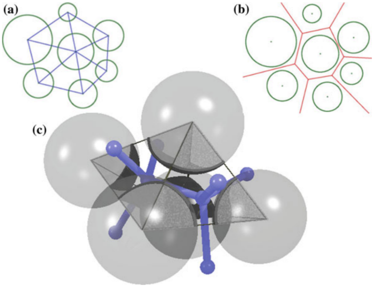
THM short-course: **Day 2**

FlowEngine - Yade's pore finite volume scheme

Robert Caulk[1], Bruno Chareyre[1]

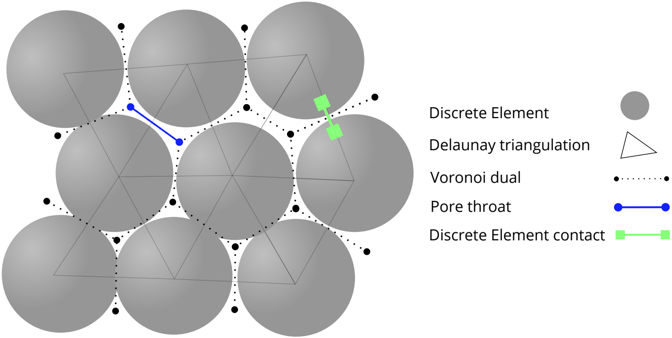
June 21th, 2022

[1]Univ. Grenoble Alpes
Grenoble INP, 3SR



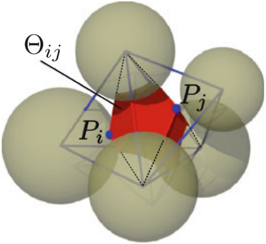
Fundamentals

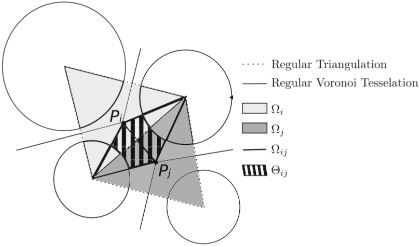
The triangulation



Caulk 2022

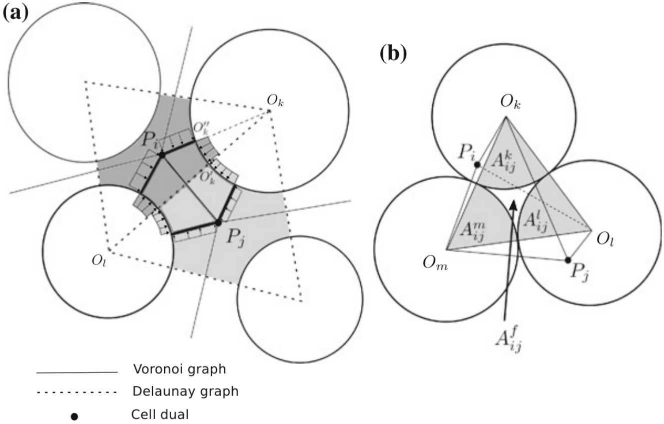
Geometric breakdown





Chareyre 2012

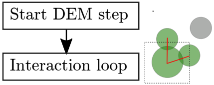
Geometric breakdown



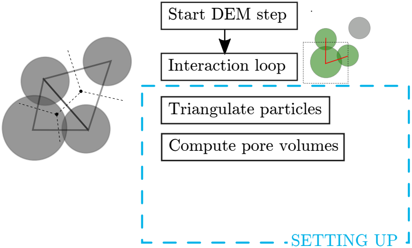
Chareyre 2012

Algorithmic considerations

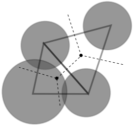
Broad overview



Broad overview

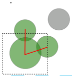


Broad overview



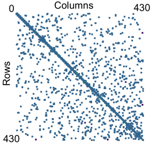
Start DEM step

Interaction loop



- Triangulate particles
- Compute pore volumes
- Build conductivity matrix [$\mathbf{G}$]
- Analyze [$\mathbf{G}$]
- Factorize [$\mathbf{G}$]

