Collisions of particles, comparison of hard-sphere and Discrete Element Method (soft-sphere)

Siddharthan Somasundaram (Matriculation number : 254304)[a,1]

[a]*Otto von Guericke University, Magdeburg, Germany*

Submitted as part of the coursework for Simulations of Mechanical Processes (WiSe 24/25).

Abstract—This study compares the hard-sphere and soft-sphere (Discrete Element Method, DEM) approaches for simulating particle collisions. The hard-sphere model assumes instantaneous collisions, determining post-collision velocities analytically based on mass and initial velocities. The soft-sphere model considers interactions over a finite duration by calculating forces based on particle overlap. MATLAB simulations analyze the collision of two particles under specified initial conditions using a numerical integration scheme to track particle motion over time. The comparison focuses on collision times, post-collision velocities, and contact points for varying coefficients of restitution. The study provides insights into the computational efficiency and practical applications of each approach.

Contents

1. Introduction

Accurate modeling of particle collisions is crucial in fields such as granular flow and material processing. Two common approaches used for collision modeling are the hard-sphere and soft-sphere (DEM) methods. The hard-sphere model assumes that collisions occur instantaneously without deformation, making it computationally efficient but less accurate. In contrast, the soft-sphere method accounts for interactions over a finite duration by considering particle overlap to compute contact forces, offering a more detailed representation of collision behavior.

This study investigates the collision dynamics of two identical particles under specified initial conditions using both methods. MATLAB simulations are used to analyze collision times, post-collision velocities, and contact points for different coefficients of restitution. A comparative analysis highlights the strengths and limitations of each approach.

1.1. Objective

This study aims to compare the hard-sphere and soft-sphere (DEM) approaches for simulating particle collisions by

- Determining the time of first contact and the coordinates of the contact points before and after the collision.
- Calculating the post-collision velocities of the particles for different coefficients of restitution.
- Evaluating the duration of the collision in the soft-sphere model.
- Implement MATLAB simulations to analyze and compare the efficiency and practical applicability of both approaches.

1.2. Methods

Hard-Sphere Algorithm:

In the hard-sphere model, collisions are assumed to occur instantaneously with no deformation. The time of first contact is calculated analytically based on the initial positions, velocities, and diameters of the particles. Post-collision velocities are determined using momentum conservation principles and the coefficient of restitution. The coordinates of the contact point before and after the collision are derived from the relative motion of the particles. This approach offers a computationally efficient solution by simplifying collision dynamics without considering force interactions over time.

Soft-Sphere (DEM) Algorithm:

The soft-sphere approach models particle interactions over a finite duration by calculating forces based on overlap between particles. A MATLAB function is implemented to compute the overlap and normal direction, which is used to determine the contact force according to a linear force model. The Leapfrog integration scheme is applied to iteratively update particle positions and velocities at each time step, using a predictor-corrector approach. The coefficient of restitution is considered by defining separate loading and unloading stiffness values. The duration of the collision is evaluated by monitoring the evolution of the overlap throughout the simulation.

Comparative Analysis:

The two approaches are compared in terms of collision times, post-collision velocities, contact points, and computation time. The computational efficiency of the analytical hard-sphere model is assessed against the iterative nature of the soft-sphere method. In addition, the effects of varying restitution coefficients on collision dynamics are analyzed to evaluate the strengths and limitations of each approach in practical applications.

1.3. Report Structure

The report is structured as follows:
Section 2: Problem Statement.
Section 3: Methodology.
Section 4: Visualization.
Section 5: Conclusions

2. Problem Statement

Consider two particles at time $t = 0$, placed at $x_1 = (0, 0, 0)$ m and $x_2 = (1.1, 1.3, 0)$ m. Both particles have a diameter of 1.0 m and a mass of 0.05 kg. The initial velocities are $v_1 = (0, 0, 0)$ m/s for the first particle and $v_2 = (-1.0, -1.0, 0.0)$ m/s for the second particle. Rotation effects are neglected.

1. Determine the time at which the particles collide (first contact).
2. Using the **hard-sphere algorithm**, determine the following:
   (a) The coordinates of the point where the two particles make contact before the collision.
   (b) The post-collision velocities of the two particles if the coefficient of restitution is $e = 1.0$.
   (c) The post-collision velocities of the two particles if the coefficient of restitution is $e = 0.75$.

