Accurate static contact law and high-order temporal schemes for computations of granular flow dynamics

DAMIEN HUET

Engineering Physics programme
Date: February 13, 2019
UBC Supervisor: Anthony Wachs
KTH Supervisor: Anatoli Belonoshko
KTH Examiner: Jack Lidmar
School of Engineering Sciences

Abstract

The smooth Distinct Element Method (DEM) is a numerical simulation technique used in the study of granular materials. The applications are numerous in the industry and in academia, especially in the life sciences field. Smooth DEM implies specifying inter-particles contact forces that are involved during the explicit numerical integration of the Newton's laws of motion.

In this thesis, we implement a tangential contact force accounting for a memory effect and leading to static behaviors that are known to show improved accordance with experiments. However, the validation procedure was tainted by what is believed to be the effect of an inaccurate rolling friction model.

Simultaneoulsy, a multi-step higher order scheme was implemented, allowing the use of larger time steps at almost no additional computation cost. Validation of this scheme was successful in some ideal cases, while the effect on error scaling of time discretization itself is extensively discussed.

Acknowledgements

The present thesis required guidance and support, and I would like to thank the persons who shared their expertise, time and energy with me.

First and foremost, I would like to express my gratitude to Dr. Anthony Wachs of the Department of Mathematics and the Department of Chemical & Biological Engineering at the University of British Columbia, who supervised this thesis. He kindly welcomed me in his group and his door was always open when I needed his advice. His patience, understanding and will to share his knowledge has been very helpful and appreciated.

I would like to thank Dr. Can Selçuk and Dr. Arthur Ghigo for their support and their willingness to get invloved in my research, bringing to the discussion highly relevant ideas and reading suggestions.

I warmly thank my examiner Dr. Jack Lidmar and my supervisor Dr. Anatoli Belonoshko of the Department of Physics at KTH Royal Institute of Technology, who accepted to mark this thesis despite tight schedules.

I also thank Brianne and Arthur who kindly offered to proof read this manuscript.

Contents

List of Figures

List of Tables

Introduction

After water, granular matter is the most manipulated medium in the industry. Today, the study of dry granular flows is of tremendous importance in the energy, pharmaceutical, agri-food or recycling industries, and many more. If a fluid is considered around the particles, the number of applications becomes even larger: from the fluidization properties of geological soils to the heat carrying properties of particle-ladens flows in solar farms facilities, to the numerous investigations in the field of life sciences with the studies of blood flow and the drug-targetting promising breakthrough.

As it turns out, studying a large number of particles, even without considering a surrounding fluid, is an N-body problem which analytical solution is not available and which statistical analysis is only efficient in a very limited number of cases. Today, granular media are studied using the Distinct Element Method (DEM) introduced by Cundall and Strack in 1979 [1]. This method computes the time evolution of particles by resolving each contact individually. Fourty years later, two methods of DEM coexist. In the so-called *stiff DEM*, one computes the post-contact properties by solving an implicit problem constrained by conservation laws [4]. This method leads to accurate results but is limited by the size of the system: at most a few thousand particles can be considered. In real life applications, far more particles are at play, and the use of another method is often mandatory. *Smooth DEM*, however, is a much more scalable approach, in which one computes the post-contact properties by explicitely solving the Newton's laws of motion at each time step. It implies being able to state and compute the forces that arise during contacts, and to integrate them via a numerical scheme.

While there is an extensive number of studies on DEM for spherical particles and some widely spread and validated codes such as LIGHHHTS [11], the vast majority of applications of granular media involve non-spherical and sometimes non-convex particles. *GRAINS3D* is one of the few codes that performs smooth DEM for particles of arbitrary shape, in the framework of high-performance computing [12, 19, 20]. As we shall see in this manuscript, DEM for non-spherical particles − sometimes shortened as non-spherical DEM − requires much more implementation effort due to the increased complexity of particle representations. As a result, there is a latancy in the advances in the field of spherical DEM versus the field of non-spherical DEM.

The aim of this thesis is, on the one hand, to apply to non-spherical DEM a tangential damping force that leads to better static behavior. This so-called memory-friction force has already become a standard in spherical DEM and must be implemented in *GRAINS3D*. On the other hand, another goal of this thesis is to implement a numerical integration method of order higher than two. Indeed having an integration scheme that offers a better order accuracy than the currently implemented order two scheme is an efficient way to reduce the computation time, because it allows the use of larger time steps. Similarly, for the same computation time, a numerical scheme of higher order achieves a better accuracy.

After a review of the main concepts relevant to non-spherical DEM in chapter 1, chapter 2 will be dedicated to the implementation of the memory friction force. Then, we will present in chapter 3 the implementation of some higher order numerical schemes. In each of chapter 2 and 3, a strong emphasis will be put on the critical analysis of the validation procedures and their results.

Chapter 1

Physics and modeling of granular dynamics

Granular matter can be defined as the clustering of a large number of discrete particles which sizes range from a few micrometers – powders – to several meters or more – rocks, asteroids, etc. Even in the case of powders, the size of the particles is large enough so their motion is not affected by thermal motion fluctuations.
In the following section we will present how granular media can be described by the Distinct Element Method (DEM) first initiated by Cundall and Strack [1]. It is this method – more precisely the smooth-DEM – that is implemented in the computer program *GRAINS3D*, which has been used and improved in this entire Master Thesis project.

1.1 The Distinct Element Methods (DEM)

1.1.1 Eulerian and Lagrangian descriptions

In granular dynamics, on the one hand the granular medium presents behaviors that are related to rigid body dynamics and referred to as the Lagrangian description of the system - a single particle can be tracked and its movement can be described solely with the Newton's laws of motions. On the other hand, the medium also shows some peculiar collective and fluid-like behaviors - for example, the medium presents particle segregation or flows inside a recipient -, which motivates the vocabulary "granular flow" and drives the adoption of an Eulerian frame of reference.
The DEM relies on a Lagrangian description. Particles are tracked individually and their trajectories are fully resolved with the laws of motions.

1.1.2 Conduct of a DEM simulation

As presented in Fig. 1.1, the core of a DEM simulation is a loop over a discretized time range. In a typical DEM simulation, each time step starts with a contact detection sequence. It is a very computationally expensive step, that aims at listing the neighbors of each particle. The details of the implementation of this step will be described further in section 1.3.1.

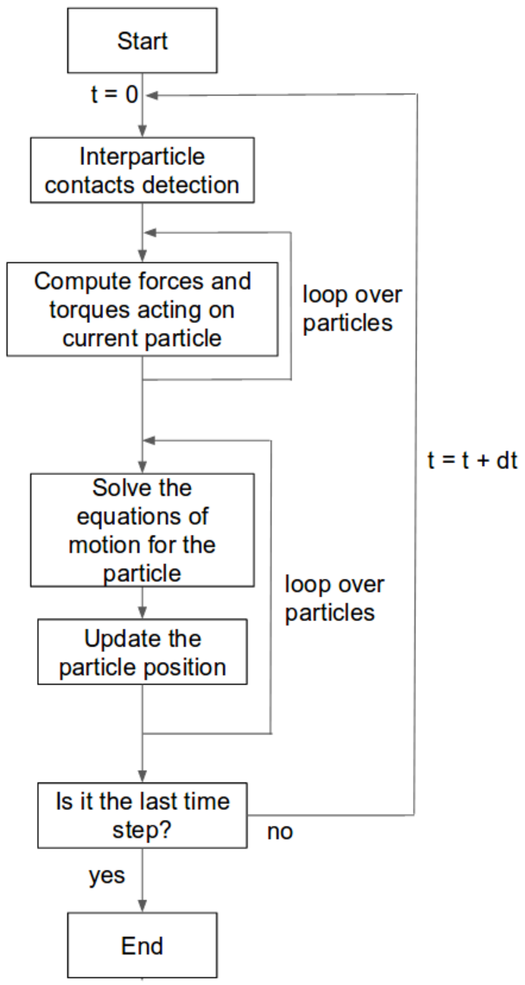

Start
t = 0
Interparticle contacts detection
Compute forces and torques acting on current particle
loop over particles
t = t + dt
Solve the equations of motion for the particle
Update the particle position
loop over particles
Is it the last time step?
no
yes
End

Figure 1.1: Schematic view of the conduct of a DEM simulation

After the contact detection phase, the next step is to resolve the forces and torques acting on each particle. This force computation step is implemented in a subloop over all the particles. Then, the equations of motion are integrated for each particle via an integration scheme. As discussed in section 1.4, there are many integration schemes to choose from and part of this thesis consists in implementing schemes of order higher than two. When the equations of motion are integrated individually for each particle, their positions and velocities are updated and time is incremented by one time step. This whole process loops over itself until reaching the end of the time range: contact detection, forces and torques computation, integration of the equations of motion, update of the positions and velocities.

It is important to note that in a typical DEM simulation – including in our program *GRAINS3D* –, space is not discretized. The coordinates of the center of mass of the particles, as well as their angular position, are computed at each time step and can theoretically take any real value.

1.2 Equations of motion

1.2.1 Laws of motion for 3D rigid bodies

As DEM is a Lagrangian description of the granular medium, each particle trajectory is computed through the integration of the laws of motions for rigid bodies. For a particle i, these equations are written as:

$$\frac{d\overrightarrow{U_i}}{dt} = \frac{\sum \overrightarrow{F}_{i,ext}}{m_i} \tag{1.1}$$

$$\overrightarrow{U_i} = \frac{d\overrightarrow{r_i}}{dt} \tag{1.2}$$

$$\frac{d\mathcal{J}_i\overrightarrow{\omega_i}}{dt} = \sum \overrightarrow{\mathcal{M}}_{i,ext} \tag{1.3}$$

$$\overrightarrow{\omega_i} = \frac{d\overrightarrow{\theta_i}}{dt} \tag{1.4}$$

Where i is the index number of the particle, $\overrightarrow{r}$ is the position of its center of mass, m is its mass, $\overrightarrow{U}$ is its velocity, $\mathcal{J}$ is its inertia tensor, $\overrightarrow{\theta}$ is its angular position, $\overrightarrow{\omega}$ is its angular velocity, and $\overrightarrow{F}_{ext}$ and $\overrightarrow{\mathcal{M}}_{ext}$ are respectively the external forces and torques that act on it. In the above equations, all quantities are evaluated in the galilean frame of reference of the laboratory – often referred to as the world frame of reference. However, it is useful to rewrite EQ (1.3) and EQ (1.4) in the frame of reference of the particle – or body-frame of reference. In all the following, quantities in the body-frame of reference will have the superscript b. Quantities without superscript will implicitely be considered in the world frame of reference. Moreover, if we take care to set the body-frame of reference along the principal axis of inertia of the particle, the inertia tensor $\mathcal{J}_i^b$ is diagonal and EQ (1.3) and EQ (1.4) in the body-frame of reference are of the form:

$$\mathcal{J}_i^b \frac{d\overrightarrow{\omega_i}^b}{dt} + \overrightarrow{\omega_i}^b \times \mathcal{J}_i^b \overrightarrow{\omega_i}^b = \overrightarrow{\mathcal{M}}_{i,ext}^b \tag{1.5}$$

$$\overrightarrow{\omega_i}^b = \frac{d\overrightarrow{\theta_i}^b}{dt} \tag{1.6}$$

To fully describe the system and have a well-posed problem, we now need to describe the forces and torques hidden in the terms $\overrightarrow{F}_{ext}$ and $\overrightarrow{\mathcal{M}}_{ext}$.

