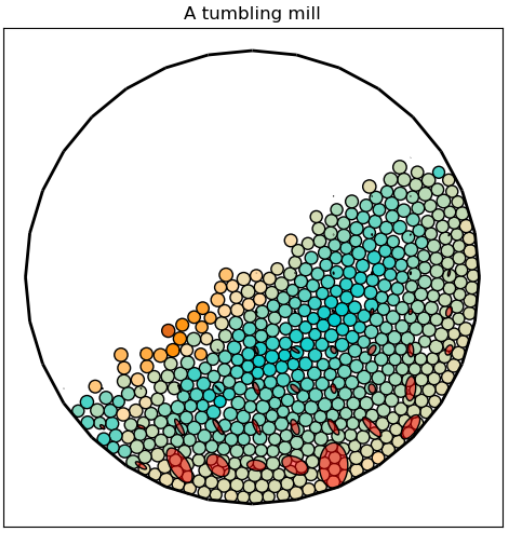
LMECA2300: Homework 3
A tumbling mill



In this third homework, we will simulate a tumbling mill with the Discrete Element Method. Tumbling mills are very common devices used in many different industrial fields: they can be used to mix grains with different properties, to heat metal ores in rotary kilns, or simply to crush grains into smaller pieces. In these devices, the grains are put inside a hollow cylinder whose rotation will, because of friction, set them in motion. The understanding of grain dynamics and of the different flowing regimes is crucial to design effective and efficient processes at the industrial scale. Since friction is at the origin of the grain dynamics inside the mill, it will be included in the contact model, along with the angular velocity of the grains.

The discrete element method with friction

We will track the positions, velocities, and angular velocities of the grains through time using Newton's second law:

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i \mathbf{g} + \sum_\beta \mathbf{f}_i^\beta$$

$$I_i \frac{d\omega_i}{dt} = \sum_\beta \mathbf{r}_i^\beta \times \mathbf{f}_i^\beta$$

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i,$$

with m_i the mass of grain i, I_i its inertia and $\mathbf{r}_i$ the cantilever associated to the contact force. The superscript β indicates all contacts β in which grain i is implied. In order to compute the contact forces, we will use a Kelvin-Voigt model in both the normal and the tangential directions, with an elastic term and a viscous term. For the normal component, the elastic term will be computed with a Hertz law, and the associated contact force will be given by:

$$f_n = \max\Big(0, \begin{cases} k_n|\delta|^{\frac{3}{2}} - \gamma_n u_n, & \delta_n < 0 \\ 0, & \delta_n > 0 \end{cases}\Big)$$

with k_n a spring constant with appropriate units, γ_n a viscosity and $u_n = (\mathbf{u}_i - \mathbf{u}_j)\cdot\mathbf{n}$ the normal relative velocity of the two grains. For the tangential component, the elastic term will be computed with Hooke's law, and the associated contact force will be given by:

$$f_t = -k_t\delta_t - \gamma_t u_t.$$

The value of δ_t can be estimated with the tangential relative velocity u_t. It should be updated at each time step, until the contact between the two grains is broken. Remember that the contact frame evolves with the relative position of the grains: you should therefore take that into account when updating the value of δ_t. Since we will also consider a Coulomb friction law, the value of the tangential force is limited by a threshold as follows:

$$f_t = \min(\mu f_n, |f_t|) f_t/|f_t|.$$

In the case the force is capped because of the threshold, the value of the displacement should be adapted to:

$$\delta_t = -(f_t + \gamma_t u_t)/k_t.$$

The stress tensor

Although granular materials are discontinuous in nature, it is possible to build an approximation of a continuous tensor that describes the stress state inside the grains. This tensor is a volume average of the contact forces in the following sense:

$$\sigma_{ij} = \frac{1}{V} \sum_{\beta} f_i^{\beta} l_j^{\beta},$$

with β denoting the contacts inside the volume V, f_i the ith component of the associated contact force and l_j the jth component of the associated branch vector (the vector from centre to centre of the two grains implied in the contact). Numerically, this tensor will be computed on a regular grid with point spacing $2\sqrt{2}d$, and with a volume $4\pi d^2 \cdot 1$ associated to each grid point.

