Distinct/Discrete Element Method

Clay Wood, Daulet Sagzhanov, Xuanchi Li

Outline

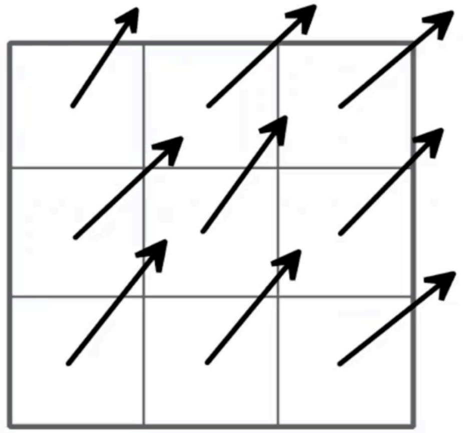
Introduction

Introduction: What is DEM?

Distinct / Discrete Element Method (DEM)

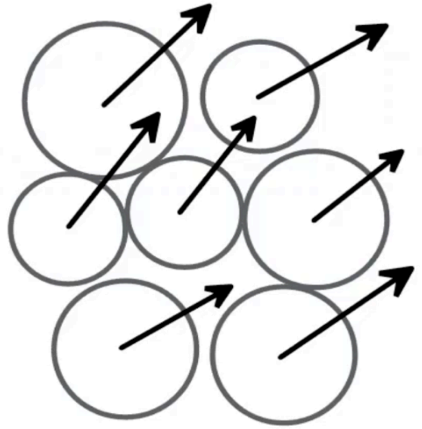
- a way of simulating discrete matter
- a numerical model capable of describing the mechanical behaviour of assemblies of discs and spheres
- a particle-scale numerical method for modeling the bulk behavior of
- granular materials and many geomaterials (coal, ores, soil, rocks, aggregates)
- capture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

CONTINUUM



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

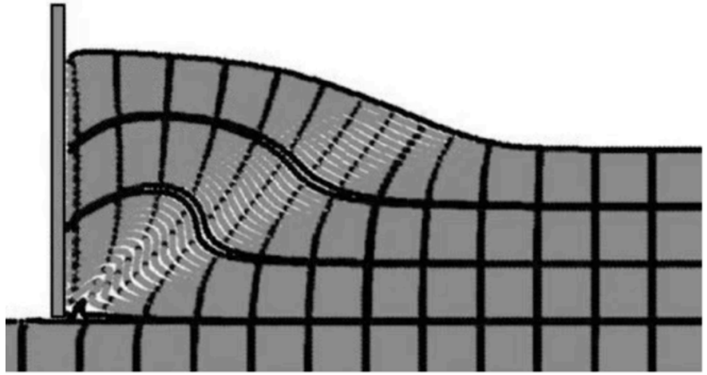
DISCRETE



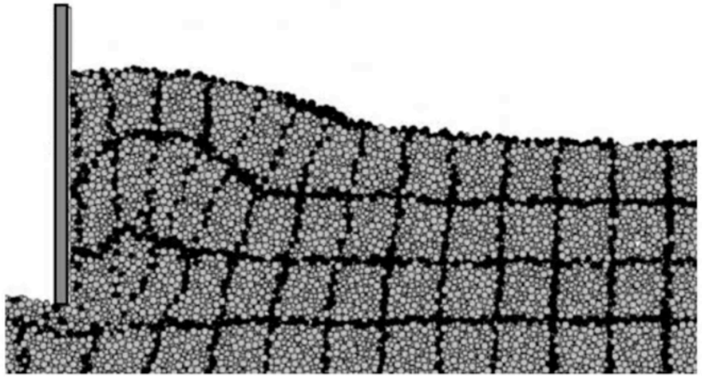
- Dis-continuous matter
- Each particle is a unique quantity
- Material = assembly of particles
- DEM

Characteristic example

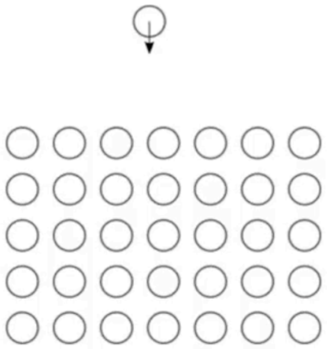
CONTINUUM



DISCRETE

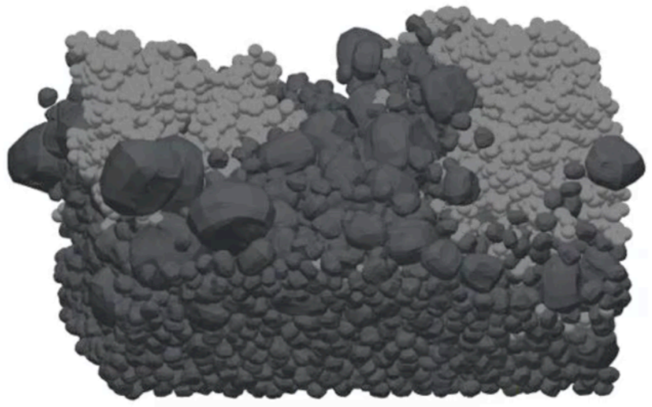


Historical Background



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:

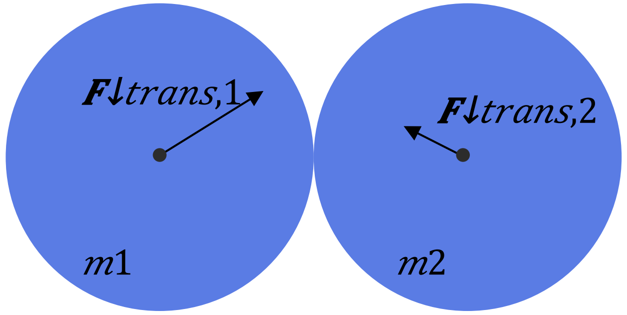
- Complex Particle Geometries and Arrangements
- Roughness, Texture
- Grain Crushing, Particle Breakage
- Non-Idealized Contacts

DEM Applications

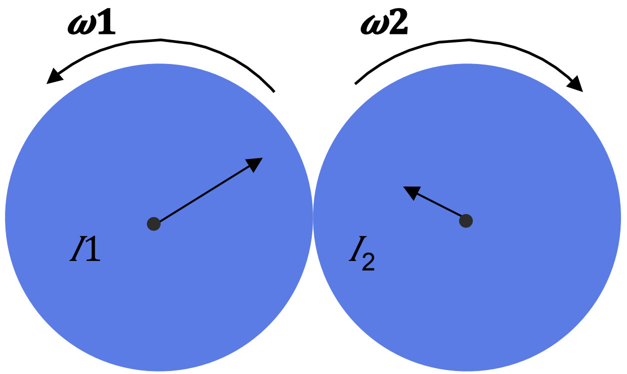
- Civil Engineering (Geotechnical Engineering)
- Chemical Engineering
- Oil and gas production
- Geomechanics
- Mineral processing
- Biochemical Engineering
- Powder metallurgy
- Agricultural Industry

