Distinct/Discrete Element Method

Clay Wood, Daulet Sagzhanov, Xuanchi Li
Outline

- **Introduction**
  - What is DEM?
  - Brief Historical Background
  - Advantages/Disadvantages
  - Applications Overview

- **Governing Equations**
  - Newtonian Mechanics
  - Surface/Material Interactions
  - General DEM Workflow

- **DEM Simulations**
  - 1D Hand Calculation
  - 2D Example

- **Applications**
  - Simple Shear
  - Granular Avalanche
  - Mixing Concrete
  - Grains Falling in Hopper
Introduction
Introduction: What is DEM?

**Distinct / Discrete Element Method (DEM)**

- a way of simulating discrete matter
- a numerical model capable of describing the mechanical behaviour of assemblies of discs and spheres
- a particle-scale numerical method for modeling the bulk behavior of granular materials and many geomaterials (coal, ores, soil, rocks, aggregates)
- capture dual nature of materials
CONTINUUM

- Continuous matter
- Occupies entire space
- Continuum Mechanics
- FEM

DISCRETE

- Dis-continuous matter
- Each particle is a unique quantity
- Material = assembly of particles
- DEM
Characteristic example

CONTINUUM

DISCRETE
Historical Background

Molecular Dynamics (MD)

1956
Alder and Wainwright

Discrete Element Method (DEM)

1979
Cundall and Strack
Advantages and Disadvantages of DEM

**Advantages:**
- Modeling Movement of Individual Particles
- Full stress and strain tensors can be measured
- Time Steps
- Progressive Failure

**Disadvantages:**
- Complex Particle Geometries and Arrangements
- Roughness, Texture
- Grain Crushing, Particle Breakage
- Non-Idealized Contacts
DEM Applications

- Civil Engineering (Geotechnical Engineering)
- Chemical Engineering
- Oil and gas production
- Geomechanics
- Mineral processing
- Biochemical Engineering
- Powder metallurgy
- Agricultural Industry
Governing Equations: Newtonian Mechanics

\[ \mathbf{F}_{\text{trans}} = m \mathbf{u} \]

\[ \sum_{i = 1}^{n_{\text{part}}} \mathbf{F}_{\text{trans}, i} + \mathbf{F}_{\text{rot}, i} = \mathbf{F}_{\text{tot}} \]

\[ \mathbf{F}_{\text{rot}} = \mathbf{T} = I \omega \]
Governing Equations: Other Interactions

\[ F_{\text{fric}} = \mu F_{\text{normal}} \]

\[ F_{\text{spring}} = k \Delta x \]

Generalised Hooke’s Law

\[
\begin{align*}
\epsilon_x &= \frac{1}{E} \left[ \sigma_x - \nu(\sigma_y + \sigma_z) \right] \\
\epsilon_y &= \frac{1}{E} \left[ \sigma_y - \nu(\sigma_x + \sigma_z) \right] \\
\epsilon_z &= \frac{1}{E} \left[ \sigma_z - \nu(\sigma_x + \sigma_y) \right]
\end{align*}
\]

Shear stress-strain relations

\[
\begin{align*}
\gamma_{xy} &= \frac{1}{G} \tau_{xy} \\
\gamma_{yz} &= \frac{1}{G} \tau_{yz} \\
\gamma_{xz} &= \frac{1}{G} \tau_{xz}
\end{align*}
\]
Governing Equations: Conservation of Momentum

\[ F = m \, u \]

\[ \sum F = 0 \]

\[ F = k \, u \]

\[ m \, u \, t + k \, u \, t = 0 \quad \Rightarrow \quad u \, t = -\frac{k \, u \, t}{m} \]
Governing Equations (cont-d):

- Numerical Integration:

  Update particle velocities and positions every time step.

  \[
  x(t + \Delta t) = x(t) + v(t)\Delta t \\
  v(t + \Delta t) = v(t) + a(t)\Delta t
  \]
Model Workflow

- Newton’s Second Law of Motion
- Force Displacement Law

Force Displacement Laws (e.g. stiffness, friction)