1.2.2 Contact modeling

Smooth DEM

Even though it is common in DEM to talk about rigid bodies, in most cases they are actually not strictly rigid and small deformations are introduced during contact in order to be able to integrate the laws of motion. We talk about *smooth contact laws* and by extension *smooth DEM*.

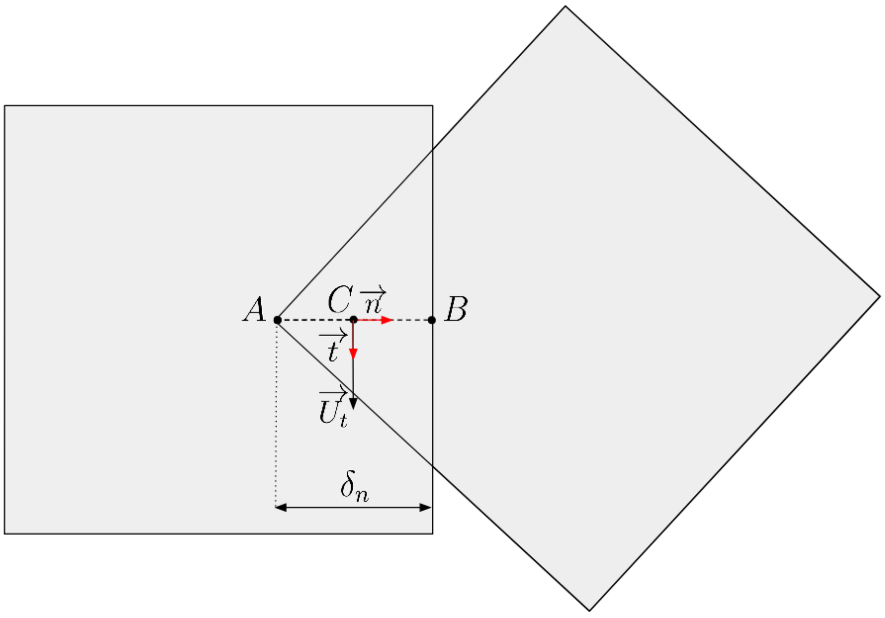


Figure 1.2: Diagram showing the notations introduced during contacts between particles (exaggerated representation).

In smooth DEM, the shape of particles remains constant and the deformation is modeled by a small penetration between particles. A very common contact model is the linear spring-dashpot model shown in FIG. 1.3. This model introduces a hookean normal force as well as viscous normal and tangential frictions forces, with the classic Coulomb saturation condition for the tangential total force: $\left\|\overrightarrow{F_t}\right\| \leq \mu_c \left\|\overrightarrow{F_n}\right\|$. In the following, the subscripts n and t respectively stand for "normal" and "tangential", $\overrightarrow{n}$ denotes the unit vector normal to the contact defined with the two contact points A and B, and $\overrightarrow{t}$ denotes the unit vector along the tangential velocity $\overrightarrow{U_t}$, as shown in FIG. 1.2. We also define the point C, middle point of the segment AB, where the contact forces are applied.

The linear spring-dashpot model

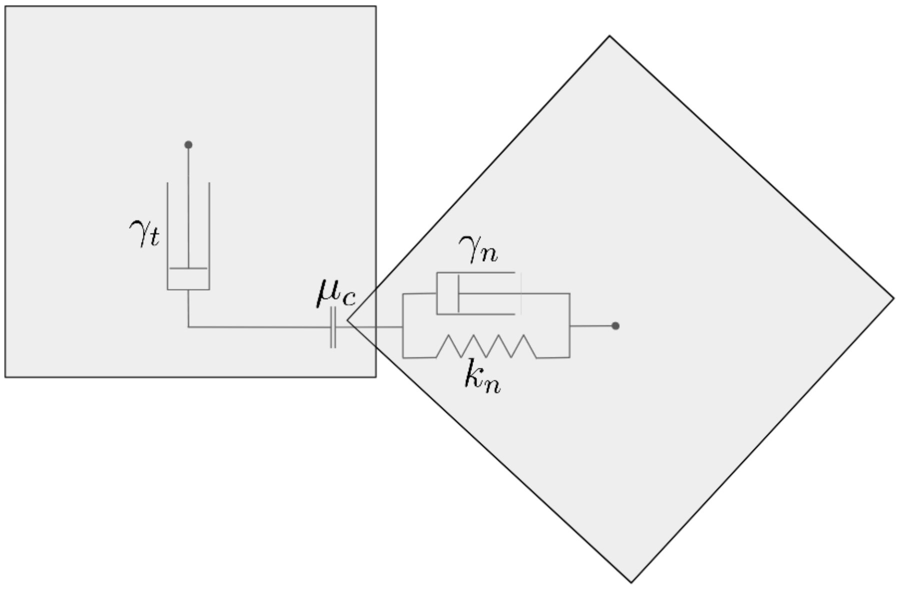


Figure 1.3: The linear spring-dashpot contact model.

1. Normal forces
   - Hookean force: $\overrightarrow{F}_{n,Hooke} = k_n \delta_n \overrightarrow{n}$,
     where k_n is a normal stiffness constant, characteristic of the material and δ_n is the penetration depth of the contact.
   - Normal viscous dissipation: $\overrightarrow{F}_{n,visq} = -2\gamma_n m_{ij} \overrightarrow{U_n}$,
     where m_{ij} and γ_n are constants of the particle material, and $\overrightarrow{U_n}$ is the velocity normal to the contact.
2. Tangential force:

   The tangential viscous dissipation is expressed as
   $\overrightarrow{F}_{t,visq} = -2\gamma_t m_{ij} \overrightarrow{U_t}$, and is saturated by the Coulomb friction criterion such that the total tangential contact force is expressed as $\overrightarrow{F}_{t,tot} = -min\left(\mu_c \left\|\overrightarrow{F}_{n,tot}\right\|, \left\|\overrightarrow{F}_{t,visq}\right\|\right) \overrightarrow{t}$, with $\overrightarrow{t}$ oriented along the tangential velocity $\overrightarrow{U_t}$.

1.3 Computational implementation challenges

DEM programs such as *GRAINS3D* belong to the field of High Peformance Computing due to the high computational resources they require. To give an example, a typical DEM code is designed to run on supercomputers such as CPU clusters ranging from dozens to a thousand processors [19]. The most expensive step in DEM is by far the contact detection phase, followed by the numerous frame transformations of the position and angular position of the particles. The implementations of both steps are described below.

1.3.1 Contact resolution

The contact resolution phase – or contact detection phase – is the step of a DEM simulation where all the translational and rotational positions of the particles are scanned to output a list of contact neighbors for each particle. As the contact point C, the penetration depth δ, and the contact normal $\overrightarrow{n}$ will be needed during later stages of the DEM simulation, the contact detection phase should also provide this information.

The contact detection phase loops over the particles and tests contacts via a binary contact detection function. A naive approach would be that a particle i tests contact with all other particles $j \neq i$, leading to a squared computational complexity. We can do better than this implementation for two reasons: (i) because when we detect a contact between particle i and particle j, we can keep the information that particle j has particle i as a neighbor and therefore save one contact test; and (ii) because we know that two particles located far away from each other will not be in contact. Taking advantage of (ii) leads to the implementation of a low expensive broad-phase detection that outputs only the possibility that particle i and j are in contact. This broad-phase detection is followed if appropriate by the more exepensive narrow-phase detection that effectively tests contact between two particles. As we shall see below, we can have several broad-phase detections in a row, each being narrower than the previous one. The purpose of this cascade of broad-phase detections is to speed up the contact resolution step by avoiding unnecessary calls to the narrow-phase detection. Therefore, each broad-phase detection is by essence less computationally expensive than the narrow-phase detection.

The usual way to implement the broad-phase detection is to discretize space in cells. One particle will only test contacts with particles located near itself, i.e. in its own cell. If the particle is located across an edge of its cell, it will test contacts with particles in the neighboring cell as well. In FIG. 1.4, particles in the same cell are the same color. Only particles of the same color will proceed to the next broad-phase detection. A popular one is the bounded-box method – sometimes referred to as AABB broad-phase detection. In this method, a particle A tests narrow-phase detection with a particle B only if the extrema coordinates of their vertices overlap. Let particle A (respectively, particle B) be a polytope formed by vertices $\{A_i\}_{i\in[1,N_A]}$ with $A_i = (x_{A_i}, y_{A_i}, z_{A_i})$ (respectively, $\{B_i\}_{i\in[1,N_B]}$ with $B_i = (x_{B_i}, y_{B_i}, z_{B_i})$). Then, the narrow-phase detection between particle A and particle B is carried out only if the boxes B_A and B_B overlap, with $B_A = \{(x,y,z), x \in [(x_A)_{min}; (x_A)_{max}], y \in [(y_A)_{min}; (y_A)_{max}], z \in [(z_A)_{min}; (z_A)_{max}]\}$ and B_B defined similarly.