Governing Equations: Newtonian Mechanics

$$\boldsymbol{F}_{trans} = m\,\boldsymbol{u}$$



$$\boldsymbol{F}_{rot} = \boldsymbol{T} = I\,\boldsymbol{\omega}$$

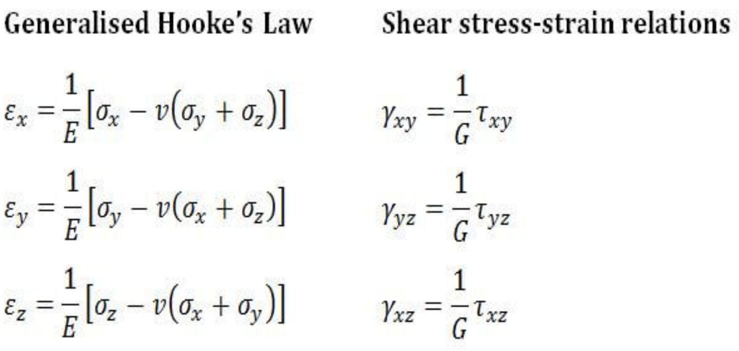


$$\boldsymbol{F}_{tot} = \sum_{i}^{n\_part} \boldsymbol{F}_{trans,\ i} + \boldsymbol{F}_{rot,\ i}$$

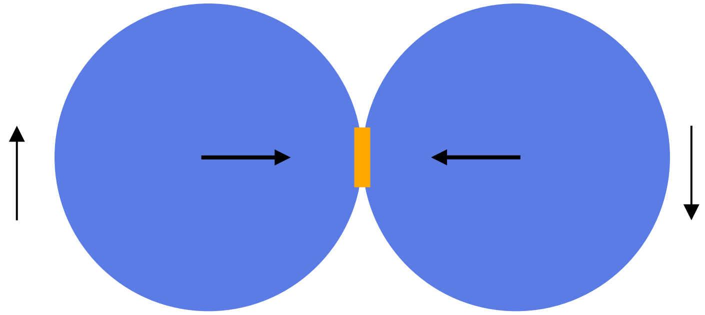
Governing Equations: Other Interactions

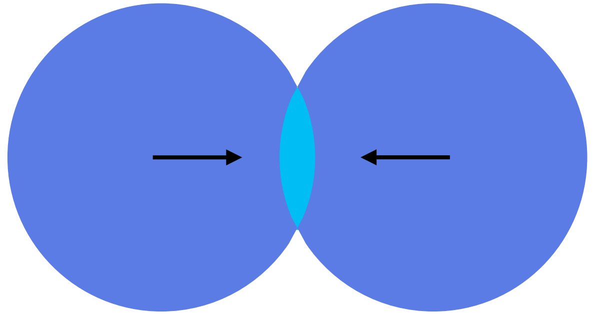
$F\downarrow fric = \mu\, F\downarrow normal$

$F\downarrow spring = k\,\Delta x$



Generalised Hooke's Law	Shear stress-strain relations
$\varepsilon_x = \frac{1}{E}\left[\sigma_x - v\left(\sigma_y + \sigma_z\right)\right]$	$\gamma_{xy} = \frac{1}{G}\tau_{xy}$
$\varepsilon_y = \frac{1}{E}\left[\sigma_y - v\left(\sigma_x + \sigma_z\right)\right]$	$\gamma_{yz} = \frac{1}{G}\tau_{yz}$
$\varepsilon_z = \frac{1}{E}\left[\sigma_z - v\left(\sigma_x + \sigma_y\right)\right]$	$\gamma_{xz} = \frac{1}{G}\tau_{xz}$





Governing Equations: Conservation of Momentum

$\boldsymbol{F} = m\,\boldsymbol{u}$

$\sum$↑▒$\boldsymbol{F} = \mathbf{0}$

$m\,\boldsymbol{u}\,t + k\,\boldsymbol{u}t = 0 \quad \Longrightarrow \quad \boldsymbol{u}\,t = -k\,\boldsymbol{u}t/m$

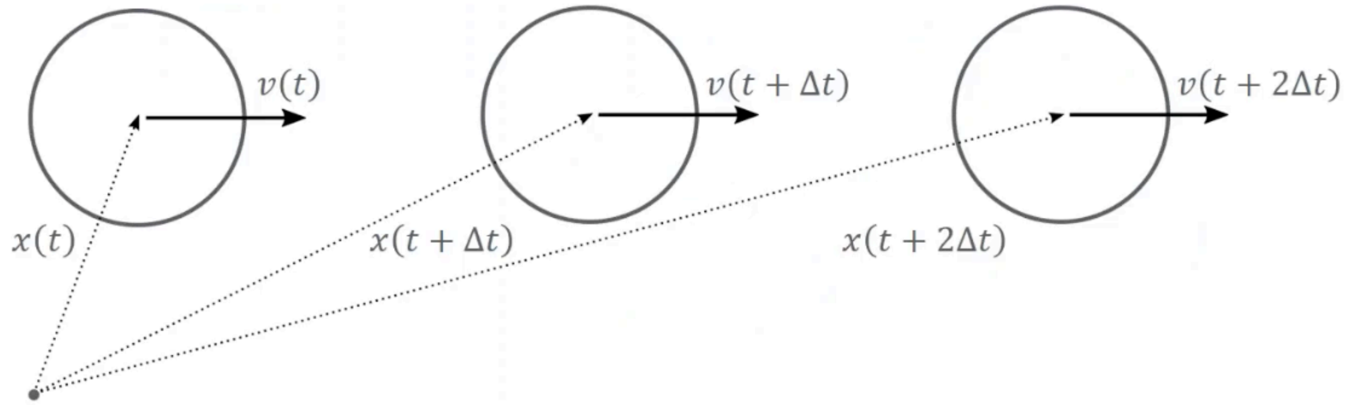
$\boldsymbol{F} = k\,\boldsymbol{u}$

Governing Equations (cont-d):

- Numerical Integration:

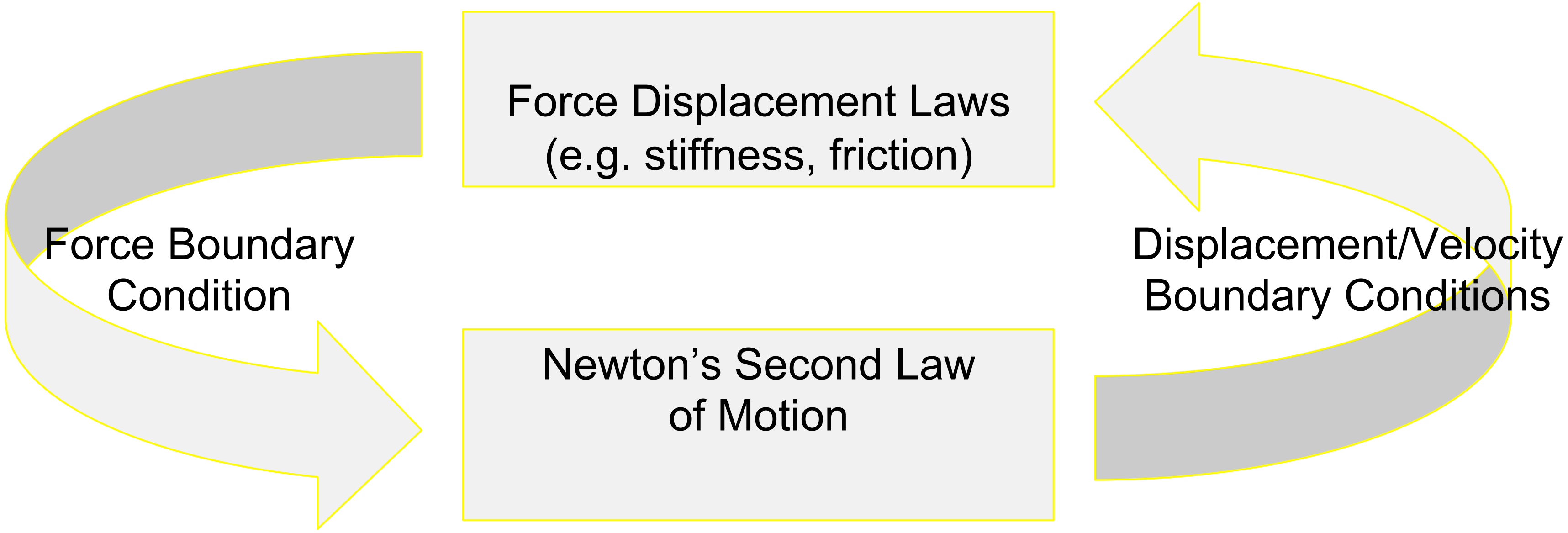
Update particle velocities and positions every time step.