(d) The duration of the collision when the coefficient of restitution is 0.75.
(e) The coordinates of the point where the two particles make contact after the collision.

3. Write a MATLAB function that takes two arbitrary particle positions and their diameters and returns the **overlap** (scalar) and the **normal** (vector).
4. Write a MATLAB program that takes the initial conditions of the particles as specified above and moves particle 2 with a fixed time step $\Delta t = 1.0 \times 10^{-4}$ s using the **Leapfrog algorithm**. The velocity update should be split into two parts (predictor-corrector). Provide a brief description of your code.
5. After each time step, evaluate the overlap. If overlap occurs, compute the normal force based on the linear model:

$$\mathbf{F_n} = -k\delta\mathbf{n}$$

where the loading stiffness k is 750 N/m. Apply this force in the correct direction on both particles. Implement the code to consider the coefficient of restitution by defining different loading and unloading spring constants using:

$$e = \sqrt{\frac{k_n^{\text{unloading}}}{k_n^{\text{loading}}}}$$

Using the **soft-sphere algorithm** and the MATLAB functions written **in** questions **3** and **4**, determine:

(a) The coordinates of the point where the two particles make contact before the collision.
(b) The post-collision velocities of the two particles if the coefficient of restitution is $e = 1.0$.
(c) The post-collision velocities of the two particles if the coefficient of restitution is $e = 0.75$.
(d) The duration of the collision according to the DEM approach when the coefficient of restitution is 0.75.
(e) The coordinates of the point where the two particles make contact after the collision.

6. Describe the similarities and differences between the two approaches with respect to colliding particles.

3. Methodology

This section outlines the numerical methods used to model particle collisions using both the **hard-sphere** and **soft-sphere (Discrete Element Method, DEM)** approaches. The equations governing the particle motion consider forces such as contact forces and restitution effects, which are implemented using analytical and numerical techniques.

Simulations are performed under the given initial conditions to determine collision times, post-collision velocities, and contact points for different coefficients of restitution.

A comparative analysis is conducted to evaluate the differences in collision outcomes, computational efficiency, and accuracy of both approaches.

3.1. Collision Time:

The time at which the two particles collide (first contact) is determined analytically by evaluating when they come within a distance equal to half the sum of their diameters:

$$\frac{1}{2}(d_1 + d_2)$$

The time of collision is calculated using the following quadratic equation:

$$t_{col} = \frac{-\mathbf{x}_{12} \cdot \mathbf{v}_{12} \pm \sqrt{(\mathbf{x}_{12} \cdot \mathbf{v}_{12})^2 - \mathbf{v}_{12}^2 \cdot \left(\mathbf{x}_{12}^2 - \left(\frac{1}{2}d_1 + \frac{1}{2}d_2\right)^2\right)}}{\mathbf{v}_{12}^2}$$

where:

- $\mathbf{x}_{12} = \mathbf{x}_1 - \mathbf{x}_2$ is the relative position between the two particles.
- $\mathbf{v}_{12} = \mathbf{v}_1 - \mathbf{v}_2$ is the relative velocity between the two particles.
- d_1 and d_2 are the diameters of the particles.

3.2. Hard-Sphere Algorithm Analysis

Using the **hard-sphere algorithm**, the following aspects are determined:

a) **Coordinates of the contact point before the collision:**
Using the previously calculated collision time t_{col}, the positions of the particles are updated using the Euler method:

$$\mathbf{x}_{1,\text{pre-collision}} = \mathbf{x}_{1,initial} + \mathbf{v}_{1,initial} t_{\text{col}}$$

$$\mathbf{x}_{2,\text{pre-collision}} = \mathbf{x}_{2,initial} + \mathbf{v}_{2,initial} t_{\text{col}}$$

The contact point before the collision is then calculated as the midpoint between the updated particle positions:

$$\mathbf{x}_{\text{contact, pre-collision}} = \frac{\mathbf{x}_{1,\text{pre-collision}} + \mathbf{x}_{2,\text{pre-collision}}}{2}$$

b) **Post-collision velocities (coefficient of restitution = 1):**
For elastic collisions, the velocities are calculated using the impulse-based approach:

$$j = \frac{-(1+e)(\mathbf{v}_{12} \cdot \mathbf{n})}{\frac{1}{m_1} + \frac{1}{m_2}}$$