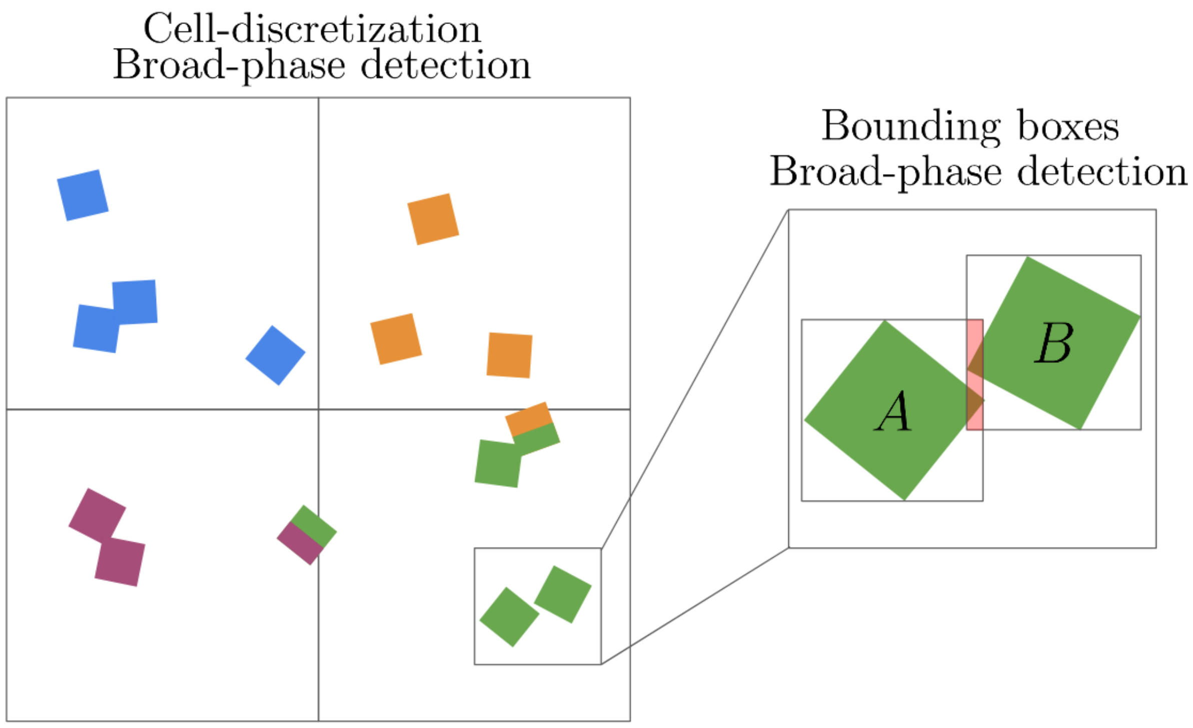


Figure 1.4: Two stages of broad-phase detection: space-discretization in cells and bounding boxes. Particles with the same color will undergo the bounding boxes broad-phase detection, and particles which bounded-boxes overlap will launch narrow-phase detection.

Finally, the narrow-phase detection is what requires the most effort, both in terms of computation and implementation. It would be an easy task if the considered particles were spheres, as the contact point, penetration depth, and contact normal can be computed analytically with little effort. However, in our case particles are of arbitrary shapes like cubes, tetrahedra, or even non-convex shapes; which makes contact-detection far less trivial. In fact, the first contact detection algorithm was proposed only three decades ago by Gilbert, Johnson and Keerthi [2], and has later been improved in a fast, efficient implementation by Van den Bergen in the late nineties [3, 6]. This so-called GJK algorithm has become a standard, and it is widely used in computer graphics and DEM simulations [12, 20, 21, 15]. In the following, we shortly present the principle of this contact detection algorithm.

As any non-convex shape can be decomposed into a set of glued convex bodies, we can restrict our discussion to convex shapes only. The concepts of Minkowski difference, support mapping, and simplex are defined in APPENDIX. A. First, we note that two convex shapes $\mathcal{A}$ and $\mathcal{B}$ are in contact if and only if their Minkowski difference $\mathcal{C}$ contains the origin. From this result, the aim of the contact detection algorithm is to determine whether $\mathcal{C}$ includes the origin. The following GJK distance algorithm ALG. 1 returns zero if the origin lies inside $\mathcal{C}$, otherwise it returns the point from $\mathcal{C}$ that lies closest to the origin. The first steps of the GJK algorithm are shown in FIG. 1.5. In our case, $\mathcal{C}$ is the Minkowski difference of the two particles and its raw distance from the origin presents little interest for us. It turns out Van den Bergen shows in the description of his famous implementation [3] that at each iteration, the algorithm solves

```
Data: A convex body $\mathcal{C}$ with its support mapping function $s_{\mathcal{C}}$, a tolerance $\epsilon > 0$
Find a vector $\mathbf{v}_0$ pointing towards $\mathcal{C}$;
Compute the point $w_0 = s_{\mathcal{C}}(-\mathbf{v}_0)$;
Set $\mathcal{W}_0 = \emptyset$;
Set $k = 1$;
while $k = 1$ or $\|\mathbf{v}_{k-1} - \mathbf{v}_{k-2}\|$ is greater than $\epsilon$ do
    Compute $\mathbf{v}_k$ as the point of the simplex $\mathcal{W}_{k-1} \cup w_{k-1}$ that lies closest to the
      origin;
    Set $\mathcal{W}_k$ as the smallest subset of $\mathcal{W}_{k-1} \cup w_{k-1}$ that contains $\mathbf{v}_k$;
    Compute $w_k = s_{\mathcal{C}}(-\mathbf{v}_k)$;
    Set $k = k + 1$;
end
Result: $w_k$, the point of $\mathcal{C}$ closest to the origin
```

Algorithm 1: The GJK distance algorithm

a linear system of size $n \leq 4$. He then presents how to access the two closest points A and B of the particles $\mathcal{A}$ and $\mathcal{B}$, by cleverly reusing the coefficients that solved the linear system in the last step.

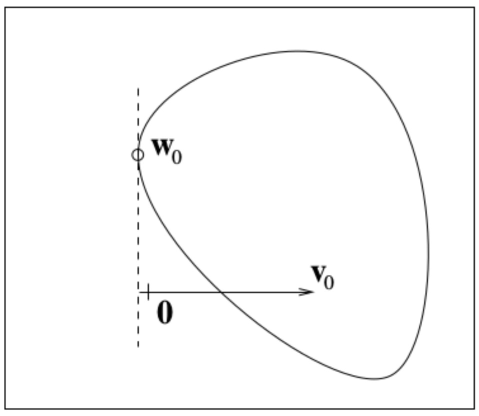
The above GJK distance algorithm allows us to access the closest points of two particles, and is relevant only if the two particles are not in contact. However, our goal is to access the contact points of two overlapping particles. Therefore, once the GJK distance algorithm has confirmed contact by returning the output zero, we need additional computation to access these contact informations. The easiest way to do so is to slightly shrink the two bodies so they are not in contact anymore, and to perform the GJK algorithm again. As shown in FIG. 1.6, this second GJK algorithm iteration will output the two closest points A and B of the shrinked particles, from which the two contact points C and D and the contact normal can then be deduced. The penetration depth is therefore the distance CD and the contact point is the middle point of the segment CD. Another method is the Expanding Polytope Algorithm (EPA) [21], which is about the same complexity as GJK as it involves solving linear systems.

1.3.2 Describing rotation: the use of quaternions

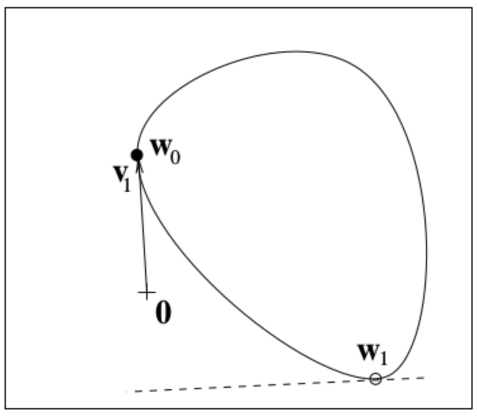
There are three common ways of representing angular position in space: Euler angles, rotational matrices and quaternions. Euler angles can lead to dead-locks that prevent their use in computer graphics, molecular dynamics and DEM. Rotational matrices are often the option one is the most familiar with, but they present some redundancies that are not acceptable in applications that require fast computation such as DEM or computer graphics. In these fields, quaternions are a very powerful tool and we shall briefly define them and present how to manipulate them.

Definition and basic operations

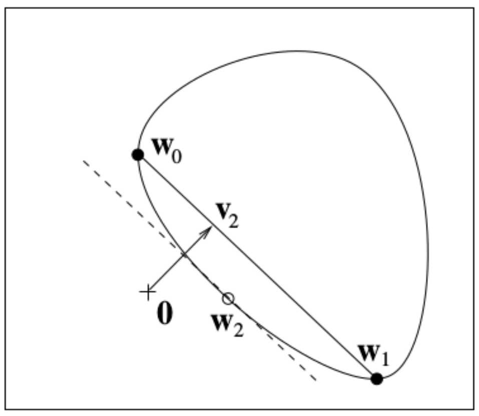
Quaternions were introduced by Sir Hamilton in 1843. He was seeking for an operation in three dimensions that would generalize complex multiplication but he realized that a fourth dimension had to be introduced. This quaternions introduction is largely



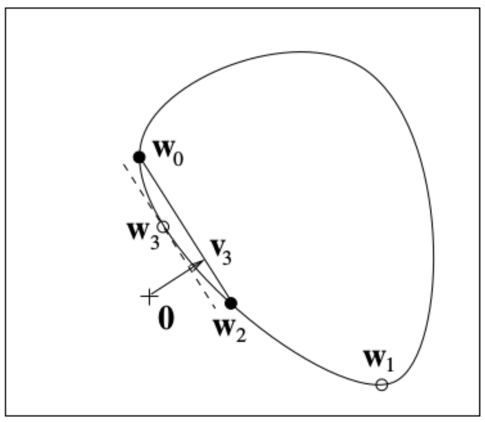
(a) Initialization. $k = 0$, $\mathcal{W}_0 = \emptyset$



(b) Step 1. $k = 1$, $\mathcal{W}_1 = \{w_0\}$



(c) Step 2. $k = 2$, $\mathcal{W}_2 = \{w_0, w_1\}$



(d) Step 3. $k = 3$, $\mathcal{W}_3 = \{w_0, w_2\}$

Figure 1.5: The first interations of the GJK distance algorithm

inspired by the work of Yan-Bin [16]. In the following, vectors are printed in bold type while quaternions and scalars remain in plain type.