$$\boldsymbol{x}(\boldsymbol{t} + \Delta \boldsymbol{t}) = \boldsymbol{x}(\boldsymbol{t}) + \boldsymbol{v}(\boldsymbol{t})\Delta \boldsymbol{t}$$
$$\boldsymbol{v}(\boldsymbol{t} + \Delta \boldsymbol{t}) = \boldsymbol{v}(\boldsymbol{t}) + \boldsymbol{a}(\boldsymbol{t})\Delta \boldsymbol{t}$$



Model Workflow

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



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

$$F = \begin{cases} mg & for\ x \geq 0 \\ mg - kx & for\ x < 0 \end{cases}$$

Hand Calculation Example Continued, Matlab Code

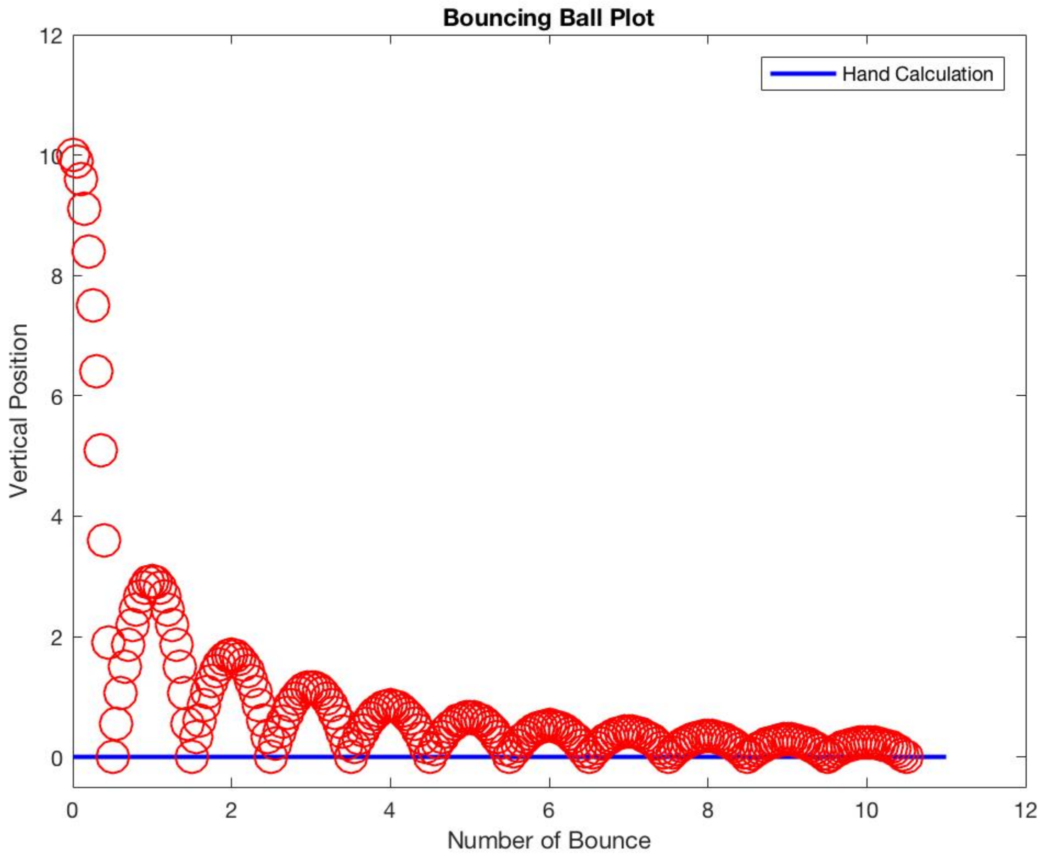
```
clear, format compact
height=8;                % Height in meters
v_t=10;                % Terminal velocity in meters per second
g=9.8;               % Gravitational Acceleration
C_R=0.9;             % Coefficient of restitution
h(1)=height;         % Vertical height to release the ball from
b=1;               % Initialize bounce number
for b=1:8            % Loop through three bounces
   v_impact(b)=v_t*sqrt(1-exp(-2*g*h(b)/(v_t^2)));
   v_r(b)=C_R*v_impact(b)*(1-0.01*rand());
   h(b+1)=-((v_t^2)/g)*log(cos(atan(v_r(b)/v_t)));
end
sprintf('The height of the third bounce is %0.3f meters.', h(4))
```

$$C_R = \frac{v_r}{v_i} \qquad v_i = v_t * \sqrt{1 - e^{\left(\frac{-2gh}{v_t^2}\right)}}$$

$$h_{rebound} = -\frac{v_t^2}{g} * ln(\cos(\tan^{-1}\frac{v_r}{v_t}))$$

```
close all
plot([0 length(h)],zeros(1,2),'k','LineWidth',2) % plot the
floor
hold on
ylim([-0.05*height 1.2*height]); % set the vertical limits
% Plot the first drop as a half parabola
traj=@(x) h(1).*(x+0.5).*(x-0.5)./((0+0.5).*(0-0.5));
plot(0:0.05:0.5,traj(0:0.05:0.5),'ro','MarkerSize',15)
% Plot each bounce as a full parabola
for b=1:length(h)-1
   traj=@(x) h(b+1).*(x-(b-0.5)).*(x-(b+0.5))./((b-
(b-0.5)).*(b-(b+0.5)));
   plot((b-0.5):0.05:(b+0.5),traj((b-0.5):0.05:
(b+0.5)),'ro','MarkerSize',15)
end
title('Solution of a Bouncing Ball');
xlabel('Time t');
ylabel('Vertical Position');
legend('Hand Calculation');
```

```
v_impact =[9.2690        vr =[8.7666
          6.5921              6.2194
          5.2813              4.9992
          4.4716              4.2167
          3.8854              3.6766
          3.4507              3.2558
          3.0959              2.9204
          2.8033              2.6513
          2.5628              2.4342
          2.3651]             2.2394]
```