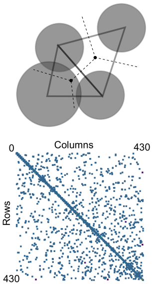
SETTING UP

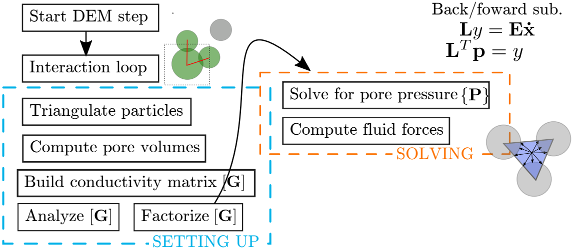


$$\mathbf{G}\mathbf{p} = \mathbf{E}\dot{\mathbf{x}} + \mathbf{Q} \quad \text{Linear system}$$

$$\mathbf{G} = \mathbf{L}\mathbf{L}^T \quad \text{Cholesky decomposition}$$

Broad overview





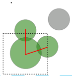
$$\mathbf{G}\mathbf{p} = \mathbf{E}\dot{\mathbf{x}} + \mathbf{Q}$$ Linear system

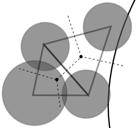
$$\mathbf{G} = \mathbf{L}\mathbf{L}^T$$ Cholesky decomposition

Broad overview

Start DEM step

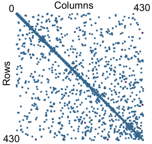
Interaction loop





SETTING UP

- Triangulate particles
- Compute pore volumes
- Build conductivity matrix [**G**]
- Analyze [**G**]
- Factorize [**G**]



$$\mathbf{G}\mathbf{p} = \mathbf{E}\dot{\mathbf{x}} + \mathbf{Q}$$ Linear system

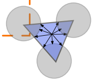
$$\mathbf{G} = \mathbf{L}\mathbf{L}^T$$ Cholesky decomposition

SOLVING

- Solve for pore pressure $\{\mathbf{P}\}$
- Compute fluid forces

Back/foward sub.

$$\mathbf{L}y = \mathbf{E}\dot{\mathbf{x}}$$
$$\mathbf{L}^T\mathbf{p} = y$$



Law of motion

$$\ddot{\mathbf{x}} = \frac{f}{m}$$ Newton's 2nd law

Translating to Yade

FlowEngine - instantiation

Users add FlowEngine() to their O.engines list to initiate the fluid coupling.

```
O.engines = [
        ForceResetter(),
        InsertionSortCollider([Bo1_Sphere_Aabb()]),
        InteractionLoop(
        [Ig2_Sphere_Sphere_ScGeom()],
        [Ip2_FrictMat_FrictMat_FrictPhys()],
        [Law2_ScGeom_FrictPhys_CundallStrack()]
        ),
        FlowEngine(label="flow"),  # Add FlowEngine here
        NewtonIntegrator()
]
```

The broad functionality of `FlowEngine()` can be explored in the trusty `Class Reference`

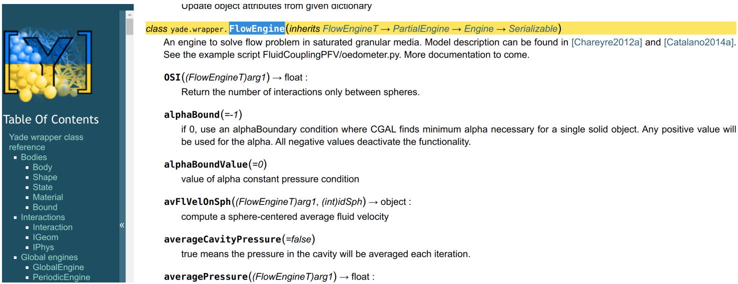


Table Of Contents

Update object attributes from given dictionary

class `yade.wrapper.`**`FlowEngine`**(*inherits FlowEngineT → PartialEngine → Engine → Serializable*)

An engine to solve flow problem in saturated granular media. Model description can be found in [Chareyre2012a] and [Catalano2014a]. See the example script FluidCouplingPFV/oedometer.py. More documentation to come.

`OSI`(*(FlowEngineT)arg1*) → float :

Return the number of interactions only between spheres.

`alphaBound`(*=-1*)

if 0, use an alphaBoundary condition where CGAL finds minimum alpha necessary for a single solid object. Any positive value will be used for the alpha. All negative values deactivate the functionality.

