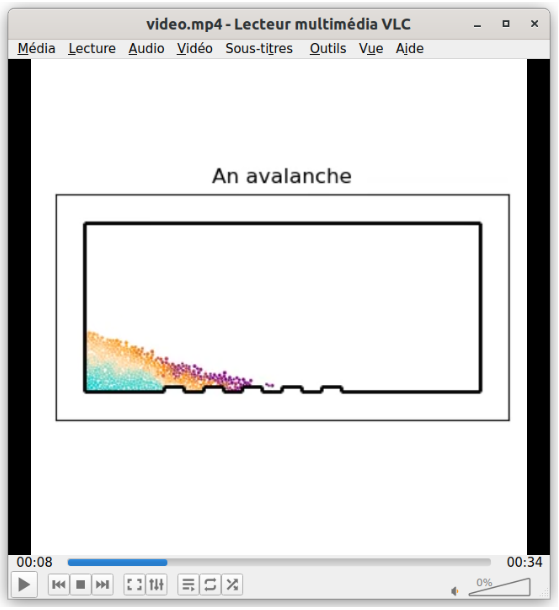
LMECA2300: Homework 5
An avalanche...

In this fifth homework, we will simulate the collapse of a column of grains – in other words, an avalanche – on a rough surface with Non-smooth Contact Dynamics. Notably because of its similarity with real life avalanches, the column collapse experiment is recognised as a benchmark case to study the flowing behaviour of granular materials. The most important characteristic is the column runout, i.e. the farthest distance reached by a grain still in contact with the main grain mass. Basically, the runout answers the question: will the village be destroyed by the avalanche? Other characteristics include the collapse time or the front velocity and kinetic energy, for example. However, we will focus on the runout in this homework.


video.mp4 - Lecteur multimédia VLC
Média Lecture Audio Vidéo Sous-titres Outils Vue Aide
An avalanche
00:08
00:34
0%

Solving frictional 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:

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

$$I_i \frac{d\omega_i}{dt} = \sum_\beta \mathbf{l}_\mathbf{i}^\beta \times \mathbf{f}^\beta,$$

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

with m_i the mass of grain i, I_i its inertia and $\mathbf{l}_i^\beta$ the cantilever associated to the contact force $\mathbf{f}^\beta$ for grain i. The superscript β indicates the contacts in which grain i is implied.

To solve the contacts, we will use contact laws expressed in the local frame. We will hence work with local variables, which can be obtained with the transformation matrix $\mathbf{H}^\alpha$:

$$\mathbf{V}^\alpha = \mathbf{H}^\alpha \mathbf{v}^\alpha = \mathbf{H}^\alpha \begin{bmatrix} \mathbf{v}_i \\ \omega_i \\ \mathbf{v}_j \\ \omega_j \end{bmatrix}$$

where $\mathbf{V}^\alpha$ is the relative velocity at the contact in the local frame of reference and $\mathbf{v}^\alpha$ is the relative velocity at the contact in the global frame of reference. The superscript α denotes the contact in question, but it will be omitted in what follows for the sake of conciseness. The local equation of dynamics can then be written as:

$$\mathbf{V}^+ = \mathbf{V}^- + \underbrace{\overbrace{\mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T}^{\mathbf{W}}\mathbf{P}}_{\Delta\mathbf{V}},$$

where $\mathbf{P}$ is the impulse due to the contact. The matrix $\mathbf{W}$ is called the Delassus operator and is the result of the following operations:[1]

$$\mathbf{W} = \begin{bmatrix} n_x & n_y & 0 & \text{-}n_x & \text{-}n_y & 0 \\ t_x & t_y & r_i & \text{-}t_x & \text{-}t_y & r_j \end{bmatrix} \begin{bmatrix} \frac{1}{m_i} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m_i} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_i} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{m_j} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m_j} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_j} \end{bmatrix} \begin{bmatrix} n_x & t_x \\ n_y & t_y \\ 0 & r_i \\ \text{-}n_x & \text{-}t_x \\ \text{-}n_y & \text{-}t_y \\ 0 & r_j \end{bmatrix} = \begin{bmatrix} W_{nn} & W_{nt} \\ W_{tn} & W_{tt} \end{bmatrix}$$

For disks, $\mathbf{W}$ is diagonal, meaning we can decouple the resolution of the normal and tangent components of the contact. The normal component is solved as before:

$$\Delta V_n = \max(0, -V_n - \max(0, d)/dt)$$

The tangential component is solved as follows:

$$\begin{aligned} &\textbf{If } |V_t| W_{nn} > \mu \Delta V_n W_{tt} : \\ &\qquad \Delta V_t = -\mu \Delta V_n \frac{V_t}{|V_t|} \frac{W_{tt}}{W_{nn}} \\ &\textbf{Else} : \\ &\qquad \Delta V_t = -V_t, \end{aligned}$$

with μ the friction coefficient. Once $\Delta\mathbf{V}$ has been computed, the velocities of the corresponding grains can be updated as follows:

$$\mathbf{v} \leftarrow \mathbf{v} + \mathbf{M}^{-1}\mathbf{H}^T\mathbf{W}^{-1}\Delta\mathbf{V}$$

Computing the runout

Since the runout is defined as the position of the farthest grain still in contact with the main mass (plus its radius), it is necessary to analyse the contact network to find this farthest grain. It can be a good idea to consider this contact network as an undirected graph: this way, it is possible to identify the connected components and find the farthest grain associated to the largest connected component. Don't forget to only consider the contacts between the grains, and to discard the ones with the boundaries.

What you have to do

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, angular velocities, radii, masses and inertias of the grains as class variables, as well as the friction coefficient, a list of the boundary segments, a list of the boundary disks, and a list of the contacts.

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.

[1] These were performed live during the lecture, so you may avoid doing all these operations if you paid attention :-)

2. 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. `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. This time, you have to solve frictional contacts! On the contrary to the Non-Linear Gauss-Seidel method, the Jacobi method is vectorisable, which makes it more suited to languages like Python. This is why it is the only one we will be coding in this homework.

3. Write a function

   ```
   compute_runout(c,x,r)
   ```

   that computes the current runout of the avalanche. `c` is the list of all contacts between the grains and between the grains and the boundaries. `x` is an array of size `n`×`2` containing the positions of the grains. `r` is an array of size `n`×`1` containing the radii of the grains. You are free to use any external library you see fit, but don't forget to add the import instructions in the file you submit to the server.

You have to implement your functions in the file `avalanche.py`. You are also given the scripts `avalancheRun.py` and `avalancheAnimate.py` so that you can test your functions. `avalancheRun.py` will run the simulation and write the resulting data into the `"result"` directory, while `avalancheAnimate.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 `avalancheRun.py`. To validate you code, it can be a good idea to check if the rolling without slipping of the two grains in the easy initial conditions is simulated correctly. Once you are confident you code is correct, try it on the `"avalanche.msh"` mesh with more interesting initial conditions. If you code is slow, probably by lack of proper vectorisation, you can reduce the amount of grains by increasing their size size with `Rmax` and reducing the size of the column with `W` and `H`. You should pay attention to writing fast functions: the efficiency of your program will be part of the grading. You have until Tuesday May 7 23:59 to submit your program `avalanche.py` on the server in a `.zip` file. You may add comments in your code.