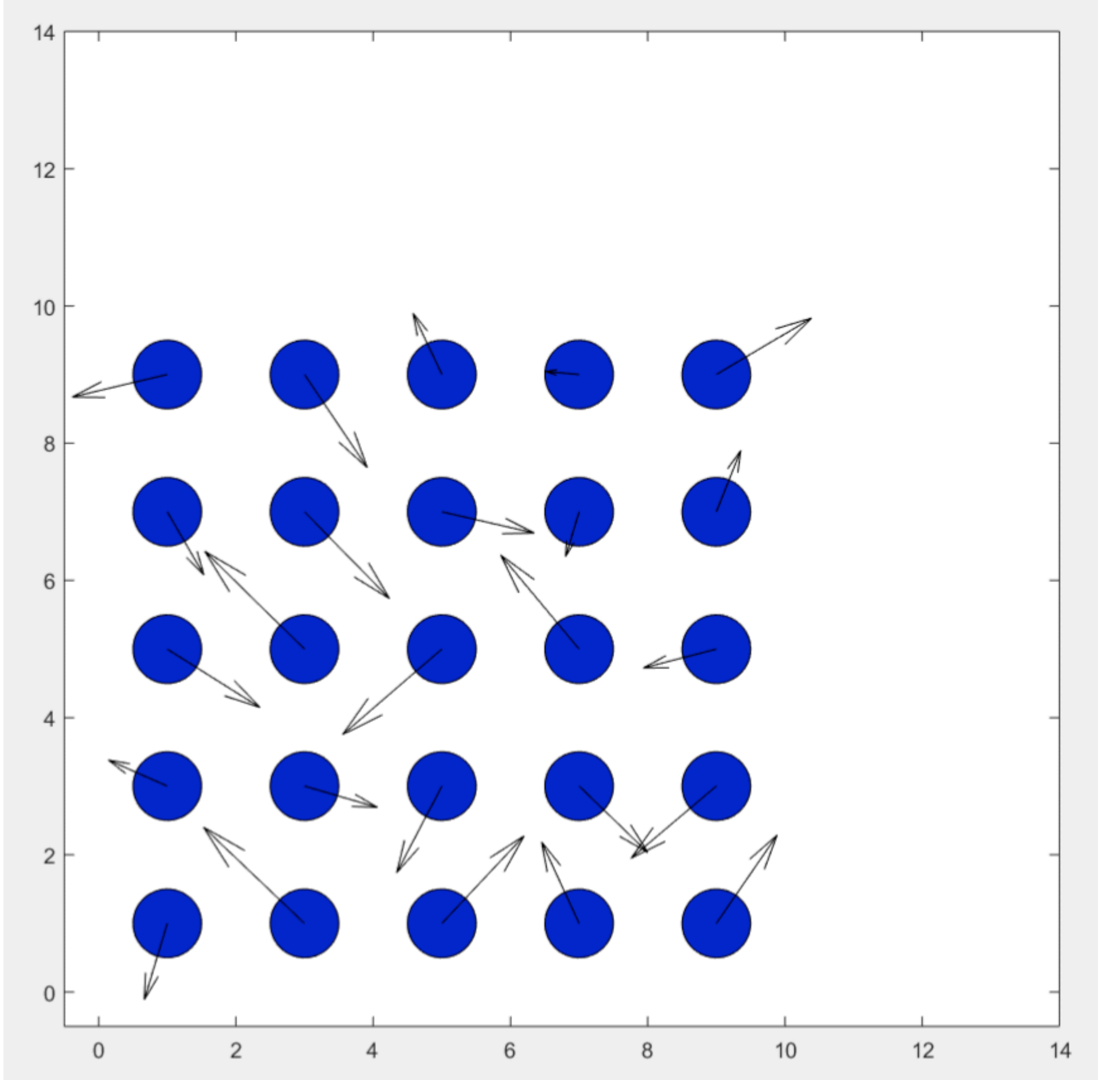


2D DEM

2D DEM Example

Evolution of system of 25 particles

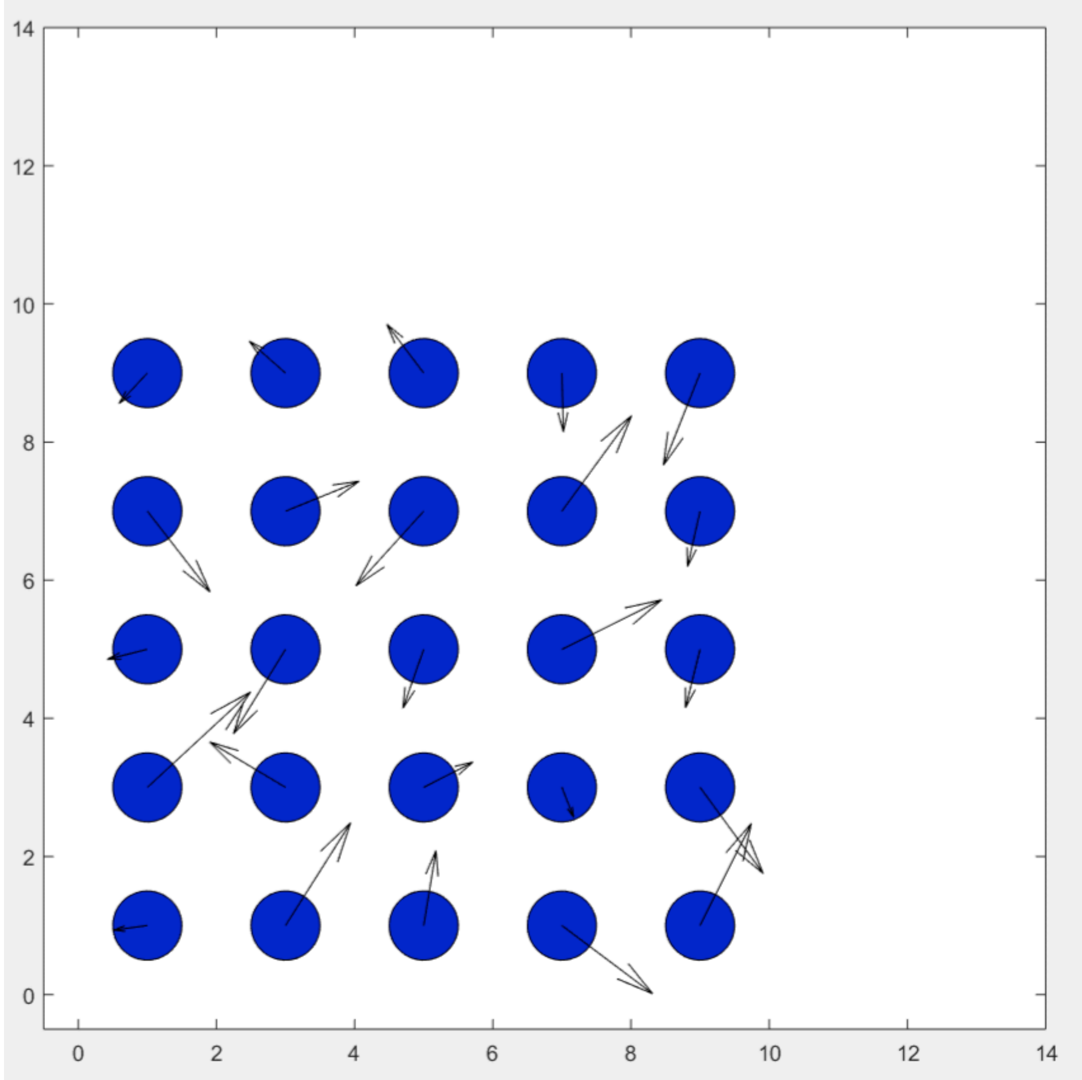
1. Particles have initial velocity tensor.
2. Particles fall, $\mathbf{v}_{\text{initial}}$ dominated by $\mathbf{g}$.
3. Bouncing dictated by E (Young's modulus) between particles and sides of box.