`alphaBoundValue`(*=0*)

value of alpha constant pressure condition

`avFlVelOnSph`(*(FlowEngineT)arg1, (int)idSph*) → object :

compute a sphere-centered average fluid velocity

`averageCavityPressure`(*=false*)

true means the pressure in the cavity will be averaged each iteration.

`averagePressure`(*(FlowEngineT)arg1*) → float :

https://yade-dem.org/doc/yade.wrapper.html#yade.wrapper.FlowEngine

Setting BCs

Users need to set their boundary conditions and flow parameters before the first step of the *coupled* simulation. Considering a typical cuboid shape:

```
# boundaries xmin, xmax, ymin, ymax, zmin, zmax
flow.bndCondIsPressure = [0, 0, 1, 1, 0, 0]
flow.bndCondValue = [0, 0, 100, 50, 0, 0]
flow.boundaryUseMaxMin = [0, 0, 0, 0, 0, 0]
```

Controlling the mesh

Users should be aware that particle deformations will only be reflected in the triangulation is a `flow.meshUpdateInterval` is set:

```
flow.meshUpdateInterval = 200  # remesh every 200 iterations
flow.defTolerance = 0.3  # optional deformation detection
```

Extracting quantities

Quantities of interest, such as pressure, boundary flux, etc. using a plethora of `getters`:

Table Of Contents

`getCellFlux`(*(FlowEngineT)arg1*, *(int)cond*) → float :
Get influx in cell associated to an imposed P (indexed using 'cond').

`getCellFluxFromId`(*(FlowEngineT)arg1*, *(int)id*) → float :
Get influx in cell.

`getCellInvVoidVolume`(*(FlowEngineT)arg1*, *(int)id*) → float :
get the inverse of the cell volume for cell 'id' after pore volumes have been initialized and FlowEngine:iniVoidVolumes = True, or compressibility scheme active with FlowEngine::fluidBulkModulus.

`getCellPImposed`(*(FlowEngineT)arg1*, *(int)id*) → bool :
get the status of cell 'id' wrt imposed pressure.

`getCellPressure`(*(FlowEngineT)arg1*, *(int)id*) → float :
get pressure by cell 'id'. Note: getting pressure at position (x,y,z) might be more usefull, see :yref`FlowEngine::getPorePressure`:

`getCellTImposed`(*(FlowEngineT)arg1*, *(int)id*) → bool :
get the status of cell 'id' wrt imposed temperature.

`getCellTemperature`(*(FlowEngineT)arg1*, *(int)id*) → float :
get pressure in cell 'id'.

`getCellVelocity`(*(FlowEngineT)arg1*, *(Vector3)pos*) → object :
Get relative cell velocity at position pos[0] pos [1] pos[2].

`getCellVolume`(*(FlowEngineT)arg1*, *(Vector3)pos*) → float :
Get volume of cell at position pos[0] pos [1] pos[2].

`getConductivity`(*(FlowEngineT)arg1*, *(int)cellId*, *(int)throat*) → float :
get conductivity from cell and throat, with throat between 0 and 3 (same ordering as incident cells)

`getConstrictions`(*(FlowEngineT)arg1*[, *(bool)all=True*]) → list :
Get the list of constriction radii (inscribed circle) for all finite facets (if all==True) or all facets not incident to a virtual bounding sphere (if all==False). When all facets are returned, negative radii denote facet incident to one or more fictious spheres.

Extracting quantities

getters are typically called inside a PyRunner with the plot.addPlotData():

```
def getPFVquantities():
        plot.addData(
                p = flow.getPorePressure((0.5,0.5,0.5))  #
                ↪  pore pressure at coordinate
                k = flow.getConductivity(10,2)  # get
                ↪  conductivity at cell 10, pore 2
        )

O.engines = O.engines + [PyRunner(iterPeriod=200,
↪  command='getPFVquantities()', label='pfvgetter')]
```

Exporting the mesh

The triangulation can be exported to VTK for Paraview post-processing using `flow.saveVtk()`:

```
O.engines = O.engines + [PyRunner(iterPeriod=200,
↪  command='flow.saveVtk()', label='savevtk')]
```

https://yade-dem.org/doc/yade.wrapper.html#yade.wrapper.FlowEngine.saveVtk

Exporting the mesh

Opening the VTK files in Paraview leverages deep post-processing tools: