Cohesive Particle Model using the Discrete Element Method on the Yade platform

Václav Šmilauer

CTU Prague & UJF Grenoble

24 June 2010
Outline

1. Discrete Element Method
   - Cohesive Particle Model
     - Overview
     - Stresses
     - Calibration

2. Yade
   - Introduction
   - Simulation
   - Code
   - Functionality

3. Yade-powered projects
   - Past
   - Present
Particle models history

- in mechanics since 1970 (Cundall, soil mechanics)
- continuum = particles + contacts, relatively simple contact laws
- suitable for "discontinua" behavior
- different flavors of particle methods (DOF number, static/dynamic problem, particle type)
Classification

- lattice models (no mass, 3/6 DOFs, spring/beam links between particles; 2D variants)
- mass-spring (3 DOFs, explicit dynamics)
- DEM (discrete element method) — particles with 6 DOFs, rigid shape (rigid), inter-particle collisions
- DEM+FEM (“multi-body dynamics”) — particles are deformable (FEM) and collide (DEM)
Classification

- lattice models (no mass, 3/6 DOFs, spring/beam links between particles; 2D variants)
- mass-spring (3 DOFs, explicit dynamics)
- DEM (discrete element method) — particles with 6 DOFs, rigid shape (rigid), inter-particle collisions
- DEM+FEM ("multi-body dynamics") — particles are deformable (FEM) and collide (DEM)
Classification

- lattice models (no mass, 3/6 DOFs, spring/beam links between particles; 2D variants)
- mass-spring (3 DOFs, explicit dynamics)
- DEM (discrete element method) — particles with 6 DOFs, rigid shape (rigid), inter-particle collisions
- DEM+FEM (“multi-body dynamics”) — particles are deformable (FEM) and collide (DEM)
Classification

- lattice models (no mass, 3/6 DOFs, spring/beam links between particles; 2D variants)
- mass-spring (3 DOFs, explicit dynamics)
- DEM (discrete element method) — particles with 6 DOFs, rigid shape (rigid), inter-particle collisions
- DEM+FEM (“multi-body dynamics”) — particles are deformable (FEM) and collide (DEM)
DEM and DEMs

- **smooth**
  - collision prediction
  - $t$ – between collisions
  - restitution equations
  - gas dynamics (few contacts)

- **non-smooth**
  - collision as shape overlaps
  - $t$ – a given $\Delta t$
  - repulsive force, integration of motion equations
  - dense packings
DEM and DEMs

- **smooth**
  - collision prediction
  - $t$ – between collisions
  - restitution equations
  - gas dynamics (few contacts)

- **non-smooth**
  - collision as shape overlaps
  - $t$ – a given $\Delta t$
  - repulsive force, integration of motion equations
  - dense packings
Outline

1. Discrete Element Method

2. Cohesive Particle Model
   - Overview
   - Stresses
   - Calibration

3. Yade
   - Introduction
   - Simulation
   - Code
   - Functionality

4. Yade-powered projects
   - Past
   - Present
Overview

- concrete (generally cohesive-frictional materials)
- contacts have 3 DOFs: normal and shear strains $\varepsilon_N$, $\varepsilon_T$ and stresses $\sigma_N$, $\sigma_T$
- model features:
  - tensile damage + visco-damage
  - compressive plasticity
  - shear plasticity + visco-plasticity
- parameters mostly with a physical meaning
- calibration procedures designed
Normal stress evaluation

\[
\sigma_N = \left[1 - \omega H(\varepsilon_N)\right] k_N \varepsilon_N + \sigma_{Nv}(\dot{\varepsilon}_{Nd})
\]

- \(\varepsilon_N\) normal strain
- \(k_N\) contact normal modulus
- \(\omega\) internal damage variable, \(\omega = g(\kappa), \kappa = \max \varepsilon_N\)
- \(g(\kappa)\) damage evolution function (parameters \(\varepsilon_0\) and \(\varepsilon_f\))
- \(\sigma_{Nv}\) damage overstress, evaluated iteratively from induced damage strain rate \(\dot{\varepsilon}_{Nd}\) (\(\tau_d, M_d\))
Normal stress evaluation

\[ \sigma_N = [1 - \omega H(\varepsilon_N)] k_N \varepsilon_N + \sigma_{Nv}(\dot{\varepsilon}_{Nd}) \]