2D DEM Example

Evolution of system of 25 particles

1. Particles have initial velocity tensor.
2. Particles fall, $\mathbf{v}_{\text{initial}}$ dominated by $\mathbf{g}$.
3. Bouncing dictated by E (Young's modulus) between particles and sides of box.



2D DEM: “Spatial Setup and Solver”

```
% physical parameters → global definitions
%number of particles
n_part=25;
% 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 definition
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 ode113 to solve function ‘dem2D’.
See ‘dem2D’ for details of particle physics

[t,y]=ode113('dem2D',[0:0.05:t_end],y0);
```

2D DEM: “Spatial Setup and Solver”

Define Physical Parameters:

- # particles
- Mass
- Radius
- Gravity
- Elasticity

```
% size of “bounding box”→ global definition
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 ode113 to solve function ‘dem2D’.
See ‘dem2D’ for details of particle physics

[t,y]=ode113('dem2D',[0:0.05:t_end],y0);
```

2D DEM: “Spatial Setup and Solver”

Define Physical Parameters:

- # particles
- Mass
- Radius
- Gravity
- Elasticity

Define System Size:

(X,Y): 0 → #part/2 +1

```
...
% 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]=ode113('dem2D',[0:0.05:t_end],y0);
```

2D DEM: “Spatial Setup and Solver”

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, ***r0***

randomize velocities, ***v0***

y0 = [$r0_{xi}$ $v0_{xi}$ $r0_{yi}$ $v0_{yi}$ …]

```
% 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]=ode113('dem2D',[0:0.05:t_end],y0);
```

2D DEM: “Spatial Setup and Solver”

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, ***r0***

randomize velocities, ***v0***

y0 = [$r0_{xi}$ $v0_{xi}$ $r0_{yi}$ $v0_{yi}$ …]

Solve for u, du/dt, du^2/dt^2 for each Δt:

Define physical interactions as ***physics_func***

solveODE(***physics_func***, (t_0:Δt:t_{end}), ***y0***)
→ [***t y***]

y = [***x i xi yi yi*** ...]

2D DEM: “Particle Physics Engine”

```matlab
function [dydt]=dem2D(t,y);
    global m rad E lmax lmin lmaxx lminx lmaxy lminy 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)<(rad(i_part)+rad(j_part)))
                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
        % 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
    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 = ***r1***

populate list of adjacent particles radii = ***r2***

If **r1 - r2** < particle radius

- $\mathbf{F}_{\mathbf{mag}}$ = Young’s Mod * amount of particle overlap
- $\mathbf{F}_{\mathbf{dir}}$ = particle overlap / norm(particle overlap)
- $\mathbf{F} = \mathbf{F}_{\mathbf{mag}} * \mathbf{F}_{\mathbf{dir}}$
- Populate **accel** vector

```
    % 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
    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 = ***r1***

populate list of adjacent particles radii = ***r2***

If **r1 - r2** < particle radius

- $\mathbf{F_{mag}}$ = Young’s Mod * amount of particle overlap
- $\mathbf{F_{dir}}$ = particle overlap / norm(particle overlap)
- $\mathbf{F} = \mathbf{F_{mag}} * \mathbf{F_{dir}}$
- Populate **accel** vector

Particle-wall interaction:

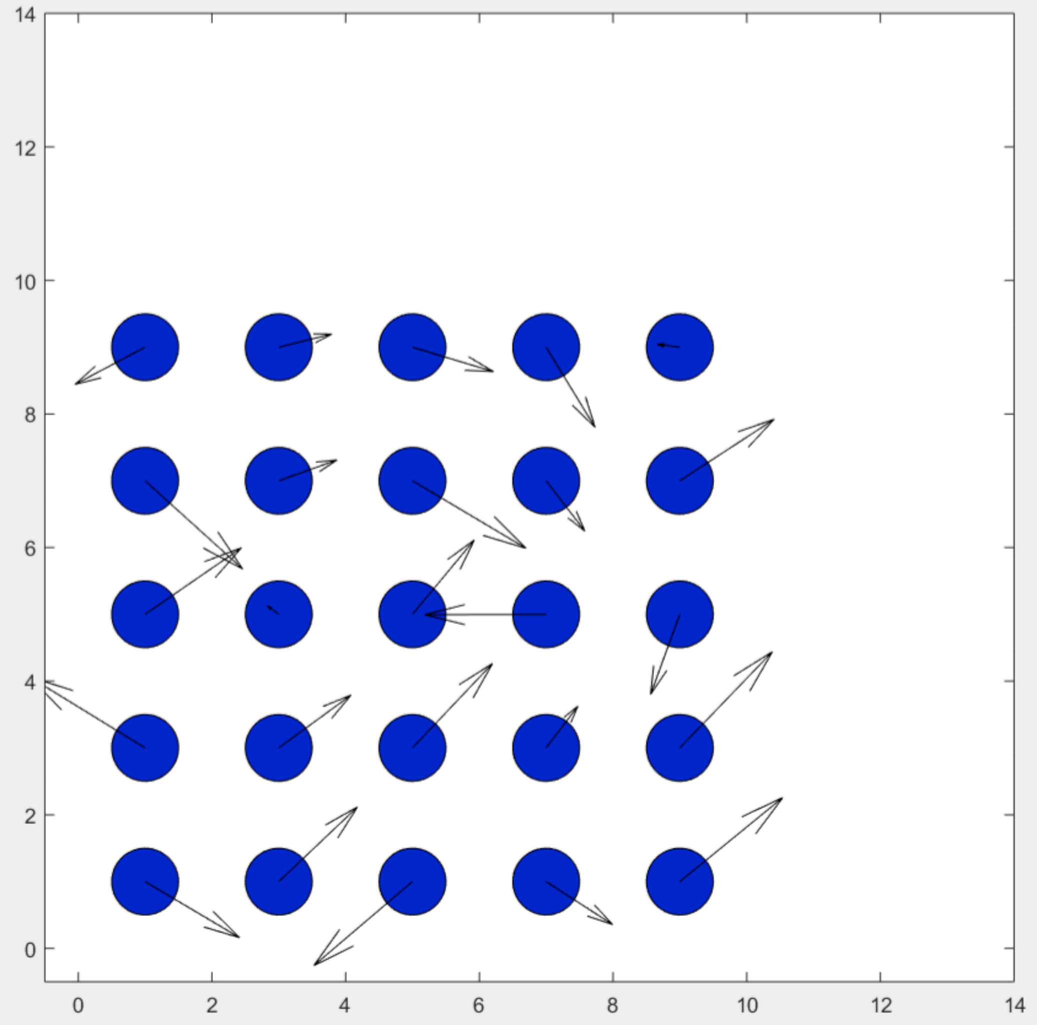
If particle center – particle radius < wall coordinate

- $\mathbf{F_{mag}}$ = Young’s Mod * amount of particle overlap
- $\mathbf{F_{dir}}$ = particle overlap / norm(particle overlap)
- $\mathbf{F} = \mathbf{F_{mag}} * \mathbf{F_{dir}}$
- Repopulate **accel** vector

