



COMSOL-Yade Interface (ICY) instruction guide

Pouyan Pirnia^{1,*}, Francois Duhaime¹, Yannic Ethier¹, and Jean-Sébastien Dubé¹

¹École de technologie supérieure, Laboratory for Geotechnical and Geoenvironmental Engineering (LG2), Montreal, Quebec

*seyed-pouyan.pirnia.1@ens.etsmtl.ca

Keywords: COMSOL; Yade; fluid-particle coupling; install guide; ICY

1 Introduction

This guide outlines the installation of ICY, which is a software interface published in the paper titled Pirnia, Pouyan, et al. "ICY: An interface between COMSOL multiphysics and discrete element code YADE for the modelling of porous media." *Computers & Geosciences* 123 (2019): 38-46. ICY is an interface between COMSOL Multiphysics finite element engine and YADE ("Yet Another Dynamical Engine") open source discrete element code. COMSOL is capable of solving systems of partial differential equations to model multiple physical phenomena simultaneously, such as flow, seepage, chemical reactions, stress-strain behaviour and heat transfer. YADE is capable of solving highly flexible and dynamic particulate simulations. User input for COMSOL is accomplished using a GUI while YADE input is controlled by Python scripts. Meanwhile, the ICY interface enables communication between YADE and COMSOL by a JAVA class. Two project folders named Verification and Application codes are provided by the authors (Pirnia et al., 2018) ([file download](#)). The Verification project code simulates a particle falling in water according to Stokes' law while the Application project code simulates an internal erosion test. The project files include JAVA classes (ICY.java, Clientcaller.java, Reader.java), property files (define.properties), YADE interface scripts (test.py), client-server scripts (client.py and server.py), mesh file (4.mesh), test0.yade (including initial specimen composed of two layers of glass beads: a finer layer on top and a coarser layer at the bottom) and COMSOL models (test.mph). The two projects involve the exchange of different data between COMSOL and YADE. The files that differ for the two projects are test.py (YADE model), test.mph (COMSOL model) and the ICY.java class (main interface code). These are also the file that should be modified to create new applications for the interface.

2 Prerequisites

YADE and ICY are only executable on Linux operating systems. YADE, COMSOL and a JAVA integrated development environment (IDE) need to be installed before using the interface. The codes have been tested under Linux Ubuntu version 14.04 using YADE version 1.14.1 and COMSOL version 5.2. The JAVA classes were compiled and run using the NetBeans IDE version 8.0.2. The JAVA classes and Python scripts may need to be adapted if ICY is run with different versions of YADE and COMSOL, or under a different JAVA IDE.

3 Preparing the COMSOL model

The COMSOL file (test.mph) contains information on geometry, materials, fluid properties, boundary conditions, and mesh. The easiest way to edit these parameters or to define new ones is through COMSOL's graphical user interface (GUI). The main JAVA class can also modify the parameters of the COMSOL model, for example the particle velocity in the verification example, and the permeability values (k1, k2, k3, k4 or k5) or hydraulic head at the top of the specimen (Hupstream) in the Application example.

Figures 1-4 show how to prepare the GUI for the Application project code. Figure 1 shows how to define parameters using the COMSOL GUI. For the Application example, the initial parameter values are arbitrary as they are controlled by the interface. Figure 2 shows how to define the points where the pressure values at the top and bottom of each cell will be obtained from the COMSOL model. Figure 3 shows how the relationship between hydraulic conductivity and the z coordinate (x in COMSOL) is defined using the parameters defined in Figure 1. The GUI can also be used to modify any other parameter on the FEM side of the model, such as the finite-element mesh (Figure 4).