- \( \varepsilon_N \): normal strain
- \( k_N \): contact normal modulus
- \( \omega \): internal damage variable, \( \omega = g(\kappa) \), \( \kappa = \max \varepsilon_N \)
- \( g(\kappa) \): damage evolution function (parameters \( \varepsilon_0 \) and \( \varepsilon_f \))
- \( \sigma_{Nv} \): damage overstress, evaluated iteratively from induced damage strain rate \( \dot{\varepsilon}_{Nd} = (\tau_d, M_d) \)
Normal stress evaluation

\[ \sigma_N = [1 - \omega H(\varepsilon_N)] k_N \varepsilon_N + \sigma_{Nv}(\dot{\varepsilon}_{Nd}) \]

- \( \varepsilon_N \): normal strain
- \( k_N \): contact normal modulus
- \( \omega \): internal damage variable, \( \omega = g(\kappa) \)
- \( \kappa = \max \varepsilon_N \)
- \( g(\kappa) \): damage evolution function (parameters \( \varepsilon_0 \) and \( \varepsilon_f \))
- \( \sigma_{Nv} \): damage overstress, evaluated iteratively from induced damage strain rate \( \dot{\varepsilon}_{Nd} (\tau_d, M_d) \)
Normal stress evaluation

\[ \sigma_N = [1 - \omega H(\varepsilon_N)] k_N \varepsilon_N + \sigma_{Nv}(\dot{\varepsilon}_{Nd}) \]

\[ \sigma_{Nv}(\dot{\varepsilon}_{Nd}) = k_N \varepsilon_0 (\tau_d \dot{\varepsilon}_{Nd})^{M_d} \]

\( \varepsilon_N \) normal strain

\( k_N \) contact normal modulus

\( \omega \) internal damage variable, \( \omega = g(\kappa), \kappa = \max \varepsilon_N \)

\( g(\kappa) \) damage evolution function (parameters \( \varepsilon_0 \) and \( \varepsilon_f \))

\( \sigma_{Nv} \) damage overstress, evaluated iteratively from induced damage strain rate \( \dot{\varepsilon}_{Nd} (\tau_d, M_d) \)

\( \varepsilon_N \) normal strain

\( k_N \) contact normal modulus

\( \omega \) internal damage variable, \( \omega = g(\kappa), \kappa = \max \varepsilon_N \)

\( g(\kappa) \) damage evolution function (parameters \( \varepsilon_0 \) and \( \varepsilon_f \))

\( \sigma_{Nv} \) damage overstress, evaluated iteratively from induced damage strain rate \( \dot{\varepsilon}_{Nd} (\tau_d, M_d) \)
Shear stress evaluation (after $\sigma_N$)

**Shear stress, plasticity function, plastic flow rule**

$$\sigma_T = k_T(\varepsilon_T - \varepsilon_{Tp})$$

$$f(\sigma_N, \sigma_T) = |\sigma_T| - (cT_0(1 - \omega) - \sigma_N \tan \varphi)$$

$$\dot{\varepsilon}_{Tp} = \dot{\lambda} \frac{\sigma_T}{|\sigma_T|}$$
Model parameters

Elastic parameters
moduli $k_N$, $k_T$, interaction radius $R_I$.

Plasticity+damage parameters
limit elastic strain $\varepsilon_0$, damage evolution parameter $\varepsilon_f$; friction angle $\varphi$, cohesion $c_{T0}$.