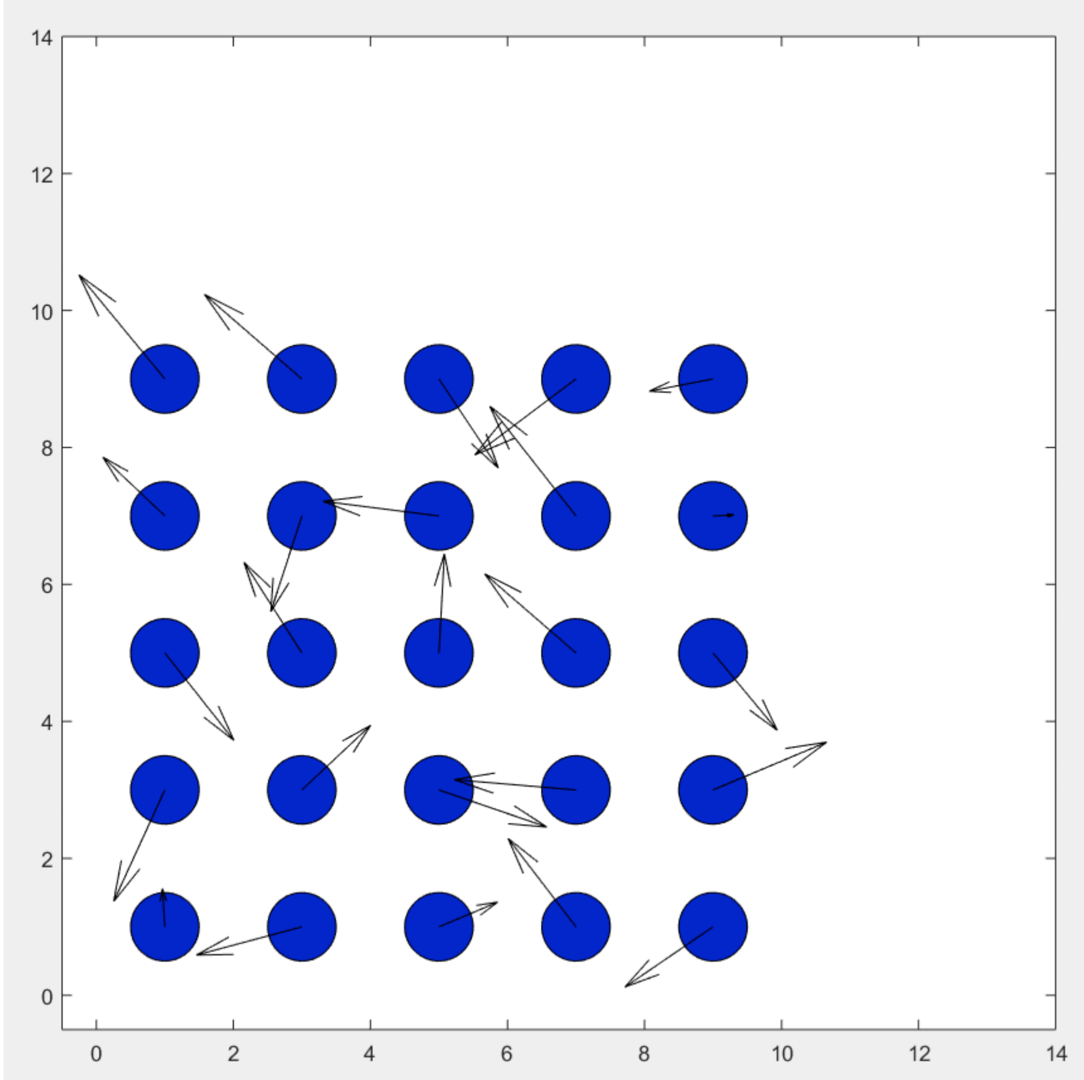
Add **g** to all $\mathbf{accel_y}$ **& /** particle **mass**

Populate dydt vector = [$\boldsymbol{x}\ \boldsymbol{i}\ \boldsymbol{x}i\ \ \boldsymbol{yi}\ \ \boldsymbol{yi}$ **...**]

2D DEM: High E



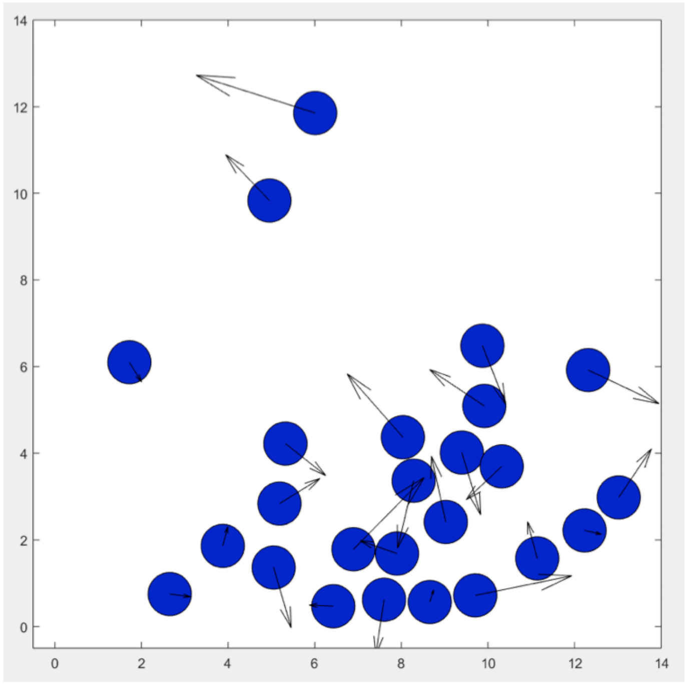
2D DEM: Low E



2D DEM Example: Limitations

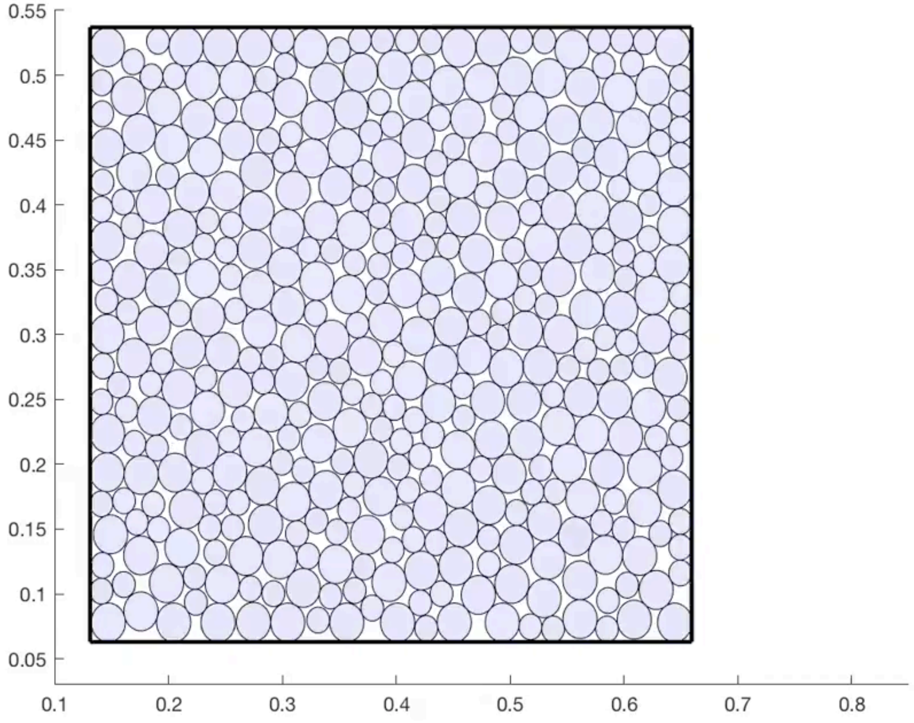
Made the following assumptions/simplifications:

- No dissipative forces
    - Friction: (Amantons’ Law or Hertzian Contact Theory)
    - Ambient fluid resistance (air/liquid)
- No particle rotation
    - Would need to calculate torque, moment of intertia...