Segments and contacts

Since we take friction and hence the tangential part of the contact into account, we have to modify the previous definitions of both segments and contacts. We will define segments as lists containing two elements : first, a list of the two points that define the segment. Second, a list that represents the velocity of the segment. The positions of the segments will not be updated with time, but giving them a velocity is what will allow the motion of the grains because of the rotation of the cylinder. Concerning contacts, we have to remember the tangential displacement associated to each contact until the contact is broken. Therefore, a contact will be represented as a list with five elements. The first element is an integer that describes the nature of the contact: its value is `1` if the contact is between a grain and a segment, and `0` if the contact is between two grains. The second element is a list of two integers that correspond to the indices of the two objects (grain and segment or grain and grain) involved in the contact. The third element is a list of two floats that contains the normal and tangential displacements associated to the contact. The fourth element is an array that contains the normal vector associated to the contact. Finally, the fifth element is an array that contains the contact force associated to the contact. This will be useful to compute the stress tensor.

Requirements

In order to load the mesh files and convert them in a segment list to define the boundaries of the problem, you will need the `gmsh` module. If you do not have it yet, you can install it with the following command: `pip install gmsh`. If you have issues installing it, contact Thomas Leyssens **as soon as possible**.

What you have to do

You are asked to:

1. Write a function

   ```
   set_angular_velocity(s,omega)
   ```

   that sets the velocities of the segments in the list `s` so that the entire boundary represented by `s` has an angular velocity equal to the float `omega`.

2. Write a function

   ```
   detect_contacts(c,cold,x,r,s)
   ```

   that detects the contacts between the grains and between the grains and the boundary segments. `c` is a list of contacts to be filled, while `cold` is the list of contacts from the previous time step. If a contact was already detected at the previous time step, the normal displacement and normal vector should be updated, the contact force should be reset, and the tangential displacement should be kept. `x` is an array of size `n`×`2` containing the positions of the grains. `r` is an array of size `n` containing the radii of the grains. `s` is a list containing the boundary segments.

3. Write a function

   ```
   f, t = solve_contacts(c,v,omega,r,s,param,dt)
   ```

   that solves all the contacts of the current time step and returns the resulting contact forces `f` of size `n`×`2` and torques of size `n`×`1` on the grains. `c` is the list of contacts. `v` is an array of size `n`×`2` containing the velocities of the grains. `omega` is an array of size `n`×`1` containing the angular velocities of the grains. `s` is the list of boundary segments. `param` is a list whose first element contains the normal contact parameters `kn,gn` and whose second element contains the tangential contact parameters `kt,gt,mu`. `dt` is the time step.

4. Write a function

   ```
   fn, ft = contact_solve(c,dv,param,dt)
   ```

   that solves a single contact and returns the normal component `fn` and tangential component `ft` of the contact force. `c` is a contact. `dv` is an array of size `2` that contains the relative velocity at the contact point. `param` is a list whose first element contains the normal contact parameters `kn,gn` and whose second element contains the tangential contact parameters `kt,gt,mu`. `dt` is the time step.

5. Write a function

   ```
   sigma = compute_stress_tensor(c,x,p,R)
   ```

   that computes the stress tensor of the grains on a regular grid. `c` is the list of contacts. `x` is an array of size `n`×`2` containing the positions of the grains. `p` is an array of size `m`×`2` containing the grid points at which the stress tensor should be computed. `R` is the radius to use to compute the averaging volume around each grid point.

You are given the implementation of the function `set_segments` that will read a `.msh` file and return a list of segments corresponding to the boundary of you problem. You have to implement the other functions in the file `mill.py`. You are also given the scripts `millRun.py` and `millAnimate.py` so that you can test your functions. `millRun.py` will run the simulation and write the resulting data into the `"result"` directory, while `millAnimate.py` will read that data and display an animation of the result of your simulation. It is advised that you begin with easy initial conditions, such as the ones set by default in `millRun.py`. Once you are confident you code is correct, try it on the `"mill.msh"` with more interesting initial conditions. You should pay attention to writing fast functions : the efficiency of your program will be part of the grading. You have until Friday March 22 23:59 to submit your program `mill.py` on the server in a `.zip` file. You may add comments in your code.