A quaternion q is defined as the sum of a scalar q_0 and a vector $\mathbf{q} = (q_1, q_2, q_3)$. Let $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$ and $\mathbf{k} = (0, 0, 1)$, we can write:

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = q_0 + \mathbf{q} \tag{1.7}$$

From this definition we introduce the following operations:

- Addition: The quaternion addition is component-wise. If q and p are two quaternions, we define

$$p + q = (p_0 + q_0) + (p_1 + q_1)\mathbf{i} + (p_2 + q_2)\mathbf{j} + (p_3 + q_3)\mathbf{k} \tag{1.8}$$

- Multiplication: The multiplication of quaternion has to follow the fundamental rule introduced by Sir Hamilton:

$$\begin{cases} \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \\ \mathbf{ij} = \mathbf{k} = -\mathbf{ji} \\ \mathbf{jk} = \mathbf{i} = -\mathbf{kj} \\ \mathbf{ki} = \mathbf{j} = -\mathbf{ik} \end{cases} \tag{1.9}$$

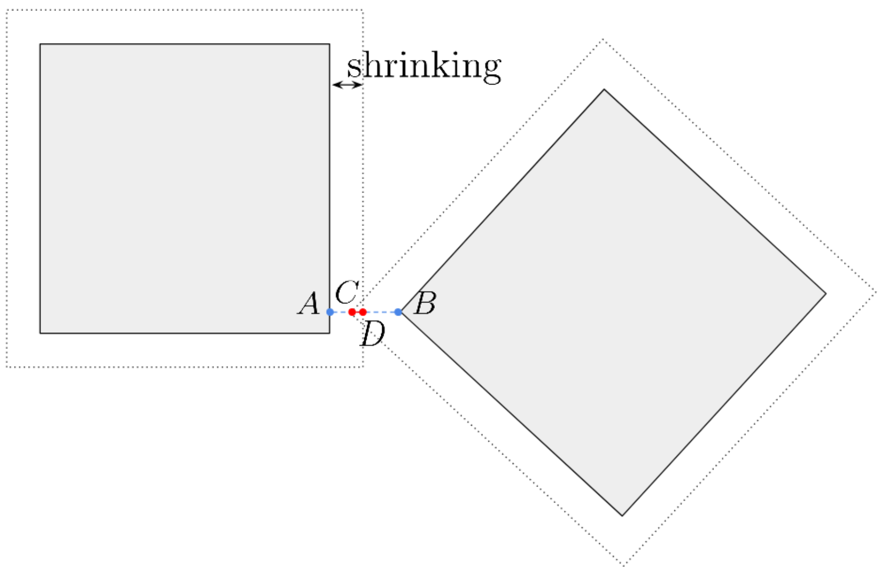


Figure 1.6: When two particles are colliding, the access of the contact information is performed by a second iteration of the GJK algorithm on slighlty shrinked versions of the particles.

In practice, it is more convenient to remember the result from the multiplication of two quaternions p and q, where $\cdot$ is the inner product and $\times$ is the cross product of two vectors:

$$pq = p_0q_0 - \mathbf{p} \cdot \mathbf{q} + p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q} \tag{1.10}$$

We note that due to the cross product the multiplication of quaternions is not commutative: $pq \neq qp$.

- Conjugate: By definition, the conjugate $q^\star$ of a quaternion q is

$$q^\star = q_0 - \mathbf{q} \tag{1.11}$$

and $(pq)^\star = q^\star p^\star$

- Norm: The norm $|q|$ of the quaternion q is defined as

$$|q| = \sqrt{q^\star q} \tag{1.12}$$

A quaternion which norm equals 1 is called a *unit* quaternion.

- Inverse: Let q^{-1} be the inverse of the quaternion q. We have

$$q^{-1} = \frac{q^\star}{|q|^2} \tag{1.13}$$

It is interesting to note that a unit quaternion's inverse is its conjugate: $q^{-1} = q^\star$ if $|q| = 1$.

Quaternion-based rotation operator

Let $q = cos(\frac{\theta}{2}) + sin(\frac{\theta}{2})\mathbf{u}$, with θ a scalar and $\mathbf{u}$ a unit vector. Then, the operator

$$L_q(\mathbf{v}) := q\mathbf{v}q^\star = (q_0^2 - \|\mathbf{q}\|^2)\mathbf{v} + 2(\mathbf{q}\cdot\mathbf{v})\mathbf{q} + 2q_0\mathbf{q}\times\mathbf{v} \tag{1.14}$$

that acts on $\mathbf{v}$ is a rotation of an angle θ about the direction given by $\mathbf{u}$. We also note that $L_{q^\star}(\mathbf{v})$ is the same rotation as $L_q(\mathbf{v})$ with the angle $-\theta$.

In the light of this rotation operator, we now have a routine to rotate a vector $\mathbf{v}$ in space:

- Create a unit quaternion $q = cos(\frac{\theta}{2}) + sin(\frac{\theta}{2})\mathbf{u}$ from the unit direction vector $\mathbf{u}$ of the axis of rotation and the angle θ.
- Apply to $\mathbf{v}$ the rotation operator: $\mathbf{v}' = L_q(\mathbf{v}) = q\mathbf{v}q^\star$. The result $\mathbf{v}'$ of this operation is the rotated vector.

1.4 Integration of the equations of motion

1.4.1 Desired characteristics of integration schemes

Once the forces and torques acting on each particle can be computed at each time step, the motion equations EQ (1.1) and EQ (1.3) can be solved numerically. As emphasized by Dziugys and Peters [5], in addition to meet the usual requirements of stability, low memory storage and accuracy, a good integration scheme in DEM should compute as few force evaluations per time step as possible, as it is a highly expensive computation. Moreover, in DEM and in other Physics applications another desired characteristic of integration schemes is the symplectic property. A symplectic integrator intrinsically conserves the Hamiltonian of the system – in our case, the total energy of the particles – when no dissipative forces are at play. In the following, we will present the most common integration schemes in DEM. The reader seeking for an exhaustive list can refer to [5] and [8].

1.4.2 Explicit time integration schemes in DEM

Integrating the translational motion equation

One-step schemes:

- Forward Euler

$$\begin{cases} x(t+\Delta t) = x(t) + U(t)\Delta t \\ U(t+\Delta t) = U(t) + a(t)\Delta t \end{cases} \tag{1.15}$$

  The Forward Euler method, or just Euler method, is the simplest integration scheme. It uses the first derivative of the considered function (the position $x(t)$ and the velocity $U(t)$ in our case) to compute its value at the next time step. It is a first order method and is therefore rarely used because of its lack of accuracy.

- Taylor expansion

$$\begin{cases} x(t+\Delta t) = x(t) + U(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \\ U(t+\Delta t) = U(t) + a(t)\Delta t \end{cases} \tag{1.16}$$

As the low order of accuracy of the Euler method is not satisfying, one can add more derivatives to the Taylor expansion of the considered function. In the above scheme EQ (1.16), the position $x(t)$ is of second order while the velocity $U(t)$ remains of first order. One could design higher-order Taylor expansion schemes like the below third order in position EQ (1.17), but the additional error introduced by the approximation – with a finite difference scheme – of the derivative of the acceleration $\frac{da}{dt}$ is unclear.

$$\begin{cases} x(t+\Delta t) = x(t) + U(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 + \frac{1}{6}\frac{da}{dt}\Delta t^3 \\ U(t+\Delta t) = U(t) + a(t)\Delta t + \frac{1}{2}\frac{da}{dt}\Delta t^2 \end{cases} \tag{1.17}$$

- Classic fourth order Runge-Kutta

$$\begin{cases} x(t+\Delta t) = x(t) + \Delta t(k_{x1} + 2k_{x2} + 2k_{x3} + k_{x4}) \\ U(t+\Delta t) = U(t) + \Delta t(k_{U1} + 2k_{U2} + 2k_{U3} + k_{U4}) \end{cases} \tag{1.18}$$

$$\text{with}\begin{cases} k_{x1} = U(t) \\ k_{x2} = U(t) + \frac{\Delta t}{2}k_{U1} \\ k_{x3} = U(t) + \frac{\Delta t}{2}k_{U2} \\ k_{x4} = U(t) + \Delta t k_{U3} \end{cases} \text{and}\begin{cases} k_{U1} = a(t) \\ k_{U2} = a(t) + \frac{\Delta t}{2}k_{x1} \\ k_{U3} = a(t) + \frac{\Delta t}{2}k_{x2} \\ k_{U4} = a(t) + \Delta t k_{x3} \end{cases}$$

The very classic fourth order Runge-Kutta presents impressive accuracy and stability behavior [8], but at the price of four force evaluations per time step. Overall, it is an expensive scheme when applied to DEM and multi-step integrators are often preferred. In fact, Kruggel-Emden et al. [8] show that in DEM, the use of simple low order schemes (i.e. second or third order) with small time steps shows better performance for an equivalent accuracy than the use of elaborated higher order schemes with larger time steps. Therefore, the use of this fourth order Runge-Kutta scheme is often restricted to the study of small systems.

Multi-step schemes:

- Leapfrog

$$\begin{cases} x(t+\Delta t) = x(t) + \Delta t U(t+\frac{\Delta t}{2}) \\ U(t+\frac{\Delta t}{2}) = U(t-\frac{\Delta t}{2}) + a(t)\Delta t \end{cases} \tag{1.19}$$

This scheme is by far the most common in DEM, together with the following Velocity Verlet integrator. It provides a second order accuracy for no additional computation than the Euler algorithm. This is due to a shift by half a time step between the position and the velocity. Therefore to start the integration one needs to compute $U(t+\frac{\Delta t}{2})$ from the initial condition $U(0)$, which can safely be perfomed with a simple low order Forward Euler scheme. Moreover, the Leapfrog scheme shows good stability, requires only one force evaluation per time step, and is symplectic. A minor drawback is that the shifted velocity must be interpolated in an expensive post-processing step if one wants to access its data: $U(t+\Delta t) = \frac{U(t+\frac{\Delta t}{2})+U(t-\frac{\Delta t}{2})}{\Delta t}$.

- Velocity Verlet

$$\begin{cases} x(t+\Delta t) = x(t) + \Delta t U(t+\Delta t) + \frac{\Delta t}{2} a(t) \\ U(t+\Delta t) = U(t) + \frac{\Delta t}{2}(a(t) + a(t+\Delta t)) \end{cases} \tag{1.20}$$

  One can modify the Leapfrog algorithm to synchronize the position and the velocity in what is referred to as the Velocity Verlet algorithm, at the cost of storing the acceleration $a(t)$. As based on the Leapfrog scheme, the Velocity Verlet is also of second order, shows good stability, is symplectic, and requires only one force evaluation per time step. Note that this is not an implicit scheme, as long as the acceleration does not depend on the velocity. Indeed, the common way of computing the Velocity Verlet is the following:

  1. Compute $x(t+\Delta t)$
  2. Compute $a(t+\Delta t)$ from $x(t+\Delta t)$
  3. Compute $U(t+\Delta t)$ from $a(t+\Delta t)$ and the stored value of $a(t)$

- Second order Adams-Bashforth

$$\begin{cases} x(t+\Delta t) = x(t) + \frac{\Delta t}{2}(3U(t) - U(t-\Delta t)) \\ U(t+\Delta t) = U(t) + \frac{\Delta t}{2}(3a(t) - a(t-\Delta t)) \end{cases} \tag{1.21}$$

  We should also mention this two-step second order Adams-Bashforth scheme, which is a good candidate for DEM due to its low computation cost (only one force evaluation per time step).

Predictor-correctors:
At last, it is possible to use an implicit algorithm and compute the uncomfortable implicit terms with a lower order explicit scheme. Dviugys & Peters [5] and Kruggel-Emden [8] present some examples, including the association of the implicit Adams-Moulton with the explicit Adams-Bashforth, the third or fifth orders Gear predictor-correctors and the fourth order Hermite predictor-corrector. It is a relatively inexpensive way of reaching the high orders provided by implicit methods without having to compute the expensive inversion of a system.

Integrating the rotational motion equation

The rotational motion is less straightforward to integrate than the translational motion. Indeed, in EQ (1.5) the inertia tensor $\mathcal{J}$ depends on the orientation – embodied by the quaterion q – which is the reason why non-spherical DEM is more expensive and requires more efforts than spherical DEM. From EQ (1.5) the angular acceleration is expressed as follows:

$$\frac{d\omega^b}{dt} = \left(\mathcal{J}^b\right)^{-1}\left(\mathcal{M}^b + \mathcal{J}^b\omega^b \times \omega^b\right) \tag{1.22}$$

A particularly interesting approach to compute ω^b while taking advantage of the quaternion representation has been developped by Zhao et al. [13] and further improved by Seelen et al. [17]. Their Predictor-Corrector Direct Multiplication method (PCDM) intrinsically conserves the norm of the quaternions, which is required for quaternions to properly represent rotations. The details of the equations for the PCDM method are well exposed in [13, 17], and there is little interest in rewriting them here. Below is a summary of the main steps:

1. The angular position q and velocities ω^b are predicted after half a time step $-$ in the case of the Leapfrog algorithm they are computed at $t + \Delta t$ from the previously known value at $t + \frac{\Delta t}{2}$.

2. The torque resultant $\mathcal{M}^b(t{+}\Delta t)$ is computed using the predicted angular position and velocity.

3. With the new torque resultant and the predicted angular velocity, the angular acceleration $\left(\frac{d\omega^b}{dt}\right)(t + \Delta t)$ can be computed using EQ (1.5).

4. Finally, the angular position and velocity are computed after the whole time step from the angular acceleration $\left(\frac{d\omega^b}{dt}\right)(t + \Delta t)$ and the predicted angular velocity $\omega^b(t + \Delta t)$ respectively.

Chapter 2

Implementation of an accurate static contact force model

This section is dedicated to the implementation of a new tangential contact force in our code *GRAINS3D*. We will show why this force is needed to better describe the dynamics of a granular medium, then we will present the algebraic expression of this force that stores information about the contact at previous time steps $-$ hence its name *memory*-friction force or *history*-friction force $-$, we will later show how this force has been implemented in the code *GRAINS3D* and we will present and discuss our validation tests.

2.1 Motivations

2.1.1 Current range of validity

In section 1.2.2, we have presented the contact model that was implemented in *GRAINS-3D*. In this model, the tangential force is a viscous friction proportional to the tangential velocity U_t. This model has been experimentally validated numerous times for systems which particles have a non-zero kinetic energy during the entire simulation, for example in FIG. 2.1 from the work of Rakotonirina et al. [20]. However, when simulating systems of particles with low kinetics energy such as packings, the code *GRAINS3D* presents a porosity that slowly decreases with time, as shown in FIG. 2.2. This unphysical behavior is very restricting because one has to make sure the problem is studied on sufficiently short time scales such that this unphysical overpacking phenomenon is neglectible.

This unphysical behavior can be intuited by the fact that when particles are packed and have a kinetic energy close to zero, they experience no tangential friction as $\overrightarrow{F}_{t,visq} = -2\gamma_t m_{ij}\overrightarrow{U_t} = 0$. It means that on long time scales, particles have a small tangential velocity that eventually overpacks the system. A solution would be to introduce a new tangential friction force which does not depend on the velocity of the particles. In section 2.2, we will present a force that is designed to meet this requirement, and that is already a standard in DEM simulations. Bringing *GRAINS3D* to the state of the art level of DEM codes by extending its validity to quasi-static and static systems is the objective of this implementation.

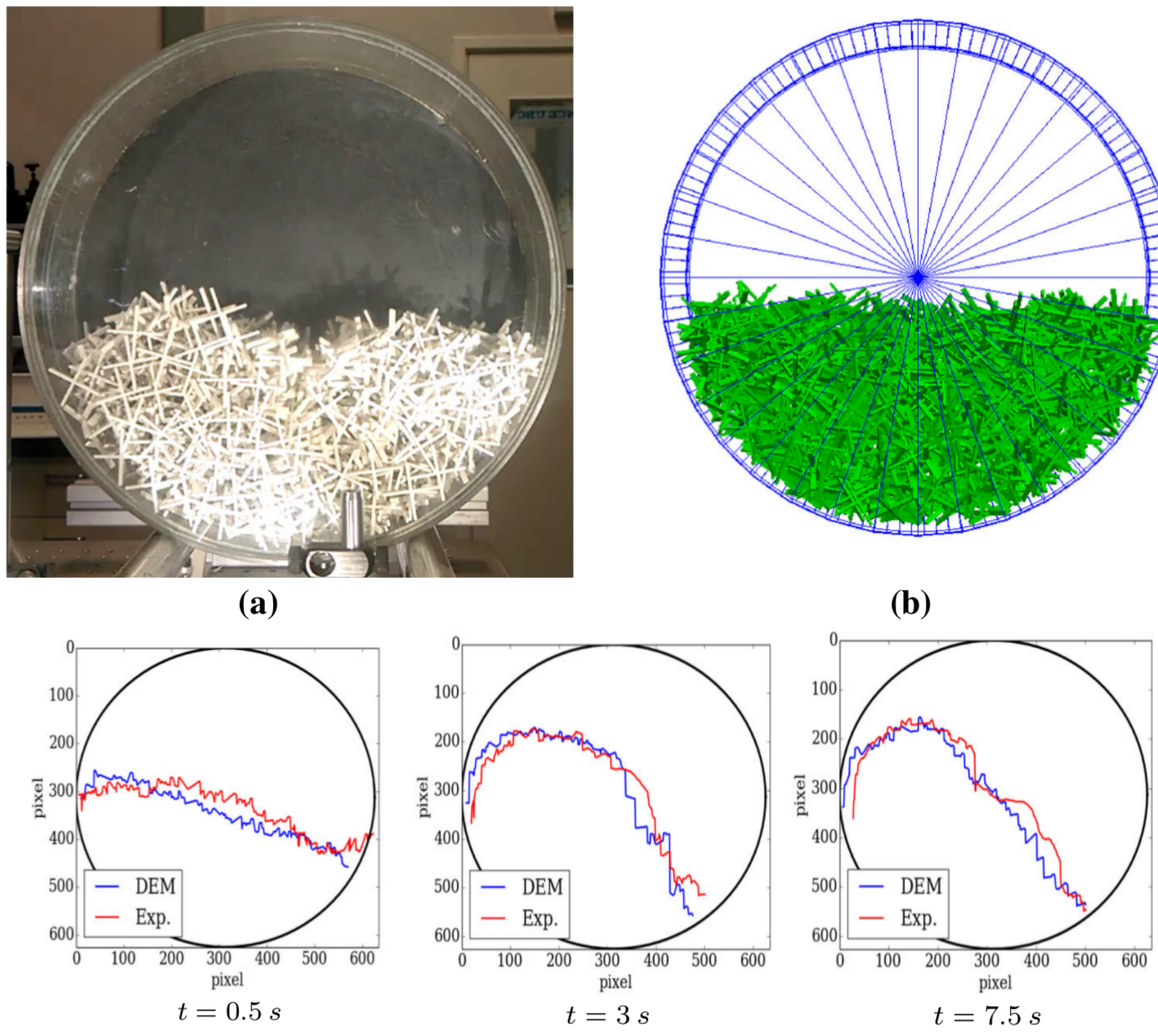
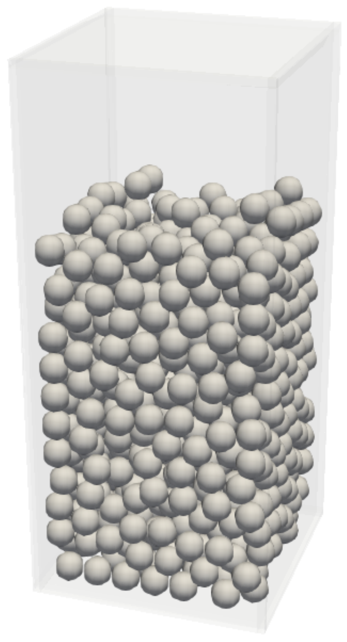
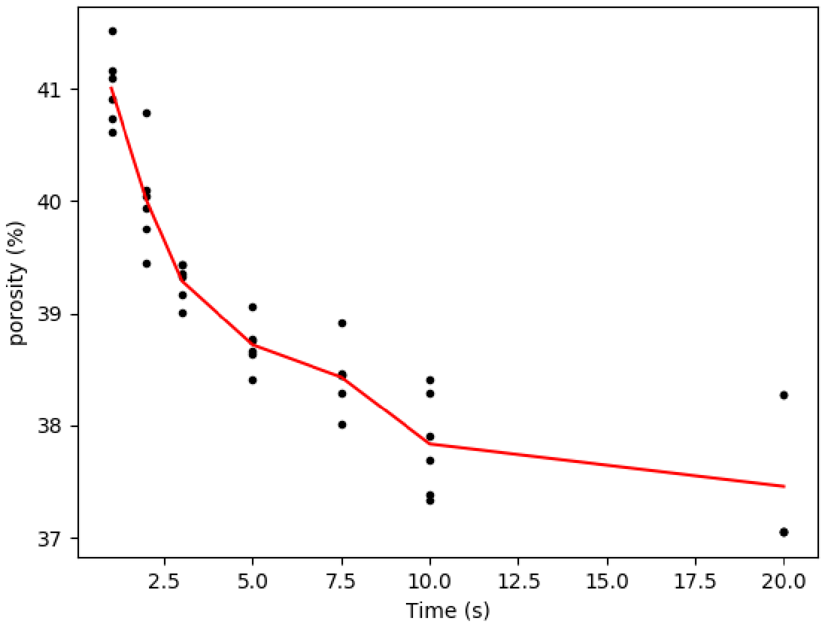