Viscosity parameters
characteristic time $\tau_d$, exponent $M_d$; $(\tau_{pl}, M_{pl})$

Confinement parameters
hardening strain $\varepsilon_s$, hardening modulus $\tilde{K}_s$, plasticity function parameter $Y_0$. 
Calibrable properties

Scalar values

- Young’s modulus $E$, Poisson’s ratio $\nu$
- tensile and compressive strengths $f_t$, $f_c$
- fracture energy $G_f$

Qualitative matching

- confinement behavior
- rate-dependent behavior
Cohesive particle model & Yade

Václav Šmilauer

DEM

CPM

Overview

Stresses

Calibration

Yade

Introduction

Simulation

Code

Functionality

Yade-powered projects

Past

Present

Uniaxial tension and compression

smooth damage

σ

ε

σ

0.5e7

0.0

-0.5

-1.0

-1.5

-2.0

-2.5

0.0 0.005 0.010

-0.0030 -0.0025 -0.0020 -0.0015 -0.0010 -0.0005 0.0000 0.0005 0.0010

0.0 0.03 0.06 0.09 0.12 0.15 0.18 0.21
Uniaxial tension and compression
Bending setup

- Cohesive particle model & Yade
- Václav Šmilauer
- DEM
- CPM
- Overview
- Stresses
- Calibration
- Yade
- Introduction
- Simulation
- Code
- Functionality
- Yade-powered projects
  - Past
  - Present
Confined compression

\[ \sigma_N \times 10^8 \]

- exp 20 MPa
- num 20 MPa
- exp 200 MPa
- num 200 MPa
- exp 100 MPa
- num 100 MPa
- exp 400 MPa
- num 400 MPa

\[ \varepsilon \]

\[ \sigma_0 \]

\[ \sigma_0 \]

\[ \varepsilon \]
Tensile rate effect

![Graph showing tensile rate effect](image-url)
Compressive rate effect

![Graph showing the compressive rate effect with data points and lines indicating relative strength and log10(strain rate).]
Generalities

- initiated 2004 by Frédéric Donzé
- in c++, on Linux/Unix
- object-oriented, “toolkit of algorithms”
- substantially reworked during the thesis
  - documentation
  - parallel computation
  - batch scheduling system
  - profiling, debugging tools
  - python interface

www.yade-dem.org
launchpad.net/yade
Python as the interface

- scripting language, large library, similar to matlab
- easy to interface with fortran/c/c++
- Yade classes mirrored in python
- scripts efficient for simulation setup, postprocessing
- compatible over Yade’s internal changes
- runtime control & debugging from the command line
## Data components

### Body (particle)

- **Shape**  Sphere, Facet, ...
- **Material**  ElastMat, FrictMat, ...
- **State**  position, orientation, velocity, ...
- **Bound**  for approximate collision detection (Aabb)

### Generalized forces

### Interaction of 2 bodies

- **InteractionGeometry**  different for Sphere+Sphere, Facet+Sphere, ...
- **InteractionPhysics**  internal state of interaction (plasticity variables, damage, history)
Functional components

Engines

- **GlobalEngine** act on all bodies/interactions
- **PartialEngine** act on some bodies/interactions
- **Dispatcher** calls functions based on classes of arguments:
  e.g. **Facet**+**Sphere** needs different function than **Sphere**+**Sphere** collision

Functors

Callable function-like objects. Accept only certain classes and are called by **Dispatchers**.
Simulation structure

- bodies
  - Shape
  - Material
  - State
  - Bound
- forces (generalized)
- increment time by Δt
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
- properties of new interactions
- constitutive law
- compute forces from strains
- simulation loop
- forces
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
- properties of new interactions
- constitutive law
- compute forces from strains
- simulation loop
- forces
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
- properties of new interactions
- constitutive law
- compute forces from strains
- simulation loop
- forces
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
- properties of new interactions
- constitutive law
- compute forces from strains
- simulation loop
- forces
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
- properties of new interactions
- constitutive law
- compute forces from strains
- simulation loop
- forces
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
- properties of new interactions
- constitutive law
- compute forces from strains
- simulation loop
- forces
- miscellaneous engines (recorders, ...)
- position update
- velocity update
- forces → acceleration
- other forces (gravity, BC, ...)
- reset forces
- update bounds
- collision detection pass 1
- collision detection pass 2
- strain evaluation
- geometry
- physics
What it looks like in python I.

Simulation loop in code

```python
O.engines=[
    ForceResetter(),
    BoundDispatcher([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()]),
    InsertionSortCollider(),
    InteractionDispatchers(
        [Ig2_Sphere_Sphere_Dem3DofGeom(), Ig2_Facet_Sphere_Dem3DofGeom(),
        Ip2_FrictMat_FrictMat_FrictPhys()],
        [Law2_Dem3Dof_FrictPhys_Basic()],
    ),
    GravityEngine(gravity=(0,0,-9.81)),
    NewtonIntegrator()
]
```

**Ig2** 2-ary functor creating `InteractionGeometry` (Bo1, Ig2, Ip2, Law2)

**Facet_Sphere** accepting **Facet** and **Sphere** as its arguments

**Dem3DofGeom** returning **Dem3DofGeom** instance
What it looks like in python II.

Simulation data in code

```python
O.materials.append(
    FrictMat(young=30e9, poisson=.3, density=3000, frictionAngle=.5)
)
O.bodies.append([
    utils.sphere((0,0,3), radius=1),
    utils.facet([(-1,-1,0),(1,0,0),(0,1,0)])
])
O.dt=.5*utils.PWaveTimeStep()
```

Running simulation

```python
O.run(10000); O.wait()  # Basic simulation control
O.save('/tmp/a.xml')
print O.bodies[3].state.vel  # inspection of (c++) data
print O.interactions[0,2].geom.normal
print O.materials[0].young
quit()
```
Engines

Loading control (*BoundaryControllers*)

*TriaxialStressController* (stress/strain rate),
*UniaxialStrainer* (strain control), *PeriIsoCompressor*
(periodic iso-stress), *Peri3dController* (periodic stress tensor)

Applying conditions

*GravityEngine* (constant gravity field), *ForceEngine*,
*RotationEngine*, *TranslationEngine*, ...

Algorithms

*InsertionSortCollider* (collision detection), *NewtonIntegrator*
(2\textsuperscript{nd} order central-differences explicit integration scheme),
*GlobalStiffnessTimeStepper* (adjust timestep based on packing stiffness)
Particles and interactions

**Shapes**

Sphere, Facet, Wall, Box. (Tetra, polyhedral grains, . . .).

**Handling collisions** *(InteractionGeometry)*

Handling collisions of $2 \times \text{Sphere}, \text{Facet} + \text{Sphere}, \text{Box} + \text{Sphere}, \text{Wall} + \text{Sphere}.$

**Constitutive laws**

Dry friction (classical DEM), Mindlin’s contact, Plassiard’s formulation, Cohesive-frictional model, rock model, concrete model, capillary effects between grains. (more outside source tree or undocumented)

**Coupling**

OpenFOAM, Comsol, fluids.
Outline

1. Discrete Element Method
2. Cohesive Particle Model
   - Overview
   - Stresses
   - Calibration
3. Yade
   - Introduction
   - Simulation
   - Code
   - Functionality
4. Yade-powered projects
   - Past
   - Present
3d lattice model of tensile concrete fracture.
Behavior of granular media with capillary effects between grains.
Missile impact on concrete structures.
Missile impact on concrete structures.
A finite volumes-DEM coupled formulation for fluid-solid interactions in granular media.
Hydride metal powders in hydrogen storage tanks — swelling & shrinking due to chemical reactions with hydrogen, creating mechanical effects.
Coupling Computational Flow Dynamics (CFD) and DEM — OpenFOAM and Yade.
Anton Gladky, Freiberg

Mineral processing — analyzing rock destruction in the machine.
Mineral processing — analyzing rock destruction in the machine.
Modeling snow grains based on CT scans, as polyhedra which can deform along crystallographic planes.
Interaction between DEM-modeled solid and Lattice Boltzmann Method (LBM) modeled fluid. (Started by Luc Scholtès)
Chiara Modenese + Boon Chiaweng, Oxford

Lunar soil mechanics (started recently)
Fractured rock mass with smooth contact discontinuities; discontinuities can be imported from Discrete Fracture Network Modelers.
Fractured rock mass with smooth contact discontinuities; discontinuities can be imported from Discrete Fracture Network Modelers.
Particle model of concrete, based on continuous formulation (plasticity, rate-dependence, damage).
Thanks for attention

Got questions?