Force Boundary Condition

Newton’s Second Law of Motion

Displacement/Velocity Boundary Conditions
1D DEM
Hand Calculation Example (1-D DEM Example)

- 1-D Bouncing Ball Matlab Simulation
- Bouncing ball released from a height of 8m
- With air resistance particles are not assumed to be elastic
- Governing Equation

\[
F = \begin{cases} 
mg & \text{for } x \geq 0 \\
mg - kx & \text{for } x < 0 
\end{cases}
\]
Hand Calculation Example Continued, Matlab Code

clear, format compact

\[ \text{height}=8; \quad % \text{Height in meters} \]
\[ \text{v}_t=10; \quad % \text{Terminal velocity in meters per second} \]
\[ g=9.8; \quad % \text{Gravitational Acceleration} \]
\[ C_R=0.9; \quad % \text{Coefficient of restitution} \]
\[ h(1)=% \text{Vertical height to release the ball from} \]
\[ b=1; \quad % \text{Initialize bounce number} \]

\textbf{for} \text{ } b=1:8 \quad % \text{Loop through three bounces}

\[ \text{v}_\text{impact}(b)=\text{v}_t\sqrt{1-\exp(-2g h(b)/(v_t^2))}; \]
\[ \text{v}_r(b)=C_R \cdot \text{v}_\text{impact}(b) \cdot (1-0.01\cdot \text{rand}()); \]
\[ h(b+1)=-((v_t^2)/g)\cdot \ln(\cos(\arctan(v_r(b)/v_t))); \]
\textbf{end}

\text{sprintf}('The height of the third bounce is \%0.3f meters.', h(4))

\[ C_R = \frac{\text{v}_r}{\text{v}_i} \]
\[ \text{v}_i = \text{v}_t \cdot \sqrt{1 - e^{\frac{-2gh}{v_t^2}}} \]

\[ h_{\text{rebound}} = -\frac{\text{v}_t^2}{g} \cdot \ln(\cos(\tan^{-1}\frac{\text{v}_r}{\text{v}_t})) \]

\textbf{close all}
\textbf{plot}([0 length(h)],zeros(1,2),',k',',LineWidth',2) \% \text{plot the floor}
\textbf{hold on}
\textbf{ylim}([-0.05*height 1.2*height]); \% \text{set the vertical limits}
\% \text{Plot the first drop as a half parabola}
\text{traj}=@(x) h(1).*((x+0.5).*((x-0.5))./((0+0.5).*((0-0.5))));
\textbf{plot}(0:0.05:0.5,\text{traj}(0:0.05:0.5),',ro',',MarkerSize',15)
\% \text{Plot each bounce as a full parabola}
\textbf{for} \text{ } b=1:length(h)-1

\[ \text{traj}=@(x) h(b+1).*((x-(b-0.5)).*((x-(b+0.5))./((b-(b-0.5)).*(b-(b+0.5)))); \]
\[ \textbf{plot}((b-0.5):0.05:(b+0.5),\text{traj}(b-0.5):0.05:(b+0.5),',ro',',MarkerSize',15) \]
\textbf{end}
\textbf{title}('Solution of a Bouncing Ball');
\textbf{xlabel}('Time t');
\textbf{ylabel}('Vertical Position');
\textbf{legend}('Hand Calculation');
\[ v_{\text{impact}} = [9.2690, 6.5921, 5.2813, 4.4716, 3.8854, 3.4507, 3.0959, 2.8033, 2.5628, 2.3651] \]

\[ v_r = [8.7666, 6.2194, 4.9992, 4.2167, 3.6766, 3.2558, 2.9204, 2.6513, 2.4342, 2.2394] \]
2D DEM
2D DEM Example

Evolution of system of 25 particles

1. Particles have initial velocity tensor.
2. Particles fall, $v_{\text{initial}}$ dominated by $g$.
3. Bouncing dictated by $E$ (Young’s modulus) between particles and sides of box.
2D DEM Example