The post-collision velocities are then given by:

$$v_{1,f} = v_{1,i} + \frac{j}{m_1}\mathbf{n}$$

$$v_{2,f} = v_{2,i} - \frac{j}{m_2}\mathbf{n}$$

where:

- $\mathbf{n}$ is the unit normal vector between the two particles at the contact point.
- $\mathbf{v}_{12} = \mathbf{v}_1 - \mathbf{v}_2$ is the relative velocity between the two particles.
- $e = 1.0$ is the coefficient of restitution.

c) **Post-collision velocities (coefficient of restitution = .75):**
In the case of inelastic collisions, the post-collision velocities are determined using an impulse-based approach. This method is analogous to that used for elastic collisions, but employs a coefficient of restitution of $e = 0.75$ to account for energy loss.

d) **Collision duration according to the hard-sphere model for coefficient of restitution of $e = 0.75$:**
The hard-sphere model simplifies particle interactions by treating them as impenetrable spheres. When two spheres collide, they interact instantaneously, changing their velocities and directions without any actual contact or deformation. A coefficient of restitution of 0.75 indicates an inelastic collision where some kinetic energy is lost. However, this energy loss does not change the fundamental assumption of instantaneous collisions in the hard-sphere model.

$$t_{\text{collision duration}} = 0$$

e) **Coordinates of the contact point after the collision:**
In the hard-sphere model, the contact point after the collision remains the same as the initial contact point because the collision is considered instantaneous. Since no further interactions occur post-collision, the final contact point is the midpoint of the particle positions at the moment of impact:
Using the previously calculated collision time t_{col} and t_{col}, the positions of the particles are updated using the Euler method:

$$\mathbf{x}_{1,\text{post-collision}} = \mathbf{x}_{1,initial} + \mathbf{v}_{1,initial}(t_{\text{col}} + t_{\text{collision duration}})$$

$$\mathbf{x}_{2,\text{post-collision}} = \mathbf{x}_{2,initial} + \mathbf{v}_{2,initial}(t_{\text{col}} + t_{\text{collision duration}})$$

$$\mathbf{x}_{\text{contact, post collision}} = \frac{\mathbf{x}_{1,\text{post-collision}} + \mathbf{x}_{2,\text{post-collision}}}{2}$$

This calculation is valid for both elastic and inelastic collisions, as the particles separate instantaneously according to their post-collision velocities.

3.3. Overlap and Normal Vector Calculation:

The overlap between the two particles is calculated on the basis of their relative positions and diameters. The overlap (scalar) is computed as the difference between the sum of the radii and the actual distance between the particles' centers:

$$\delta = \max\left(0, \frac{d_1 + d_2}{2} - \|\mathbf{x}_2 - \mathbf{x}_1\|\right)$$

The normal vector is a unit vector pointing from the first particle to the second particle, calculated as:

$$\mathbf{n} = \frac{\mathbf{x}_2 - \mathbf{x}_1}{\|\mathbf{x}_2 - \mathbf{x}_1\|}$$

Overlap and Normal Vector Calculation MATLAB Implementation:

Implementation: The MATLAB function `compute_overlap_and_normal` calculates the overlap and normal vector using the given particle positions and diameters. The function performs the following steps:

- Compute the relative position vector between the two particles.
- Calculate the Euclidean distance between the particles.
- Determine the overlap by comparing the collision distance with the actual separation.
- Calculate the unit normal vector, ensuring numerical stability if the particles are exactly at the same position.

If no overlap occurs, the function returns an overlap value of zero. The normal vector is computed only when the distance is greater than zero; otherwise, it defaults to a zero vector.

3.3.1. overlap and normal Matlab function:

```
function [overlap, normal] =
    compute_overlap_and_normal(pos1, pos2,
    diameter1, diameter2)

    % Compute the distance vector between the
    centers of the two particles
    distance_vector = pos2 - pos1;

    % Compute the distance (magnitude of the
    distance vector)
    distance = norm(distance_vector);

    % Compute the collision distance (sum of
    radii)
    collision_distance = 0.5 * (diameter1 +
    diameter2);

    % Compute overlap (scalar value)
    overlap = max(0, collision_distance -
    distance);

    % Compute the normal vector (unit vector
    pointing from particle 1 to particle 2)
    tolerance = 1e-6; % Small tolerance for
    floating-point comparisons
    if distance > tolerance
        normal = distance_vector / distance;
    else
        % If the distance is very small (near
    perfect overlap), set normal to zero
        normal = [0; 0; 0]; % Corrected to
    column vector for consistency
    end
end
```

Code 1. compute overlap and normal.