Figure 2.1: Validation experiment of *GRAINS3D*: dynamic simulation of non-convex particles in a rotating drum [20]. The three graphs at the bottom show the experimental and numerical free surfaces at different times, and show a satisfying accordance between experiments and simulation.


(a)



(b)

Figure 2.2: Numerical investigation of porosity versus time for the packing of 750 spheres, using our code *GRAINS3D*. On the plot (b), black dots are porosity values for single randomly loaded simulations, while the red curve links the means of these different porosity values. Parameters of the contacts are expressed in Table 2.1.

2.1.2 Methodology

In the following, we will use porosity analysis similar to the one in FIG. 2.2 to compare our implementations. We shall briefly show here how a porosity analysis is conducted.

The way *GRAINS3D* post-treats data to compute porosity is the following. The user enters the space region of interest (i.e. where there are particles) and the desired mesh refinement. Then, space is meshed in cubic cells which size is significantly smaller than the characteristic length of the particles – one can think of it as a 3D pixel. Each cell is then considered "full" if it lies inside a particle, "empty" otherwise. Once every cell has been labeled, the approximated porosity is the ratio of empty cells over the total number of cells. In our case, the mesh refinement has been chosen to be such that one cell size is $\frac{1}{40^{\text{th}}}$ of the characteristic size of our particles – diameter for spheres. Indeed, we have investigated that choosing a smaller cell size only impacts the porosity by less than 0.001%. It is a very acceptable accuracy considering that the porosity range of our analysis was over 5%.

Moreover, a porosity analysis can be tainted by an error due to a wall effect. As the walls of the containers are flat, it affects the nearby packing of particles and porosity in the vicinity of walls should not be considered. Indeed, close to the walls porosity is not relevant as it can be too high (in the case of spheres against a wall) or too low (in the case of cubes against a wall). Therefore, in all of our porosity analysis, the region of interest – i.e. the region in which the porosity is computed – is chosen to be away from the walls by one particle's characteristic length – for spheres, one diameter.

Finally, porosity experiments and simulations show variability with respect to the loading process. In an attempt to limit its importance, each simulation is performed several times with different loading configurations. Each loading is conducted randomly in the sense that particles appear at the top of the domain with random initial positions, providing they do not collide. For most cases, each simulation is perfomed six times, but this number is reduced to three for the 20 seconds long simulation due to high computation time (several days). In FIG. 2.2b, FIG. 2.5 and FIG. 2.6, each black dot corresponds to the porosity computed from a single random loading of particles. The red line connects the mean values of the black dots and is a more reliable way to analyze the behavior of porosity.

2.2 A new tangential friction force

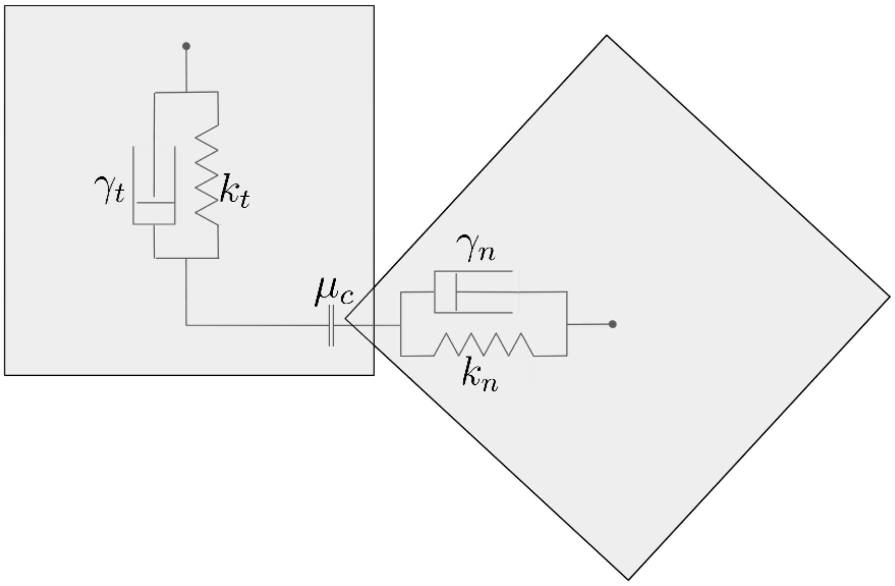


Figure 2.3: The contact model for the tangential force with memory effect.

In order to keep a non-zero friction force, we introduce a Hookean force which coefficient is proportional to the cumulative displacement $\overrightarrow{\delta(t)} = \int_{t_{contact}}^{t} \overrightarrow{U_t(s)} ds$. It is important to note that the cumulative displacement $\delta(t)$ is not related to the penetration depth δ. Still, we keep this unfortunate notation choice as it is commonly used in the literature. Then, we define the new tangential Hookean force as follows:

$$\overrightarrow{F_t} = \overrightarrow{F}_{t,visq} + \overrightarrow{F}_{t,mem} \quad \text{with} \quad \overrightarrow{F}_{t,mem} = k_t \overrightarrow{\delta(t)} = k_t \int_{t_{contact}}^{t} \overrightarrow{U_t(s)} ds \tag{2.1}$$

From the expression of $\overrightarrow{F}_{t,mem}$ we understand its name *memory*-friction force, as it is proportional to the cumulative displacement $\overrightarrow{\delta(t)}$. Even though the memory-friction force may seem artificial, it turns out it tackles the problem of zero friction when particles have small velocities. It has therefore been widely adopted by the DEM community and has become a standard [18, 21, 5, 14].

2.3 Implementation

The memory-friction force supposes each particle keeps track of its neighbors and of the cumulative displacement $\overrightarrow{\delta(t)}$. This has been implemented with a key-value structure, using the C++ standard library `std::map` object.

A key-value structure allows the storage of data that can be uniquely accessed by a key. It follows the principle of a dictionnary : the user accesses a definition (i.e. the data or the value) using the word corresponding to the definition (i.e. the key). In our case, each particle stores an `std::map` object where the keys are the identification numbers – or ids – of its neighbors, and the values are the corresponding contact information (i.e. the cumulative tangential displacements). This is graphically shown in FIG. 2.4.

We note that this implementation is not memory-efficient as the cumulative tangential displacement is stored twice: once in each particle's map structure. However, in DEM the challenge is about computation rather than storage. In fact, our code *GRAINS3D* does not suffer from storage issues, so the use of non memory-efficient data structures is allowed as long as they offer low computation costs.

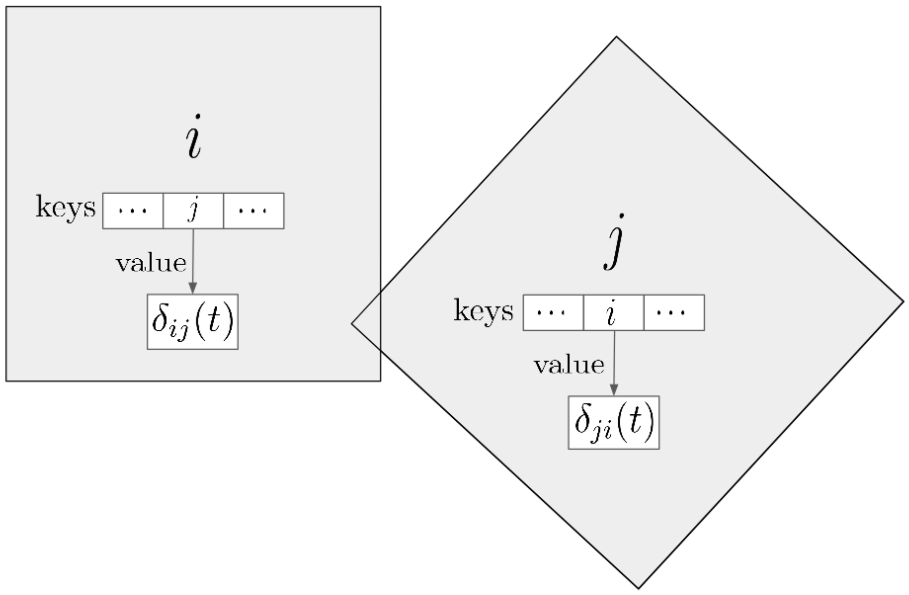


Figure 2.4: Diagram of the use of the C++ object `std::map` implementing the storage of contact information between two particles i and j. The keys of particle i (resp. particle j) are its neighbors' identification number while the values associated with the keys are the contact information to store.

