.5);
        r(i) = sqrt((x(i))^2 + (y(i))^2 + (z(i))^2);
    end
end


% velocity of the balls
vx = zeros(1,n);
vy = zeros(1,n);
vz = zeros(1,n);

% plot the initial set up
zscaled = abs(q);
%cn = ceil(max(abs(zscaled)));
cn = ceil(max(q));
cm = colormap(winter(cn));
fig = scatter3(x,y,z, [], cm(ceil(zscaled),:), 'filled');
axis equal
axis([-a a -a a -a a]);

for i = 1:N
    [Fx, Fy, Fz] = ball2ball([x',y',z'],q);
    vx = vx+dt.*Fx./mass-Dr.*vx.^2.*sign(vx);
    vy = vy+dt.*Fy./mass-Dr.*vy.^2.*sign(vy);
    vz = vz+dt.*Fz./mass-Dr.*vz.^2.*sign(vz);
    x = x+dt.*vx;
    y = y+dt.*vy;
    z = z+dt.*vz;
    rr = sqrt(x.^2+y.^2+z.^2);

    % if touch the boundary, do the reflection
    inx = find(sqrt(x.^2+y.^2+z.^2)>R);
    for j = inx
        r = sqrt(x(j)^2+y(j)^2+z(j)^2);
        x(j) = x(j)*R/r; y(j) = y(j)*R/r; z(j) = z(j)*R/r;
        m = [vx(j);vy(j);vz(j)]; n = [x(j);y(j);z(j)];
        nm = m-(2*(m'*n)./(n'*n))*n;
        vx(j) = col*nm(1); vy(j) = col*nm(2); vz(j) = col*nm(3);
        qtotal = qc+q(j);
        qc = qtotal*(ratio/(ratio+1));
        q(j) = qtotal/(ratio+1);
    end
    inx = find(x<0&y<0&z<0);
%     fig.XData = x(inx);
%     fig.YData = y(inx);
%     fig.ZData = z(inx);

%     fig.XData = x;
%     fig.YData = y;
%     fig.ZData = z;
%     drawnow

    scatter3(x,y,z,[],[zeros(size(x,2), 2),rr'./R],'filled');
    axis equal
    axis([-a a -a a -a a]);

    drawnow
    E(i) = sum(vx.^2+vy.^2+vz.^2);
end
```