3.4. Leapfrog Integration for Particle Motion:

The Leapfrog integration method is used to numerically simulate the motion of particles under the specified initial conditions. The motion is updated with a fixed time step $\Delta t = 1.0 \times 10^{-4}$ s, and the velocity is updated in two parts: predictor and corrector.

1. **Predictor Step:** The velocity is updated by half a time step using the forces acting on the particle.
2. **Drift Step:** The position is updated using the half-step velocity.
3. **Corrector Step:** The velocity is fully updated based on the new forces calculated after the drift.

Leapfrog Integration MATLAB Implementation:

Implementation: The MATLAB function `compute_ss_leapfrog` simulates the motion of two particles by updating their positions and velocities iteratively using the Leapfrog algorithm. The function takes the initial positions and velocities of the particles along with other parameters such as mass, diameter, and coefficient of restitution.

The function workflow includes:

- Preallocating arrays for positions, velocities, overlaps, and normal vectors.
- Iterating through time steps to perform predictor, drift, and corrector updates.
- Computing forces during a collision if an overlap is detected.
- Updating acceleration, velocity, and position iteratively.

If no collision is detected, the forces are set to zero, and the particles continue their motion under their initial conditions.

3.4.1. Leapfrog MATLAB Function:

```
% Performs a leapfrog integration to simulate
    the collision of two particles.
function [positions1, velocities1, positions2,
    velocities2, overlaps, normals] =
    compute_ss_leapfrog(position1, velocity1,
    position2, velocity2,mass1, mass2, diameter1
    , diameter2, time_step, total_time, CoR)
    % Number of time steps
    num_steps = ceil(total_time / time_step);

    % Preallocate arrays for positions and
    velocities
    positions1 = zeros(num_steps, 3); % Particle
     1 positions
    velocities1 = zeros(num_steps, 3); %
    Particle 1 velocities
```

```
    positions2 = zeros(num_steps, 3); % Particle 2 positions
    velocities2 = zeros(num_steps, 3); % Particle 2 velocities
    % Preallocate arrays for overlap and normal
    overlaps = zeros(num_steps, 1);    % Overlap at each timestep
    normals = zeros(num_steps, 3);     % Normal vector at each timestep
    % Initialize positions and velocities
    positions1(1, :) = position1;
    velocities1(1, :) = velocity1;
    positions2(1, :) = position2;
    velocities2(1, :) = velocity2;

    % Loop over time steps
    for i = 1:num_steps-1
        % Compute overlap and normal vector
        [overlaps(i), normals(i,:)] = compute_overlap_and_normal(positions1(i, :), positions2(i, :), diameter1, diameter2);
        if overlaps(i) > 0
            % Compute forces during collision
            [force1, force2] = compute_collision_forces(velocities1(i, :), velocities2(i, :), ...
             CoR, overlaps(i), normals(i,:));
        else
            % No collision, no forces
            force1 = [0, 0, 0];
            force2 = [0, 0, 0];
        end
        % Step 1: Compute initial acceleration
        acceleration1 = force1 / mass1;
        acceleration2 = force2 / mass2;

        % Step 2: Kick (Predictor) - Half step velocity update
        velocity1_half = velocities1(i, :) + 0.5 * acceleration1 * time_step;
        velocity2_half = velocities2(i, :) + 0.5 * acceleration2 * time_step;

        % Step 3: Drift - Update positions
        positions1(i+1, :) = positions1(i, :) + velocity1_half * time_step;
        positions2(i+1, :) = positions2(i, :) + velocity2_half * time_step;

        [overlaps(i+1), normals(i+1,:)] = compute_overlap_and_normal(positions1(i+1, :), ...
             positions2(i+1, :), diameter1, diameter2);
        if overlaps(i+1) > 0
            % Compute forces during collision
            [force1, force2] = compute_collision_forces(velocity1_half, velocity2_half, ...
             CoR, overlaps(i+1), normals(i+1,:));
        else
            % No collision, no forces
            force1 = [0, 0, 0];
            force2 = [0, 0, 0];
        end
        % Step 5: Compute new acceleration
        acceleration1 = force1 / mass1;
        acceleration2 = force2 / mass2;

        % Step 6: Kick (Corrector) - Half step velocity update
        velocities1(i+1, :) = velocity1_half + 0.5 * acceleration1 * time_step;
        velocities2(i+1, :) = velocity2_half + 0.5 * acceleration2 * time_step;
    end
end
```

Code 2. Leapfrog Integration for Particle Motion

3.5. Soft-Sphere Algorithm Analysis:

Using the **soft-sphere algorithm**, the following aspects are determined:

a) **Coordinates of the contact point before the collision:**
The initial contact point is determined when the overlap between particles becomes positive. The contact point is calculated as the midpoint of the particle positions:

$$\mathbf{x}_{\text{contact, pre-collision}} = \frac{\mathbf{x}_{1,\text{overlap-start}} + \mathbf{x}_{2,\text{overlap-start}}}{2}$$

The overlap and normal vector are calculated as described in the respective sections. The contact point corresponds to the timestep when overlap first becomes positive.

b) **Post-collision velocities (coefficient of restitution = 1):**
For elastic collisions, the interaction force is modeled using a linear spring:

$$\mathbf{F} = k\delta\mathbf{n}$$

where k is the spring constant, δ is the overlap, and $\mathbf{n}$ is the unit normal vector. The leapfrog integrator is used to update velocities and positions during the collision. The acceleration of each particle is calculated as:

$$\mathbf{a}_i = \frac{\mathbf{F}}{m_i}$$

where m_i is the mass of the particle. The velocities and positions are updated iteratively using the leapfrog scheme.

c) **Post-collision velocities (coefficient of restitution = .75):**
For inelastic collisions, energy dissipation is incorporated using different spring constants for loading and unloading phases:

$$k_{\text{unloading}} = e^2 k_{\text{loading}}$$

the conditions for the loading and the unloading are

$$\begin{aligned} \text{loading:} \quad & \mathbf{v}_{1,2} \cdot \mathbf{n} < 0, \\ \text{unloading:} \quad & \mathbf{v}_{1,2} \cdot \mathbf{n} > 0. \end{aligned}$$

During the collision, the interaction force is computed as:

$$\mathbf{F} = \begin{cases} k_{\text{loading}}\delta\mathbf{n}, & \text{during loading phase} \\ k_{\text{unloading}}\delta\mathbf{n}, & \text{during unloading phase} \end{cases}$$

The leapfrog integrator is used to update the velocities and positions, accounting for the changing spring constants during loading and unloading.

d) **Collision duration:**
The duration of the collision is calculated as the time interval between the first timestep where overlap becomes positive and the last timestep where overlap returns to zero:

$$t_{\text{collision}} = (i_{\text{overlap-end}} - i_{\text{overlap-start}})\Delta t$$

where $i_{\text{overlap-start}}$ and $i_{\text{overlap-end}}$ are the indices of the first and last timesteps with positive overlap, respectively, and Δt is the timestep size.

e) **Coordinates of the contact point after the collision:**
The final contact point is determined at the last timestep with positive overlap. It is calculated as the midpoint of the particle positions in that timestep:

$$\mathbf{x}_{\text{contact, post-collision}} = \frac{\mathbf{x}_{1,\text{overlap-end}} + \mathbf{x}_{2,\text{overlap-end}}}{2}$$

3.6. Hard-Sphere and Soft-Sphere Model Comparison

The behavior of two colliding particles is analyzed using the hard-sphere and soft-sphere models. The similarities and differences between the models, particularly concerning their impact on the behavior of the simulated particles, are summarized below.

Similarities in Modeling Particle Collisions

- **Conservation Laws:** Both models satisfy the conservation of linear momentum and account for the coefficient of restitution (e), determining the post-collision velocities based on initial conditions.
- **Collision Detection:** In both models, the moment of impact is determined by the relative positions of the colliding particles.
- **Energy Considerations in Collisions:** Both models conserve kinetic energy in perfectly elastic collisions ($e = 1.0$) and dissipate kinetic energy in inelastic collisions ($e < 1.0$).