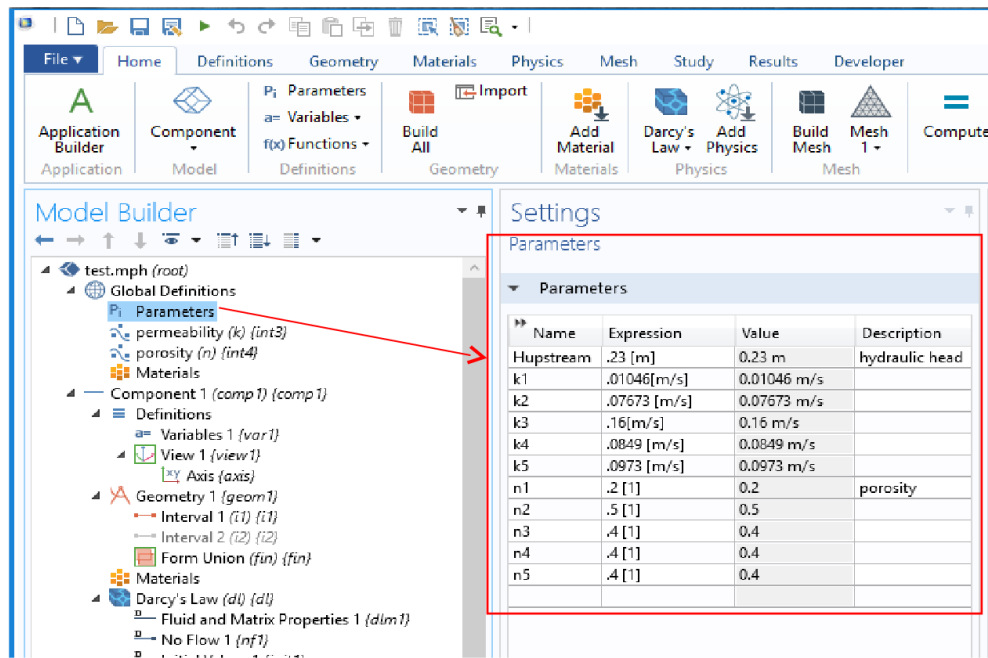


Figure 1. Parameter definition in COMSOL.

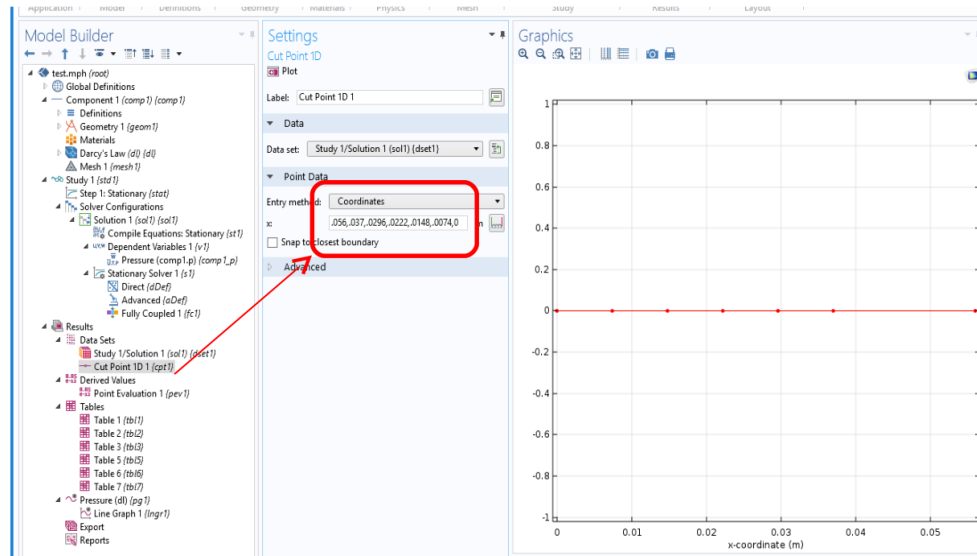


Figure 2. Definition of the points where p values will be obtained

4 Preparing the project in NetBeans

After creating an IDE project folder, the COMSOL plugins have to be added to the project through:

Properties → Libraries → Add JAR/Folder → add all .jar files in plugins folder in COMSOL installation folder (Figure 5).

For running the examples, the COMSOL file (test.mph), YADE script (test.py), client-server (client.py and server.py), mesh file (4.mesh), pressure file (pressure.txt including arbitrary initial pressures for the Application test), dragforce.txt (including drag force for the verification test), test0.yade (including initial specimen) and the JAVA source packages (src folder including ICY.java, Clientcaller.java, Reader.java, define.properties) need to be added to the IDE project folder (Figure 6).