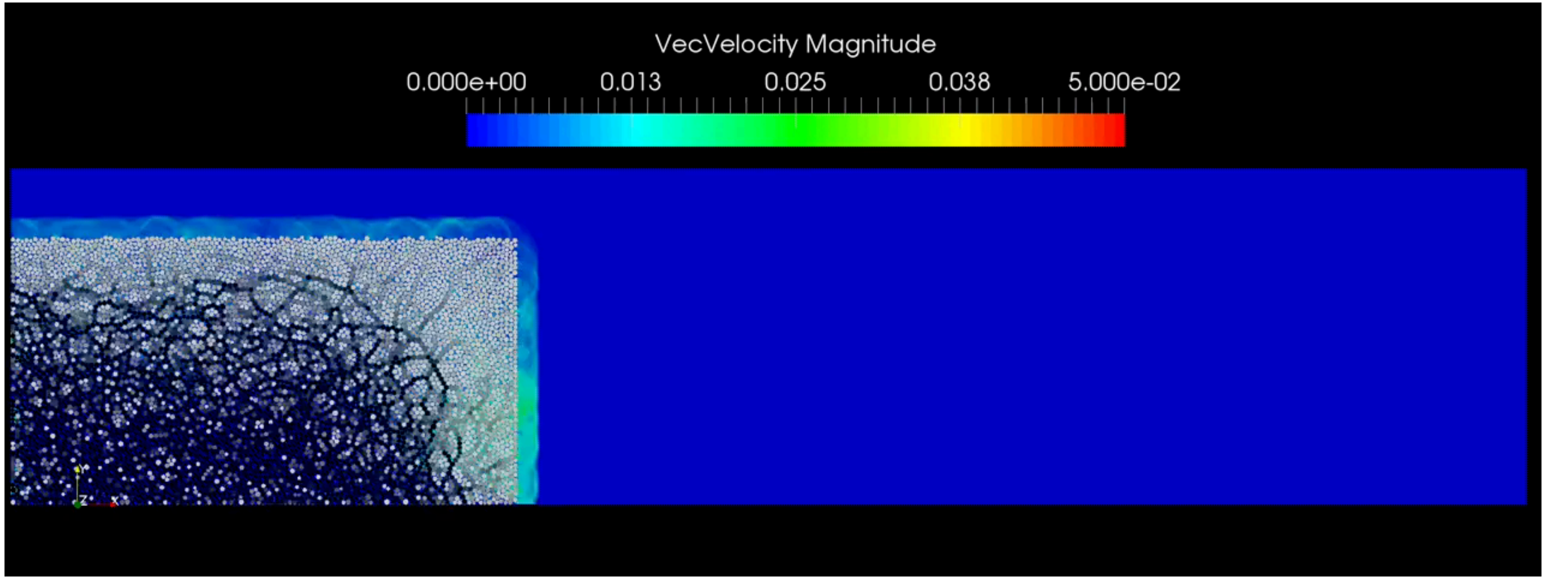


Applications: Real Systems

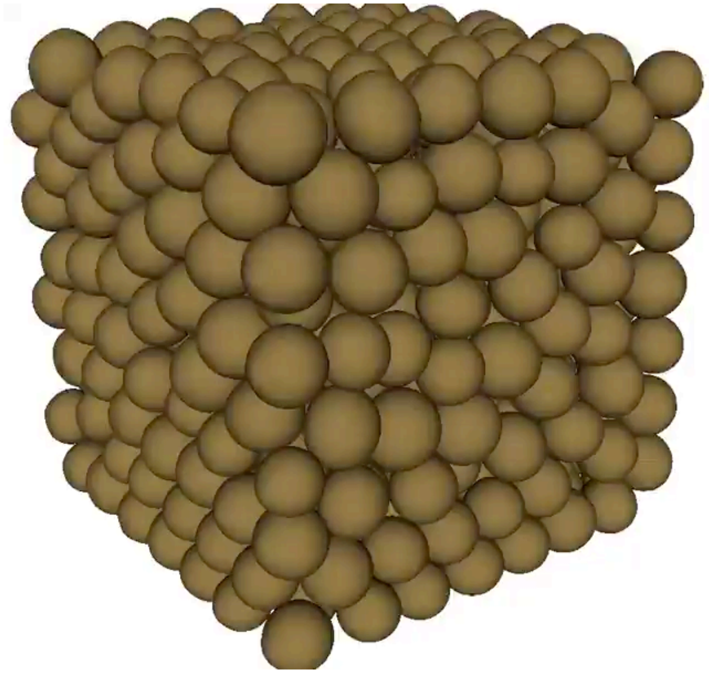
Applications: Shearing Jammed Granular System