2.4 Validation

2.4.1 Validation on porosity experiments

As a validation procedure, we use porosity experiments previously presented in section 2.1.2. We consider that our implementation is validated if the porosity decay with time is largely reduced. As we will see later, there are probably better ways to assess the validity of our implementation, but we will discuss this issue further in section 2.4.2.

Investigating the tangential hookean coefficient

The input parameters of our simulations are chosen to be the same as previous numerical experiments conducted by Wachs et al. [12], and are summarized in Table 2.1. The only parameter that is missing is the tangential hookean coefficient k_t of the new tangential force. Before conducting our porosity versus time numerical experiment, we shall investigate the value of k_t.

Table 2.1: Parameters of the spherical particles used to conduct numerical porosity analysis.

Parameter	Value
Radius (m)	$1 \cdot 10^{-3}$
k_n $(\mathrm{N} \cdot \mathrm{m}^{-1})$	$4 \cdot 10^{5}$
γ_n (s^{-1})	$2.623 \cdot 10^{3}$
μ_c (no unit)	0.5
γ_t (s^{-1})	$1 \cdot 10^{5}$

To do so, we analyze the porosity of a packing of spheres on a wide range of values for k_t – from 10^1 to 10^7 N·m^{-1}. As porosity increases with inter-particle friction, considering the effect of k_t on porosity is a way to consider the effect of k_t on the effective friction felt by the particles. As we can see in FIG. 2.5, the porosity increases somewhat like a sigmoid and reaches a plateau for values of k_t greater than 10^5 N·m^{-1}. Keeping in mind that the physical tangential friction is the Coulomb friction, and that all of the contact models of smooth DEM are only introduced for numerical stability, the value of k_t should be chosen so that the Coulomb friction – achieved by high tangential friction – is reached easily. Therefore, we need to choose a large value for k_t, but a value unreasonably large would lead to numerical instabilities. A trade-off is to take the lowest value of k_t that achieves a high effective friction for the particles, that is to say that maximizes the porosity in FIG. 2.5. In the following, the value chosen for the tangential hookean coefficient k_t is 10^5 N·m^{-1}.

Validation with porosity versus time profiles

With the value of k_t now determined, we can proceed to the porosity versus time numerical experiments. The results are shown in FIG. 2.6. If we compare this result to the one obtained without the memory-friction force in FIG. 2.2, the behavior is rather disappointing as the new friction force does not lower the porosity decay.

2.4.2 Discussion

With the numerical results of FIG. 2.6, we need to understand why implementing this memory-friction force has not improved the accuracy of our porosity simulations. It

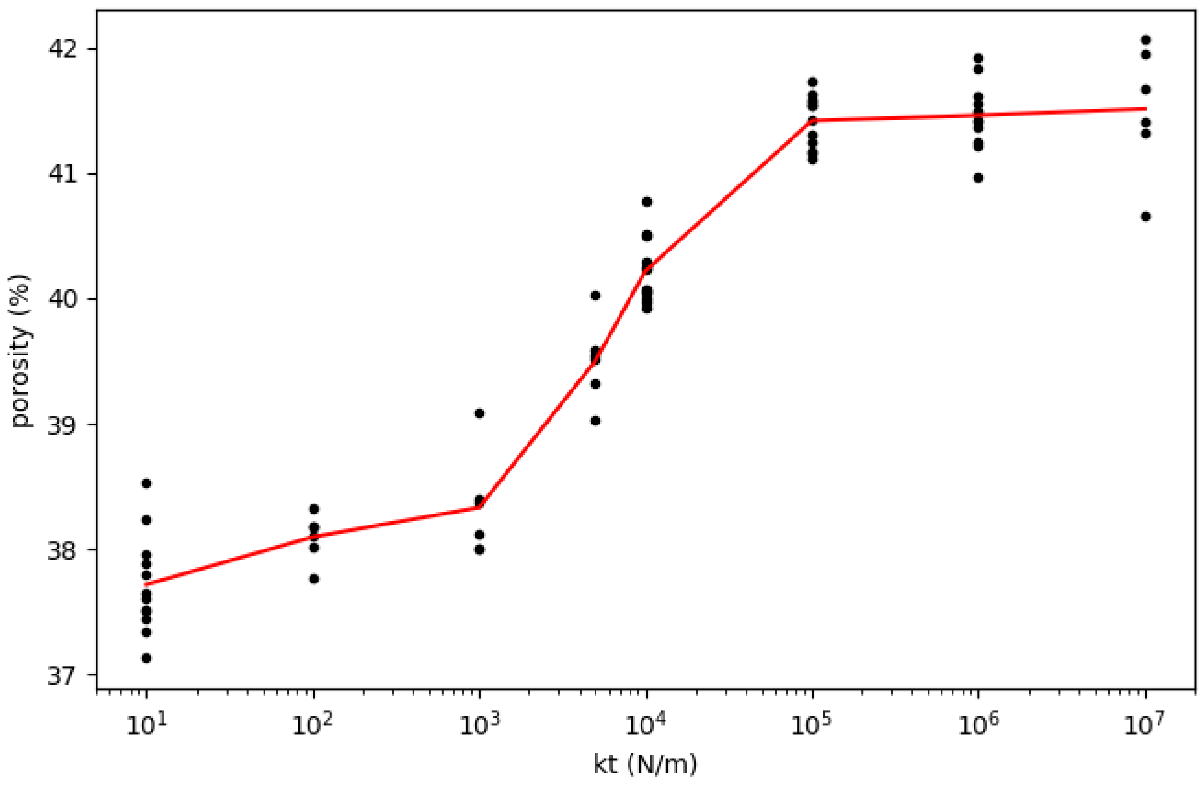


Figure 2.5: Porosity of the packing of 750 spheres with respect to the tangential hookean coefficient k_t. Parameters of the contacts are expressed in Table 2.1, except for γ_t which was taken equal to zero. Simulation time is 1.0 s.

turns out that the tangential contact force $\overrightarrow{F_t}$ is not the only variable at play in this decay of porosity. In fact, Ai et al. [9] investigate the effect of various rolling friction models for spherical DEM. They show that even with this memory friction force we have implemented in *GRAINS3D*, some commonly used rolling friction models lead to a small residual kinetic energy that tends to make the granular medium vibrate. It is well known that a way to pack a granular medium is to make it undergo vibrations. FIG. 2.7a shows that the rolling friction model implemented in *GRAINS3D* – referred to as the directional constant torque model (model A) in Ai's work – leads to these unphysical and unwanted vibrations, eventually leading to an overpacking of the system as shown in FIG. 2.7b.

To the best of our knowledge, there is no test that would validate our implementation through a macroscopic property such as porosity, without being tainted by other variables at play such as the rolling friction force. Therefore, in the light of the work of Ai et al. [9], validating our implementation through porosity analysis would require to implement a new, more accurate rolling friction model for *GRAINS3D*. This would go beyond the scope of this project, but it will have to be done in the near future in order to allow the use of the newly implemented memory-friction force in *GRAINS3D*. However, for this project we would also like to focus on numerical integration for DEM.

