```
%%Collisions in two-dimensions
% Note: I have assumed that you are storing all positions in one vector
% x = (x1 y1 x2 y2 . . . xN yN) with each row representing a timestep,
% such that the x-position of particle j
% at timestep i is stored as x(i, 2*j)
% and the y-position is x(i, 2*j+1).
% Similarly, the x-component of the velocity of particle j
% at timestep i is v(i, 2*j)
% and the y-component of the velocity is v(i, 2*j+1).
% Please modifiy the code below as necessary.

for i=2:M   % where M is the total number of timesteps
            % and i = 1 represents
            % the initial positions/velocities

% CHECK FOR COLLISIONS AND UPDATE VELOCITY:
    for j=1:N             % For each of the N spheres,
        for k=j+1:N       % check the distance to
                          % all remaining spheres.
           dsquared = (x(i,2*j)-x(i,2*k))^2 + (x(1,2*j+1)-x(i,2*k+1))^2
                          %distance between centers squared
           if dsquared < 4*r^2
                          % If the distance between
                          % centers squared is less than (2r)^2,
                          % then they collide.

            % Find the unit vector from sphere j to sphere k:
            djk=[x(i,2*j)-x(i,2*k)  x(i,2*j+1)-x(i,2*k+1)];
            unitdjk=djk/norm(djk);

            % Velocity of sphere j in the "centerline"
            %direction of unitdjk:
            vjc=dot(v(i,2*j:2*j+1),unitdjk)*unitdjk;
            % Velocity of sphere j perpendicular to unitjk:
            vjperp=v(i,2*j:2*j+1)-vjc;

            % Velocity of sphere k in the "centerline"
            % direction of unitdjk:
            vkc=dot(v(i,2*k:2*k+1),unitdjk)*unitdjk;
            % Velocity of sphere k perpendicular to unitjk:
            vkperp=v(i,2*k:2*k+1)-vkc;

            % To obtain final velocities, swap "centerline" velocities
            % as follows:
            v(i,2*j:2*j+1)=vkc+vjperp;
            v(i,2*k:2*k+1)=vjc+vkperp;

            end
        end
    end

end
```