The project's files ICY.java, Reader.java, Clientcaller.java and define.properties have first to be opened in NetBeans. The directories (MainPath and SavingFolder) in the property file (define.properties) have to be changed to correspond to the project directories on the computer, an example of the Application test was presented in Figure 7.

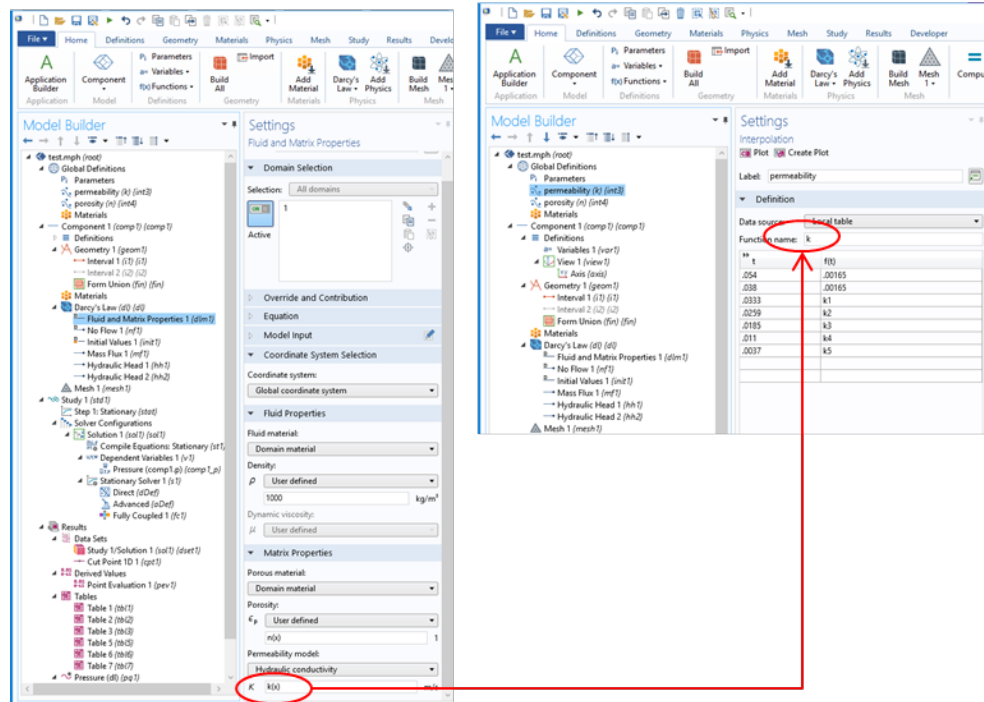


Figure 3. Definition of the hydraulic conductivity function and assignment to the domain

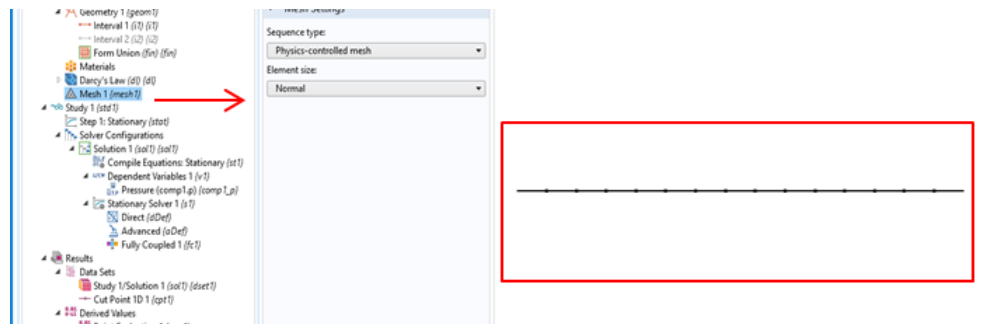


Figure 4. Mesh definition in COMSOL

Users do not need to change anything in Reader.java and clientcaller.java files for the application test. The command lines only need the YADE model and output files names which are taken from the property file (Figure 8).

Figure 9 shows the tasks are performed in the ICY.java class for the application test.