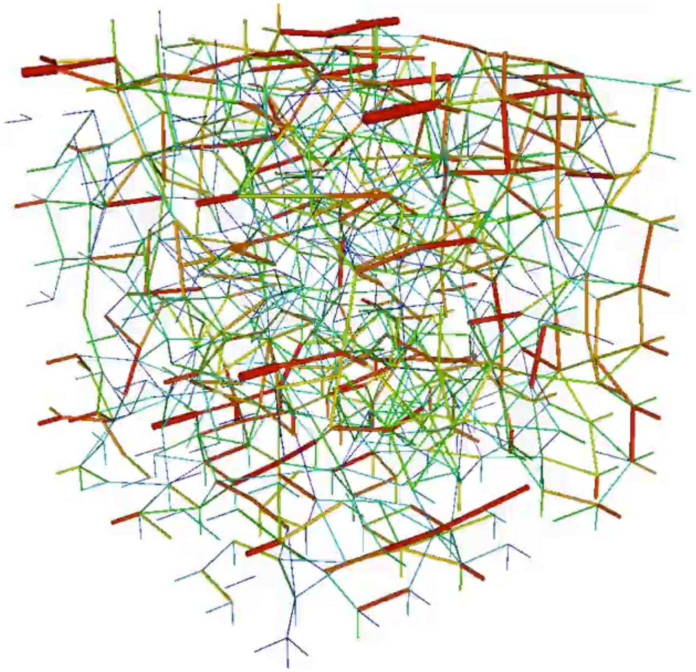


Applications: Granular Avalanche

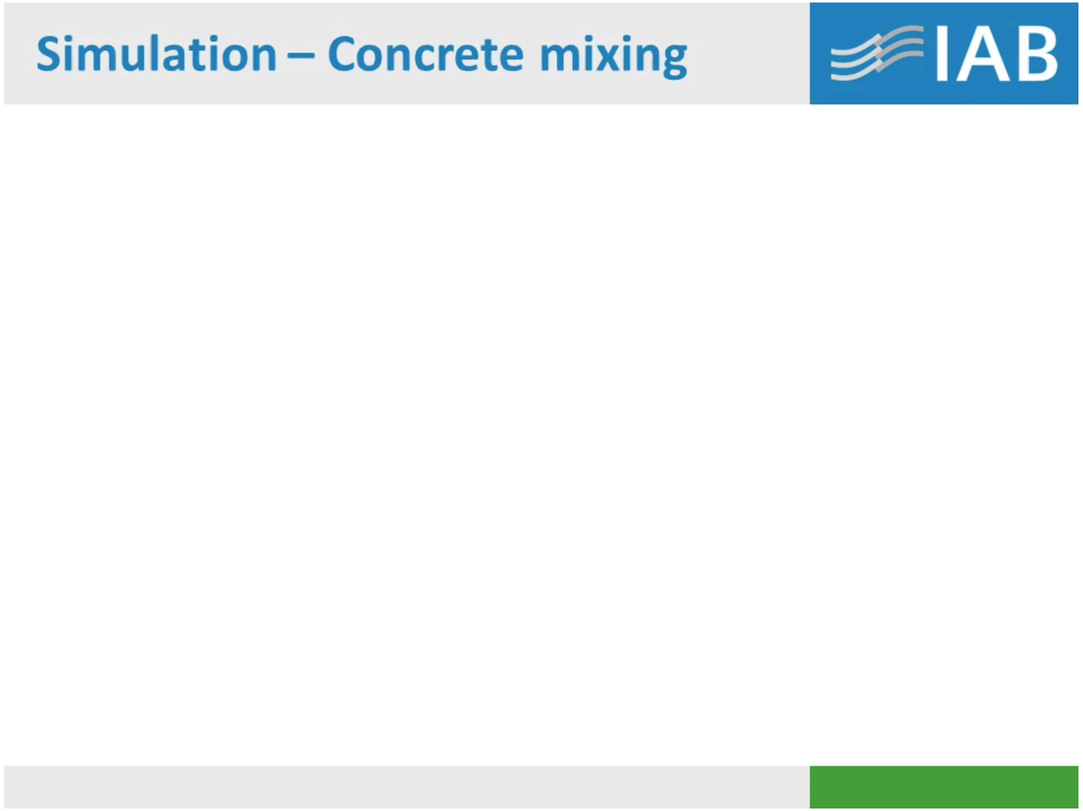


Applications: Granular Force Networks



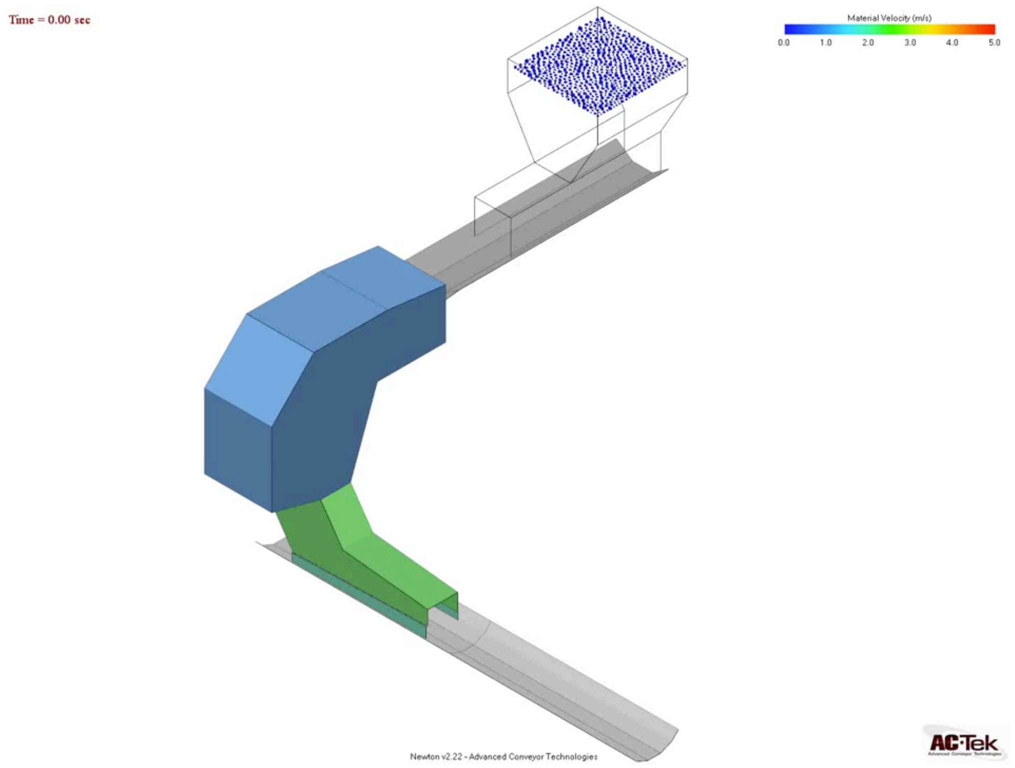


Applications: Concrete Mixing



Simulation – Concrete mixing

Applications: Grains Falling into Hopper


Time = 0.00 sec
Material Velocity (m/s)
0.0
1.0
2.0
3.0
4.0
5.0
Newton v2.22 - Advanced Conveyor Technologies
AC·Tek
Advanced Conveyor Technologies

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^{TM} 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 35: Cambridge-Berkley Geomechanics Research Group, https://www.youtube.com/watch?v=Rlb50Ed6H6Y

Slide 36: Bob Behringer, Center for Nonlinear and Complex Systems, https://www.youtube.com/watch?v=kxmqRQjeyDA&feature=youtu.be

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

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

Slide 39: Helix Technologies, https://www.youtube.com/watch?v=9_-2tsoImJM&feature=youtu.be

Extras…

Governing Equations:

DEM uses two types of governing laws:

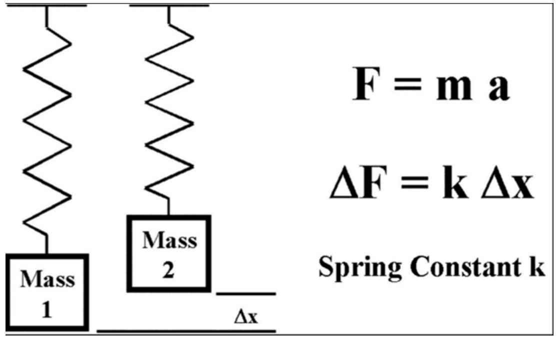
- Newton's Second Law of Motion

  F = MA

- Force-Displacement Law

  Hooke's law, friction etc…

- Time Step



Generalised Hooke's Law

$$\varepsilon_x = \frac{1}{E}\left[\sigma_x - v\left(\sigma_y + \sigma_z\right)\right]$$

$$\varepsilon_y = \frac{1}{E}\left[\sigma_y - v\left(\sigma_x + \sigma_z\right)\right]$$

$$\varepsilon_z = \frac{1}{E}\left[\sigma_z - v\left(\sigma_x + \sigma_y\right)\right]$$

Shear stress-strain relations

$$\gamma_{xy} = \frac{1}{G}\tau_{xy}$$

$$\gamma_{yz} = \frac{1}{G}\tau_{yz}$$

$$\gamma_{xz} = \frac{1}{G}\tau_{xz}$$