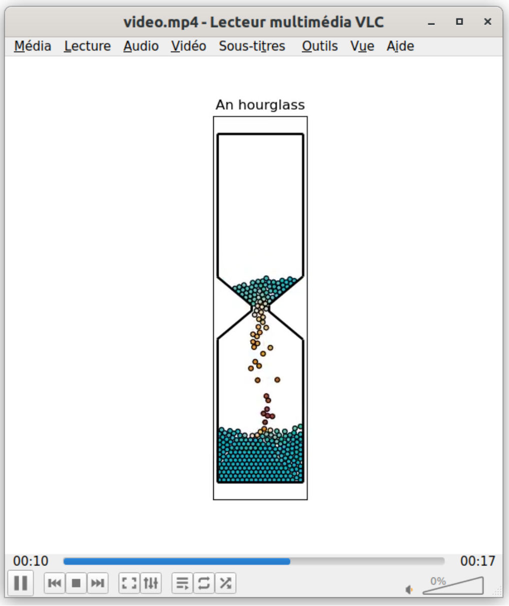
LMECA2300: Homework 4
An hourglass

In this fourth homework, we will simulate the flow of frictionless grains in an hourglass using Non-smooth Contact Dynamics. Hourglasses based on granular materials have been used as time measurement devices for centuries, due to their many advantages. Compared to mechanical clocks, they are much simpler and cheaper, although less precise. Compared to clepsydras (or water clocks), they exhibit a constant flow rate and are much less sensitive to motion and temperature.


video.mp4 - Lecteur multimédia VLC
Média Lecture Audio Vidéo Sous-titres Outils Vue Aide
An hourglass
00:10
00:17
0%

Solving contacts with Non-smooth contact dynamics

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

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

with m_i the mass of grain i. The superscript β indicates all contacts β in which grain i is implied.

A frictionless, perfectly inelastic contact between two grains can be solved as follows:

$$\begin{aligned} v_n &= (\mathbf{v}_i - \mathbf{v}_j) \cdot \mathbf{n} \\ \Delta v_n &= \max(0, -v_n - \max(0, d)/dt) \\ \mathbf{v}_i &\leftarrow \mathbf{v}_i + \frac{\Delta v_n}{W_{nn} m_i} \mathbf{n} \\ \mathbf{v}_j &\leftarrow \mathbf{v}_j - \frac{\Delta v_n}{W_{nn} m_j} \mathbf{n} \end{aligned}$$

where $\mathbf{n}$ is the normal vector, d is the distance between the surfaces of the two grains and $W_{nn} = (m_i + m_j)/m_i m_j$ is the component of the Delassus operator associated to the normal direction.

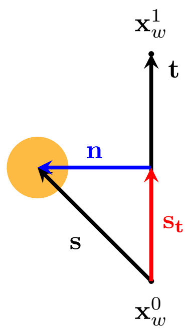
In order to solve the contacts, it is first necessary to detect them. Since interpenetration is proscribed in non-smooth contact dynamics, the detection cannot be done based on the interpenetration distance. Because the method is implicit, it is difficult to predict if two grains will actually be in contact during the following time step because of all the other contacts in which each grain can be implied. The idea is to consider *potential contacts*. Two grains are in *potential* contact if they are separated by less than an *alert distance*[1]. All potential contacts are then solved with the above procedure. If two grains are sufficiently

[1] Note that this distance is computed as the distance between the surfaces of both grains, not their centres

far or have a positive normal velocity so that they won't come into contact when only considering their current state (that is, ignoring other contacts), then the max operator ensures that Δv_n is zero.

Remember, Non-smooth dynamics is an implicit method: it is then necessary to iterate over all the contacts and solve them repeatedly until we reach a given convergence criterion. Typically, this criterion is represented by a tolerance `tol` on the interpenetration between grains. Convergence is reached when, for every contact, the velocity correction Δv_n is smaller than $\texttt{tol}/dt$, with dt the time step.

Contacts with boundaries



As usual, boundary segments are defined by two points $\mathbf{x}_w^0$ and $\mathbf{x}_w^1$. To compute the normal vector $\mathbf{n}$ and the distance between a grain and a segment, one first computes the vectors $\mathbf{t} = \mathbf{x}_w^1 - \mathbf{x}_w^0$ and $\mathbf{s} = \mathbf{x}_g - \mathbf{x}_w^0$. The projection of $\mathbf{s}$ on $\mathbf{t}$ is then computed as $\mathbf{s}_t = \frac{(\mathbf{s}\cdot\mathbf{t})}{|\mathbf{t}|^2}\mathbf{t} = s_t\mathbf{t}$. Finally, the normal vector is obtained as $\mathbf{n} = \mathbf{s} - \mathbf{s}_t$. The distance between the surface of the grain and the segment is then given by $|\mathbf{n}| - r_g$ with r_g the radius of the grain. If this distance is smaller than the alert distance, then the contact between the grain and the segment should be added to the list of potential contacts.

However, this approach will result in considering infinitely long segments, i.e. lines, in the sense that it will still work if the grain is outside the range $(\mathbf{x}_w^0, \mathbf{x}_w^1)$. To avoid that undesired behaviour, it is necessary to check that $0 \leq s_t \leq 1$ before adding the contact to the list of potential contacts.

In addition to boundary segments, it can be relevant to add boundary disks at the ends of the boundary segments. These disks have a radius of 0 and infinite mass. They are useful to prevent the grains from going through the segments in the case where $-r_g/|\mathbf{t}| \leq s_t < 0$ or $1 < s_t \leq 1 + r_g/|\mathbf{t}|$. The contact detection with disks is similar to the one between grains, and the contact resolution is similar to the one with segments.

What you have to do

In this homework, we will use Python classes and objects. If you are not familiar with them, you can easily find information about that on the internet[2]. The problem we will be solving will be represented by an instance of the class `grain_problem`. Each instance of this class contains the positions, velocities, radii and masses of the grains as class variables, as well a list of the boundary segments, a list of the boundary disks, and a list of the contacts of the problem.

You are asked to:

1. Write a function

   ```
   detect_contacts(self,alert)
   ```

   that detects the contacts between the grains and between the grains and the boundaries based on the given alert distance `alert`. Its first operation should be to clear the contact list from the previous time step.

2. Write a function

   ```
   solve_contacts_nlgs(self,tol,dt,itmax=1000)
   ```

   that solves the contacts using a Non-Linear Gauss-Seidel iterative approach: every time a contact is solved, the velocities of the corresponding grains are modified. `tol` is the geometric tolerance used to define the convergence criteria, `dt` is the time step, and `itmax` is the maximum number of iterations in case the solver does not converge.

3. Write a function

   ```
   solve_contacts_jacobi(self,tol,dt,itmax=1000)
   ```

   that solves the contacts using a Jacobi iterative approach: The velocities of the grains are not modified until the end of each iteration, when all contacts have been solved.

In addition to the implementation of the above function, you are asked to answer the following question:

- Which method converges faster, Non-linear Gauss-Seidel or Jacobi? Which one is better suited to the Python language?

You have to implement your functions in the file `hourglass.py`. You are also given the scripts `hourglassRun.py` and `hourglassAnimate.py` so that you can test your functions. `hourglassRun.py` will run the simulation and write the resulting data into the `"result"` directory, while `hourglassAnimate.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 `hourglassRun.py`. Once you are confident you code is correct, try it on the `"hourglass.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 April 19 23:59 to submit your program `hourglass.py` on the server in a `.zip` file. You may add comments in your code.

[2] https://www.w3schools.com/python/python_classes.asp