5 Preparing the YADE script

Parameters can be modified in the YADE script for the Application example as presented in Figure 10.

6 Running the interface

Before compiling ICY in NetBeans, the COMSOL server and the python client-server need to be launched. To start the COMSOL server manually, a terminal window is opened and the following command is typed in the COMSOL installation directory:

```
$ . / comsol mphserver
```

For connecting the client to the server, a second terminal window is opened. The following command is typed in the directory containing the server.py file (MainPath directory):

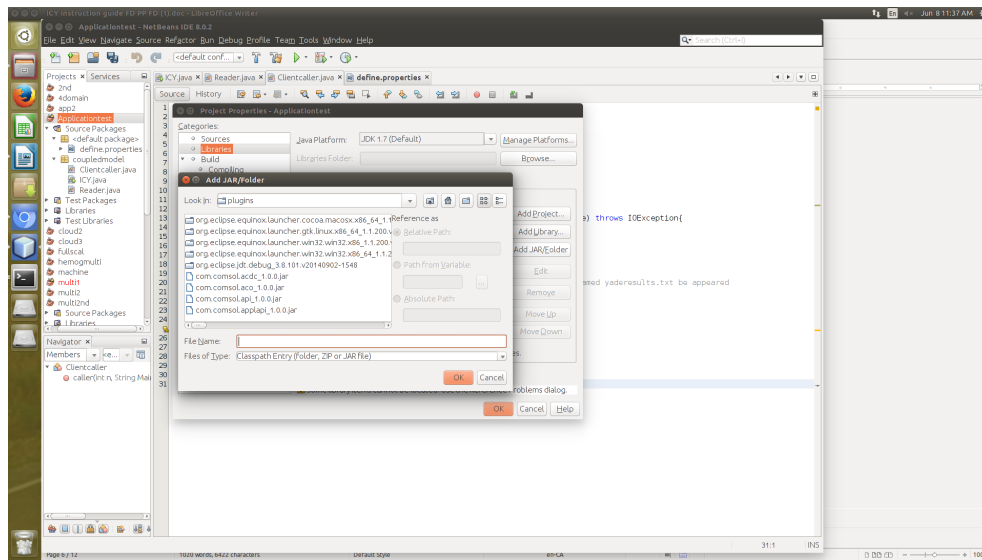


Figure 5. Add .jar files to plugins folder in COMSOL

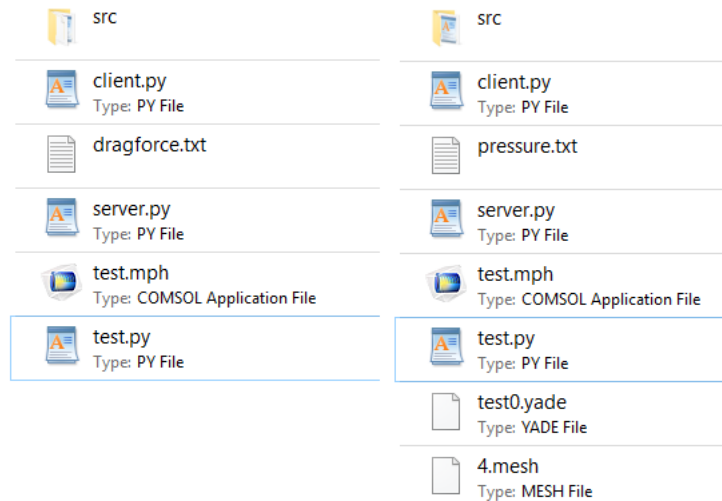


Figure 6. Required files and folder for running the a) Verification example and b) Application test.

```
$ python server.py
```

At this point, ICY can be compiled and run. The simulation progress is printed step by step on the NetBeans screen. It lets users follow the YADE and COMSOL outputs during the simulation.

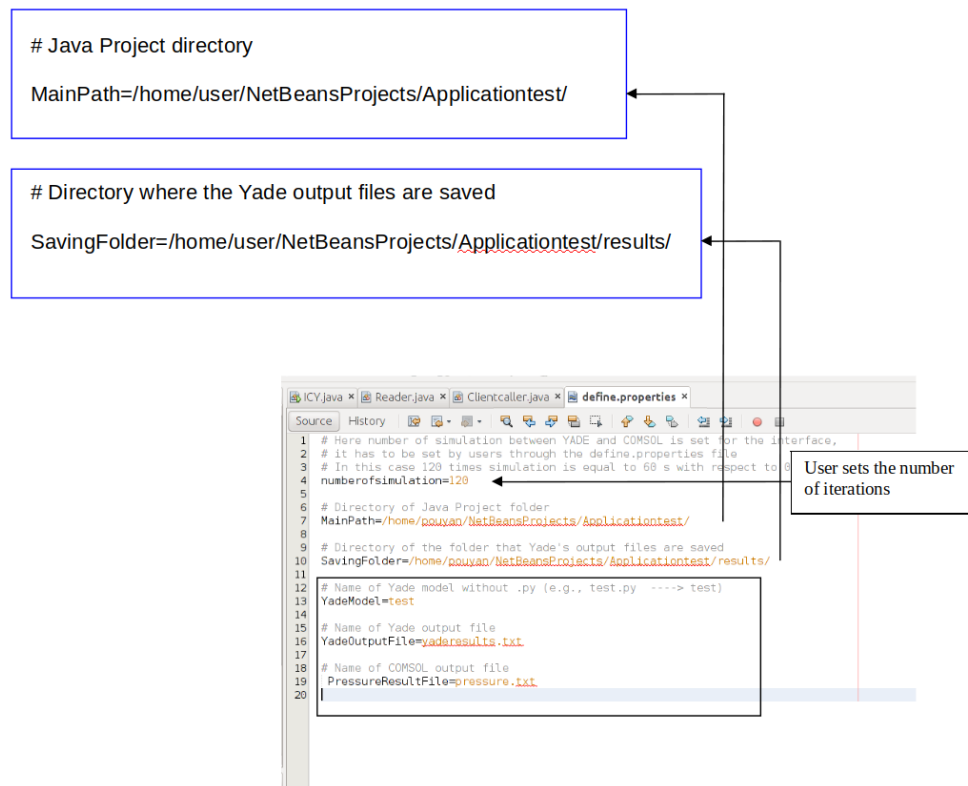


Figure 7. Setting variables and directories in the property file.

The screenshot shows an IDE with two tabs: `Reader.java` and `Clientcaller.java`. The `Reader.java` tab is active, showing the following code:

```

1 package coupledmodel;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.*;
8 import java.util.Scanner;
9
10
11 /**
12  *
13  * @author Pouyan Pirnia
14  */
15
16 public class Reader {
17     public String[] row;
18
19
20     public void method (String MainPath, String YadeOutputFile, String SavingFolder) {
21
22         // Open and read YADE's output file (yaderesults.txt)
23         File file = new File(MainPath+YadeOutputFile);
24
25         try {
26
27             Scanner sc = new Scanner(file);
28
29             // Read the first line, I suppose the first line is header
30             String nextLine = sc.nextLine();
31
32             // Regex to break on any ammount of spaces
33             String regex = "(\\s)+";
34             String[] header = nextLine.split(regex);
35
36

```

The `Clientcaller.java` tab is also visible, showing the following code:

```

1 package coupledmodel;
2
3 import java.io.File;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.util.*;
7
8 /**
9  *
10  * @author pouyan
11  */
12
13 public class Clientcaller {
14
15     public void caller (int n, String MainPath, String YadeModel, String YadeOutputFile) throws IOException{
16
17         // Launch a terminal on Ubuntu and run the comments inside two quotation marks
18         String cmd= "/usr/bin/xterm -e python client.py 'iYadeModel:' 'in';
19         Runtime rt = Runtime.getRuntime();
20         Process pr = rt.exec(cmd);
21
22         // Following syntaxes stop running the rest of program until YADE's output file named yaderesults.txt be appeared
23         File f = new File(MainPath+YadeOutputFile);
24         while (!f.exists()) {
25             try {
26                 Thread.sleep(100);
27             } catch (InterruptedException ie) { /* safe to ignore */ }
28         }
29     }
30 }
31

```

Red boxes highlight the following code snippets:

- In `Reader.java`, lines 23-24: `File file = new File(MainPath+YadeOutputFile);`
- In `Clientcaller.java`, lines 18-19: `String cmd= "/usr/bin/xterm -e python client.py 'iYadeModel:' 'in';`

Figure 8. Names in `Reader.java` and `clientcaller.java` files taken from the property file.

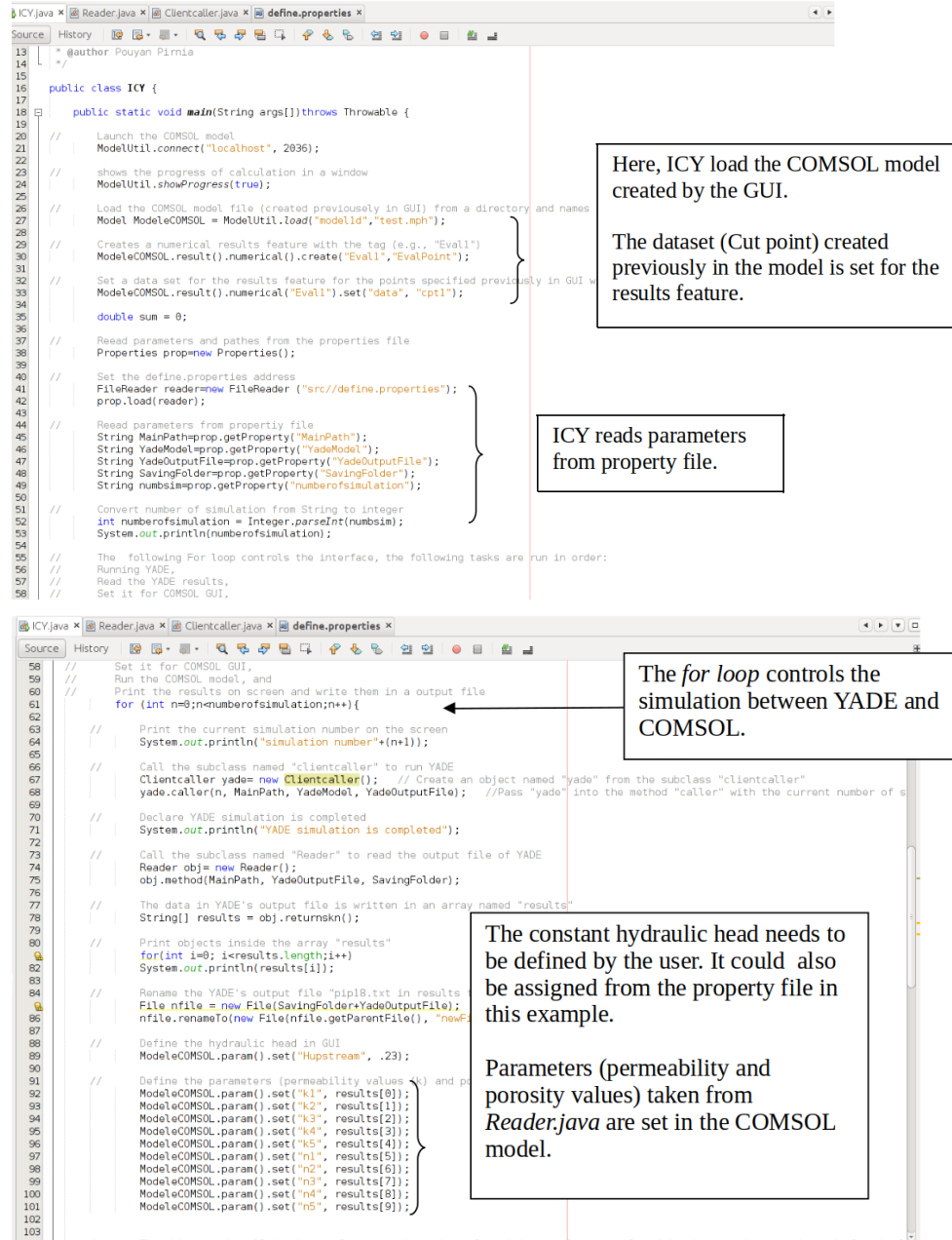


Figure 9. Main tasks in the ICY.java class script.

```

test.py x
28 p6=float(f)
29 p7=float(g)
30 print 'p1,p2,p3,p4,p5,p6,p7 : ', p1,p2,p3,p4,p5,p6,p7
31
32 dp1=p1-p2
33 dp2=p2-p3
34 dp3=p3-p4
35 dp4=p4-p5
36 dp5=p5-p6
37 dp6=p6-p7
38 print 'dp1,dp2,dp3,dp4,dp5,dp6 : ', dp1,dp2,dp3,dp4,dp5,dp6
39
40 # counter is the number of the current simulation
41 global counter
42 counter=counter+1
43 print 'counter : ', counter
44
45 # load YADE's simulation from previous time step, it involves particles and facets positions and forces
46 O.load('test'+counter+'.yade')
47 ##### Test information #####
48 r1 = .01/2.0 # coarse particles radius (m)
49 r2 = .0005/2.0 # fine particles radius (m)
50 E1 = 1e7 # filter Young's modulus
51 E2 = 1e7 # soil Young's modulus
52 poisson1 = 0.3 # filter poisson ratio
53 poisson2 = 0.3 # sand Young's modulus
54 frictionAngle = 4
55 density1 = 2500 # kg/m3
57 density2 = 2500 # kg/m3
58 densitywater = 1000 # kg/m3
59 g=-9.806 # gravity m/s2
60 nfac=100 # number of filter Particles
61 ns=10000 # number of fine Particles
62 G=2.5
63 plate=h
64 npc2=h
65 npc3=h
66 npc=h
67 ##### Defining materials for spheres #####
68 O.materials.append(FrictMat(young=E2,poisson=poisson2,frictionAngle=frictionAngle,density=250000))
69 mat2 = O.materials[1]
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92 bodies1 = [b for b in O.bodies if isinstance(b.shape,Sphere) and b.shape.radius==r1]
93
94 for b in bodies1:
95     b.shape.color = (1,1,0)
96
97 bodies2 = [b for b in O.bodies if isinstance(b.shape,Sphere) and b.shape.radius==r2]
98
99 for b in bodies2:
100     b.shape.color = (0,1,0)
101
102 kw={'material':0}
103 O.bodies.append(utils.geom.facetBox((0.005,0.005,0.1),(0.005,0.005,0.1),wallMask=31))
104 O.bodies.append(utils.geom.mesh('4.mesh', shift=Vector3(0, 0, .03), scale=.001, orientation=Quaternion((0, 0, 0), **kw)))
105 print 'len(O.bodies) final : ',len(O.bodies)
106
107 # FUNCTIONAL COMPONENTS #####
108 # simulation loop #####
109 O.engines=[
110     ForceResetter(),
111     InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb(),Bo1_Wall_Aabb()]),
112     InteractionLoop([
113         # the loading plate is a wall, we need to handle sphere-sphere, sphere-facet, sphere-wall
114         [Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom(),Ig2_Wall_Sphere_ScGeom()], # collision geometry
115         [Ip2_FrictMat_FrictMatPhys()], # collision "physics"
116         [Law2_ScGeom_FrictPhys_CundallStrack()] # contact law -- apply forces
117     ]),
118     NewtonIntegrator(gravity=(0,0,g),damping=.4), # apply gravity force and damping: numerical dissipation of energy
119     PyRunner(command='func()',virtPeriod=.1,label='checker1'), # call the func function (defined below) every 3000 iteration
120     PyRunner(command='save()',virtPeriod=.5,label='checker2')
121 ]
122
123 # set timestep to a fraction of the critical timestep
124 O.dt=.1*utils.PiaveTimestep()
125 print 'O.dt= ', O.dt
126 O.trackEnergy=True
127 O.resetTime() #Reset simulation time
128
129

```

Users can introduce their own materials and mechanical properties to the YADE model.

Particles color

The box and mesh prepared before by Gmsh are defined into the model.

Set the Contact model

Time step can be set by O.dt

Figure 10. Components in the YADE script for the Application example.