Evolution of system of 25 particles

1. Particles have initial velocity tensor.
2. Particles fall, $v_{\text{initial}}$ dominated by $g$.
3. Bouncing dictated by $E$ (Young’s modulus) between particles and sides of box.
% physical parameters → global definitions
n_part=25;
% number of particles
% initialize radius, mass, & gravity
global rad, rad(1:n_part)=0.5;
global m, m(1:n_part)=1;
global g, g=-9.81;
% Young’s modulus
global E, E=10000;
% size of “bounding box” → global definitions
global lmaxx, lmaxx=n_part/2+1;
global lminx, lminx=0;
global lmaxy, lmaxy=n_part/2+1;
global lminy, lminy=0;

% initialize positions and velocities
% random number generator
rng('shuffle','combRecursive');
% create/sort particle centers: x,y
r0_x=2*mod([1:n_part],5)+1;
r0_y=sort(r0_x);
% give each particle initial random velocity
v0_x=rand(size(r0_x))-0.5;
v0_y=rand(size(r0_y))-0.5;
% array of spatial vector components
y0(1:4:4*n_part-3)=r0_x;
y0(2:4:4*n_part-2)=v0_x;
y0(3:4:4*n_part-1)=r0_y;
y0(4:4:4*n_part)=v0_y;

% initialize positions and velocities
% set timescale for simulation (arb units)
t_end=5;
% create vector of time and particle position.
Use ode13 to solve function ‘dem2D’.
See ‘dem2D’ for details of particle physics
[t,y]=ode13('dem2D',[0:0.05:t_end],y0);
Define Physical Parameters:

- # particles
- Mass
- Radius
- Gravity
- Elasticity

% size of "bounding box" → global definition
\[
\text{global } I_{\text{maxx}}, I_{\text{maxy}} = \text{n\_part}/2+1;
\]
\[
\text{global } I_{\text{minx}}, I_{\text{miny}} = 0;
\]
\[
\text{global } I_{\text{maxxy}}, I_{\text{maxy}y} = \text{n\_part}/2+1;
\]
\[
\text{global } I_{\text{minxy}}, I_{\text{miny}y} = 0;
\]

% initialize positions and velocities
% random number generator
rng('shuffle','combRecursive');
% create/sort particle centers: x,y
r0_x = 2*mod([1:n_part],5)+1;
r0_y = sort(r0_x);
% give each particle initial random velocity
v0_x = rand(size(r0_x))-0.5;
v0_y = rand(size(r0_y))-0.5;
% array of spatial vector components
y0(1:4:4*n_part-3) = r0_x;
y0(2:4:4*n_part-2) = v0_x;
y0(3:4:4*n_part-1) = r0_y;
y0(4:4:4*n_part) = v0_y;

% initialize positions and velocities
% set timescale for simulation (arb units)
t_end = 5;
% create vector of time and particle position.
Use ode113 to solve function ‘dem2D’.
See ‘dem2D’ for details of particle physics
\[
[t,y] = \text{ode113}('\text{dem2D}', [0:0.05:t\_end], y0);
\]
Define Physical Parameters:

- # particles
- Mass
- Radius
- Gravity
- Elasticity

Define System Size:

\((X,Y): 0 \rightarrow \text{#part/2 +1}\)

...% initialize positions and velocities% random number generatorrng('shuffle','combRecursive');% create/sort particle centers: x,yr0_x=2*mod([1:n_part],5)+1;r0_y=sort(r0_x);% give each particle initial random velocityv0_x=rand(size(r0_x))-0.5;v0_y=rand(size(r0_y))-0.5;% array of spatial vector componentsy0(1:4:4*n_part-3)=r0_x;y0(2:4:4*n_part-2)=v0_x;y0(3:4:4*n_part-1)=r0_y;y0(4:4:4*n_part)=v0_y;

% initialize positions and velocities% set timescale for simulation (arb units)t_end=5;% create vector of time and particle position.Use ode13 to solve function ‘dem2D’.See ‘dem2D’ for details of particle physics[t,y]=ode13('dem2D',[0:0.05:t_end],y0);
Define Physical Parameters:

- # particles
- Mass
- Radius
- Gravity
- Elasticity

Set Particle Position/Velocity:

create grid of particle centers, \( r0 \)
randomize velocities, \( v0 \)
\[ y0 = [r0_{x1}, v0_{x1}, r0_{y1}, v0_{y1}, ...] \]

Define System Size:

\((X,Y): 0 \rightarrow \text{#part}/2 + 1\)

% initialize positions and velocities
% set timescale for simulation (arb units)
t_end=5;
% create vector of time and particle position.
Use ode113 to solve function ‘dem2D’.
See ‘dem2D’ for details of particle physics

\[ [t,y]=\text{ode113}('dem2D',[0:0.05:t_end],y0); \]
Define Physical Parameters:
- # particles
- Mass
- Radius
- Gravity
- Elasticity

Define System Size:
(X,Y): 0 → #part/2 +1

Set Particle Position/Velocity:
create grid of particle centers, \( r_0 \)
randomize velocities, \( v_0 \)
\( y_0 = [r_{0x_1} v_{0x_1} r_{0y_1} v_{0y_1} \ldots] \)

Solve for \( u, \frac{du}{dt}, \frac{du^2}{dt^2} \) for each \( \Delta t \):
Define physical interactions as \( \text{physics_func} \)
solveODE(\( \text{physics_func} \), (\( t_0 \):\( \Delta t \):\( t_{\text{end}} \)), \( y_0 \))
→ \([t \ y]\)
\( y = [x_i \ x_i \ y_i \ y_i \ldots] \)
function [dydt]=dem2D(t,y);
    global m rad E lmaxx lmaxy lminx lminy lmax lmin g n_part
    a=zeros(2,n_part);
    for i_part=1:n_part
        r1=[y(4*i_part-3)
            y(4*i_part-1)]; % position of first particle
        rad1=rad(i_part);
    % Particle-Particle Interaction
        for j_part=i_part+1:n_part
            r2=[y(4*j_part-3)
                y(4*j_part-1)]; % position of second particle
            rad2=rad(j_part);
            if (norm(r1-r2)<(rad1+rad2))
                forcemagnitude=E*abs(norm(r1-r2)-(rad1+rad2));
                forcedirection=(r1-r2)/norm(r1-r2);
                f=forcemagnitude*forcedirection;
                a(:,i_part)=a(:,i_part)+f;
                a(:,j_part)=a(:,j_part)-f;
            end
        end
    end
    a(2,:)=a(2,:)+g;
    dydt=zeros(4*n_part,1);
    dydt(1:4:4*n_part-3)=y(2:4:4*n_part-2);
    dydt(2:4:4*n_part-2)=a(1,:)./m;
    dydt(3:4:4*n_part-1)=y(4:4:4*n_part);
    dydt(4:4:4*n_part)=a(2,:)./m;
    return

% Particle-wall Interaction
    if (r1(1)-rad1)<lminx
        a(1,i_part)=a(1,i_part)-E*((r1(1)-rad1)-lminx);
    end
    if (r1(1)+rad1)>lmaxx
        a(1,i_part)=a(1,i_part)-E*((r1(1)+rad1)-lmaxx);
    end
    if (r1(2)-rad1)<lminy
        a(2,i_part)=a(2,i_part)-E*((r1(2)-rad1)-lminy);
    end
    if (r1(2)+rad1)>lmaxy
        a(2,i_part)=a(2,i_part)-E*((r1(2)+rad1)-lmaxy);
    end
    a(2,:)=a(2,:)+g;
    dydt=zeros(4*n_part,1);
    dydt(1:4:4*n_part-3)=y(2:4:4*n_part-2);
    dydt(2:4:4*n_part-2)=a(1,:)./m;
    dydt(3:4:4*n_part-1)=y(4:4:4*n_part);
    dydt(4:4:4*n_part)=a(2,:)./m;
    return
2D DEM: “Particle Physics Engine”

Pull in global variables.
Create **accel** vector: [x-comp y-comp; 1 : #part]

populate list of particle radii = \( r_1 \)

populate list of adjacent particles radii = \( r_2 \)