Differences in Collision Dynamics and Computational Approach

- **Collision Duration:**
  - **Hard-Sphere Model:** Assumes instantaneous collisions with no contact duration, simplifying calculations.
  - **Soft-Sphere Model:** Accounts for finite collision duration where interaction forces act during the overlap phase.
- **Force Interaction Modeling:**
  - **Hard-Sphere Model:** Uses an impulse-based method for updating post-collision velocities, without explicit force calculations.
  - **Soft-Sphere Model:** Models interparticle forces using a spring-dashpot system dependent on particle overlap.
- **Handling of Particle Overlap:**
  - **Hard-Sphere Model:** Assumes particles do not overlap; they instantly separate post-collision.
  - **Soft-Sphere Model:** Overlap represents physical deformation, with restoring forces controlling motion.
- **Computation of Post-Collision Velocities:**
  - **Hard-Sphere Model:** Employs impulse calculations to determine instantaneous velocity changes.
  - **Soft-Sphere Model:** Uses the leapfrog algorithm to iteratively update velocities based on continuous force interactions.
- **Computational Complexity:**
  - **Hard-Sphere Model:** Efficient for large-scale simulations due to simplified collision handling.
  - **Soft-Sphere Model:** More computationally expensive as it requires numerical integration over time.

Summary of Similarities and Differences in Two-Particle Collisions

The hard-sphere and soft-sphere models differ in their treatment of particle collisions. In both approaches, colliding particles obey the conservation of linear momentum and energy principles, with energy dissipation dependent on the restitution coefficient. However, the hard-sphere model assumes instantaneous collisions with impulse-based velocity updates, making it computationally efficient. In contrast, the soft-sphere model accounts for finite collision duration, employing the leapfrog method to iteratively compute velocity changes based on continuous force interactions. This results in more accurate modeling of deformation effects but at a higher computational cost.

3.7. Results for the Two Simulated Particles:

- **Collision Occurrence Time:**
  - Time of Collision: 0.50 seconds
- **Initial Contact Point for Both Models:**
  - Initial Contact Coordinates: `[0.300, 0.400, 0.000]`
- **Elastic Post-Collision Velocities (e = 1.0):**
  - Hard-Sphere Model:
    Particle 1: `[-0.840, -1.120, 0.000]`
    Particle 2: `[-0.160, 0.120, 0.000]`
  - Soft-Sphere Model:
    Particle 1: `[-0.838, -1.121, 0.000]`
    Particle 2: `[-0.162, 0.121, 0.000]`
- **Inelastic Post-Collision Velocities (e = 0.75):**
  - Hard-Sphere Model:
    Particle 1: `[-0.735, -0.980, 0.000]`
    Particle 2: `[-0.265, -0.020, 0.000]`
  - Soft-Sphere Model:
    Particle 1: `[-0.733, -0.981, 0.000]`
    Particle 2: `[-0.267, -0.019, 0.000]`
- **Collision Duration:**
  - Hard-Sphere Model:
    Duration: 0.0000 seconds (Instantaneous Collision)
  - Soft-Sphere Model:
    Duration: 0.0210 seconds
- **Final Contact Point and Deformation Effects:**
  - Hard-Sphere Model: `[0.300, 0.400, 0.000]`
  - Soft-Sphere Model: `[0.289, 0.389, 0.000]`

4. Conclusion

The analysis of the hard-sphere and soft-sphere models highlights their trade-offs between computational efficiency and physical accuracy in simulating particle collisions:

- The hard-sphere model offers significant computational advantages due to its simplified assumptions, making it suitable for large-scale simulations where detailed collision dynamics are less critical. The computational time for the hard-sphere model is very low compared to the soft-sphere model.
- The soft-sphere model provides a more realistic representation of particle interactions by considering deformation and force dynamics, leading to higher accuracy. However, this increased realism comes at a higher computational cost.
- The choice of model should be driven by the specific objectives of the simulation, carefully balancing the need for accuracy with available computational resources. When computational speed is paramount, the hard-sphere model is clearly advantageous.

For high-fidelity simulations requiring accurate representation of deformation and contact mechanics, the soft-sphere model is preferred, despite its higher computational demands. Conversely, for large-scale simulations where computational speed is paramount, the hard-sphere model remains a valuable and efficient tool. The order of magnitude difference in computational time observed in our simulations underscores this trade-off.