If \( r_1 - r_2 < \) particle radius
- \( F_{mag} = \) Young’s Mod * amount of particle overlap
- \( F_{dir} = \) particle overlap / norm(particle overlap)
- \( F = F_{mag} * F_{dir} \)
- Populate **accel** vector

% Particle-wall Interaction
if \((r1(1)\text{-}rad1)<lminx\)
\[
a(1,i\_part)=a(1,i\_part)\cdot E*\((r1(1)\text{-}rad1)\text{-}lminx)\numix);
end
\]
if \((r1(1)\text{+}rad1)>lmaxx\)
\[
a(1,i\_part)=a(1,i\_part)\cdot E*\((r1(1)\text{+}rad1)\text{-}lmaxx)\numaxx);
end
\]
if \((r1(2)\text{-}rad1)<lminy\)
\[
a(2,i\_part)=a(2,i\_part)\cdot E*\((r1(2)\text{-}rad1)\text{-}lminy)\numix);
end
\]
if \((r1(2)\text{+}rad1)>lmaxy\)
\[
a(2,i\_part)=a(2,i\_part)\cdot E*\((r1(2)\text{+}rad1)\text{-}lmaxy)\numaxx);
end
\]
end
end
a(2,:)\text{=}a(2,:)+g;
dydt=\text{zeros}(4*n\_part,1);
dydt(1:4\cdot n\_part-3)\text{=}y(2:4\cdot n\_part-2);
dydt(2:4\cdot n\_part-2)\text{=}a(1,:)\text{/}m;
dydt(3:4\cdot n\_part-1)\text{=}y(4:4\cdot n\_part);
dydt(4:4\cdot n\_part)\text{=}a(2,:)\text{/}m;
return
2D DEM: “Particle Physics Engine”

Pull in global variables.
Create **accel** vector: [x-comp y-comp; 1 : #part]

populate list of particle radii = $r_1$

populate list of adjacent particles radii = $r_2$

If $r_1 - r_2 < $ particle radius
- $F_{mag} = $ Young’s Mod * amount of particle overlap
- $F_{dir} = $ particle overlap / norm(particle overlap)
- $F = F_{mag} * F_{dir}$
- Populate **accel** vector

Particle-wall interaction:

If particle center – particle radius < wall coordinate
- $F_{mag} = $ Young’s Mod * amount of particle overlap
- $F_{dir} = $ particle overlap / norm(particle overlap)
- $F = F_{mag} * F_{dir}$
- Repopulate **accel** vector

Add g to all **accel**, & / particle mass

Populate dydt vector = $\left[ x_i \dot{x}_i y_i \dot{y}_i \ldots \right]$
2D DEM: High E
2D DEM: Low E
2D DEM Example: Limitations

Made the following assumptions/simplifications:

- No dissipative forces
  - Friction: (Amontons’ Law or Hertzian Contact Theory)
  - Ambient fluid resistance (air/liquid)
- No particle rotation
  - Would need to calculate torque, moment of inertia...
Applications: Real Systems
Applications: Shearing Jammed Granular System
Applications: Granular Avalanche
Applications: Granular Force Networks
Applications: Concrete Mixing

Simulation – Concrete mixing
Applications: Grains Falling into Hopper
Applications: Real Systems

OBSS 3
Vessel: Brave Wind
Coal Type: Walkworth Steam
Date: 09/06/09

Helix DEM Chute Design - www.helixtech.com.au
Thank you
Citations

Slides 5-7&13: Diagrams from EDM™ Webinar

1D DEM: adapted from MATLAB “Bouncing Ball” Example

2D DEM: adapted from “Understanding the Discrete Element Method”, Matuttis, H., Chen, J.

Slide 34: https://www.youtube.com/watch?v=ruFsRGAw2Rw


Slide 37: SimulationLABWeimar, https://www.youtube.com/watch?v=2szJ38qcZro

Slide 38: https://www.youtube.com/watch?v=3EbE45qGG6s

Extras...
Governing Equations:

DEM uses two types of governing laws:

- Newton’s Second Law of Motion
  \[ F = MA \]

- Force-Displacement Law
  Hooke’s law, friction etc…

- Time Step