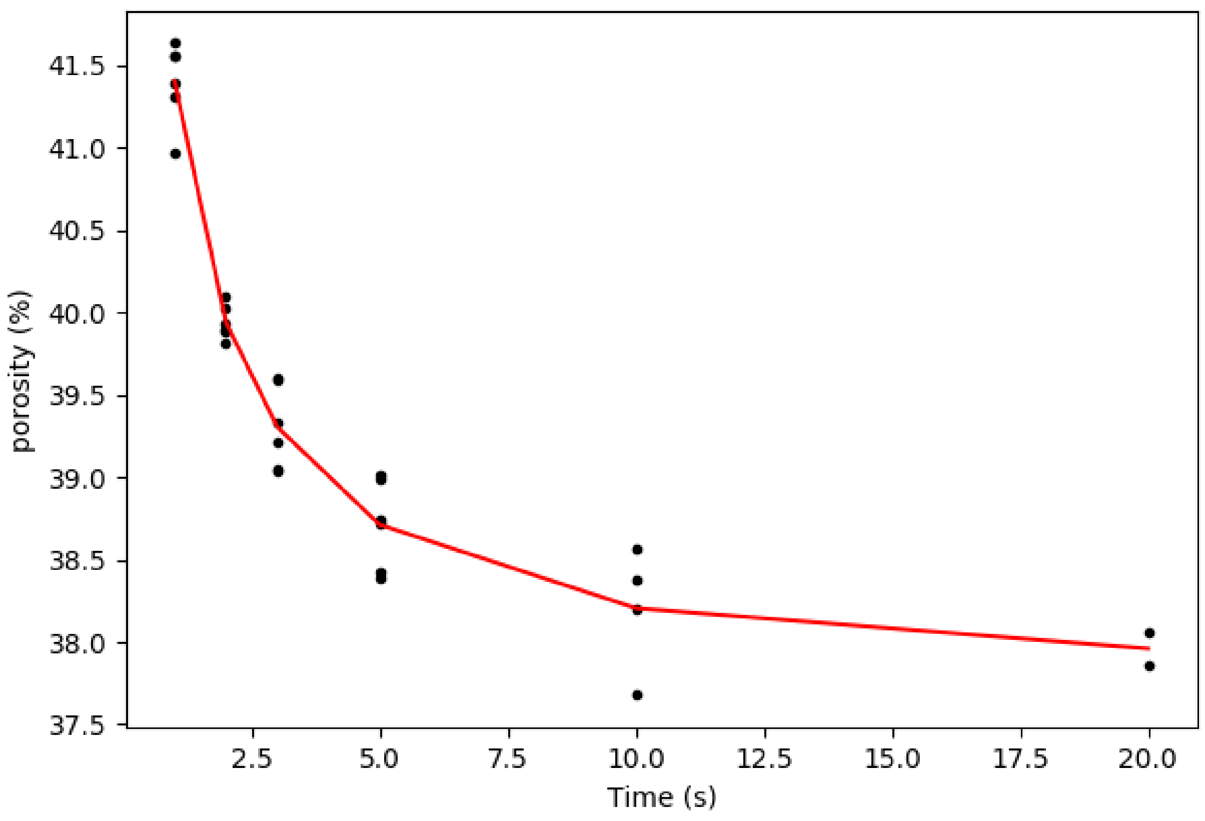
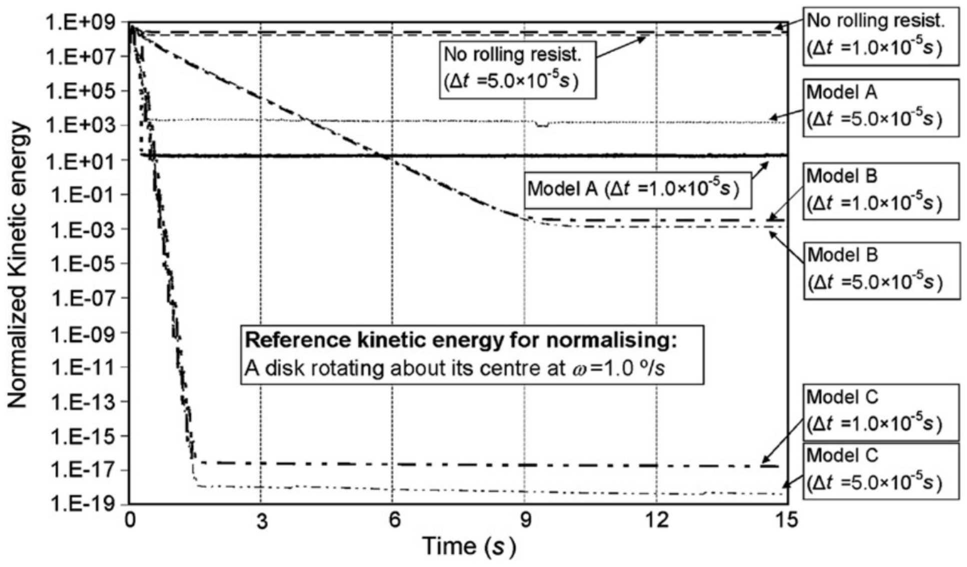
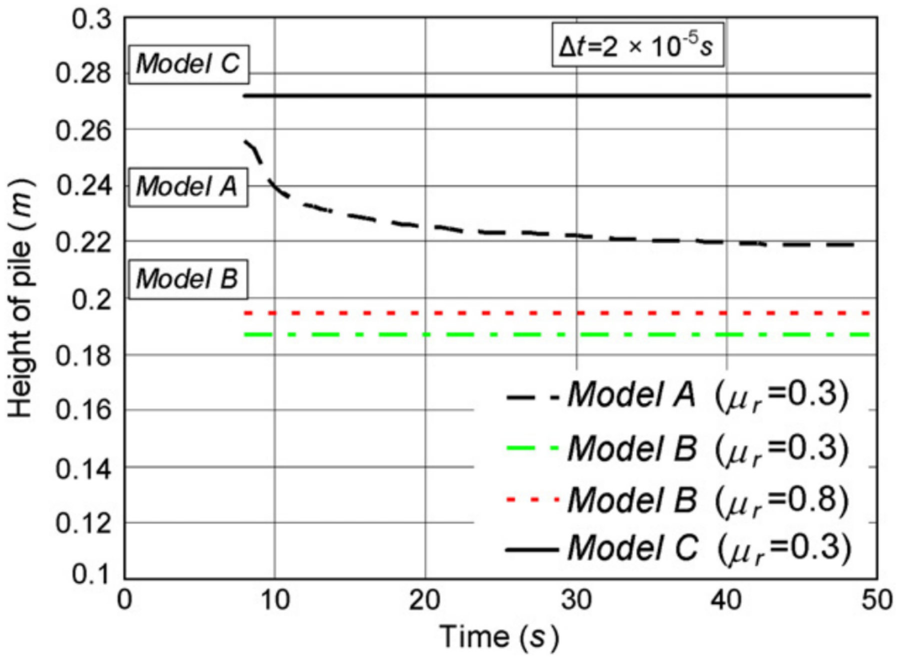


Figure 2.6: Porosity of the packing of 750 spheres with respect to time, with the contribution of the new tangential hookean force. Parameters of the contacts are expressed in Table 2.1, and k_t is taken equal to 10^5 N·m^{-1}.



(a)



(b)

Figure 2.7: The effect on rolling friction models on the residual kinetic energy and the packing of the system. Figures from Ai et al [9].

Chapter 3

Higher order integration schemes

In this chapter, we investigate the implementation of higher order schemes in DEM. We choose an Adams-Bashforth scheme of third order, coupled with a second-order Runge-Kutta for the first steps. We shortly present the new schemes and motivations for a first Python implementation in section 3.1. Then, we conduct validation simulations in section 3.2 and investigate the error scaling with respect to analytical solutions for the position and the restitution coefficient respectively. As we shall see, for some tests the error scaling is not what one would expect for a third-order scheme. Section 3.3 is dedicated to discuss these results and to show they come from theoretical matters rather than implementation ones.

3.1 Considered schemes and implementation

3.1.1 Considered integration schemes

The third-order Adams-Bashforth scheme is the following:

$$\left\{ \begin{array}{l} x(t+\Delta t) = x(t) + \frac{\Delta t}{12}(23U(t) - 16U(t-\Delta t) + 5U(t-2\Delta t)) \\ U(t+\Delta t) = U(t) + \frac{\Delta t}{12}(23a(t) - 16a(t-\Delta t) + 5a(t-2\Delta t)) \end{array} \right. \tag{3.1}$$

As a three-step algorithm, the first two steps need to be computed by another one-step scheme. For this purpose, we implement a second-order Runge-Kutta scheme.

3.1.2 Implementation with a high-level language

In large low-level programming projects, if possible it is often a good idea to implement new features in a different, smaller project programmed with a high-level language such as Python, Matlab or Octave. It gives flexibility to the programmer to test the initial implementation and apply corrections if necessary, before putting time and effort in implementing the definitive feature in the main code.

In our case, as we shall see in section 3.2, the numerical scheme validation requires only one spherical particle hitting a wall. Therefore, it becomes very easy to program our own high-level language DEM code, specifically designed to try different numerical schemes. In the following, the schemes are programmed using the language Python. For data computed from *GRAINS3D*, see APPENDIX. C.

3.2 Validation

3.2.1 Position error

Our validation test consists of the computation of the impact of a single spherical particle hitting a wall at a constant speed, with no gravity. First, we will compare the position of the sphere of the simulated solution to the analytical solution. In other words, we compare the computed penetration depth to an analytical solution δ, which derivation is shown below.

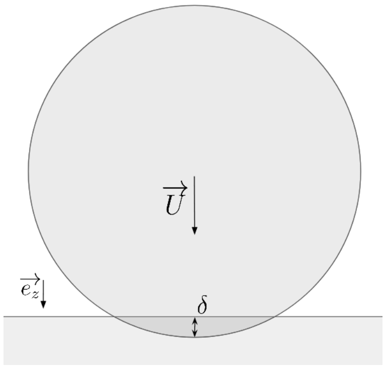


Figure 3.1: Schematic view of the validation simulation: a single particle hits a plane, and the error on δ is analysed.

Analytical solution

To derive the penetration depth δ, we solve the Newton's laws of motions during contact. Only the normal forces are to be considered, that is:

- $\overrightarrow{F}_{n,Hooke} = -k_n \delta \overrightarrow{e_z}$
- $\overrightarrow{F}_{n,visq} = -2\gamma_n M \overrightarrow{U} = -2\gamma_n M \frac{d\delta}{dt}\overrightarrow{ez}$

with k_n the material stiffness, γ_n the normal viscous factor, M the mass of the particle, $\overrightarrow{U}$ its velocity and $\overrightarrow{e_z}$ the unit vector in the vertical (and normal) direction – see FIG. 3.1. Applying Newton's laws of motion along $\overrightarrow{e_z}$ gives us:

$$\frac{d^2\delta}{dt^2} + 2\gamma_n \frac{d\delta}{dt} + \omega_0^2 \delta = 0, \quad \text{with} \quad \omega_0 = \sqrt{\frac{k_n}{M}} \tag{3.2}$$

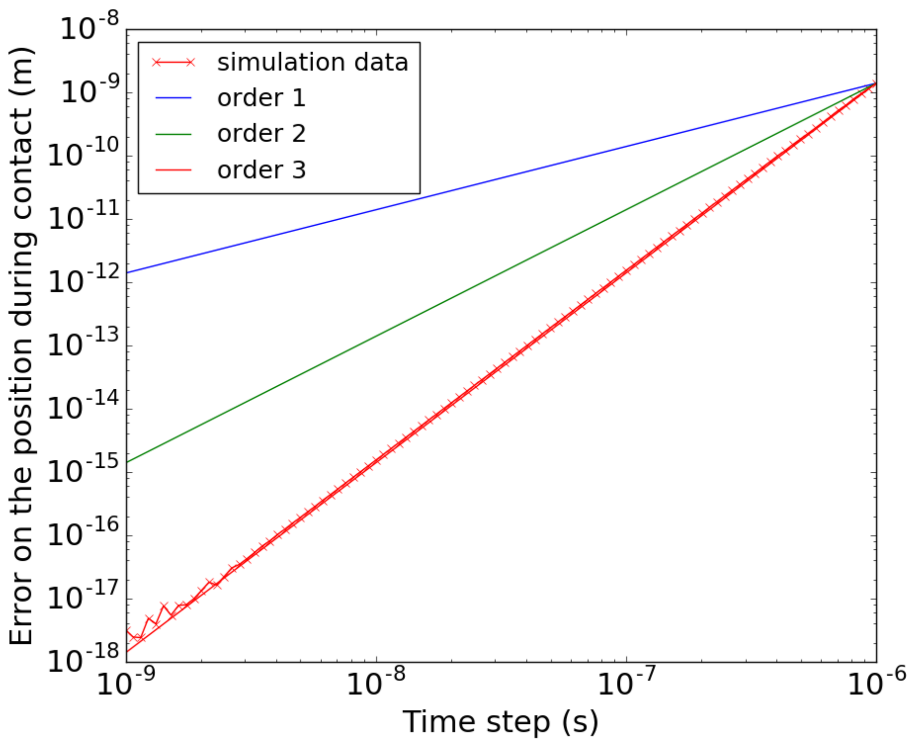
and the analytical value δ of the penetration depth is therefore:

$$\delta(t) = \frac{U_0}{\omega} e^{-\gamma_n (t-t_c)} \sin\left(\omega(t-t_c)\right) \tag{3.3}$$

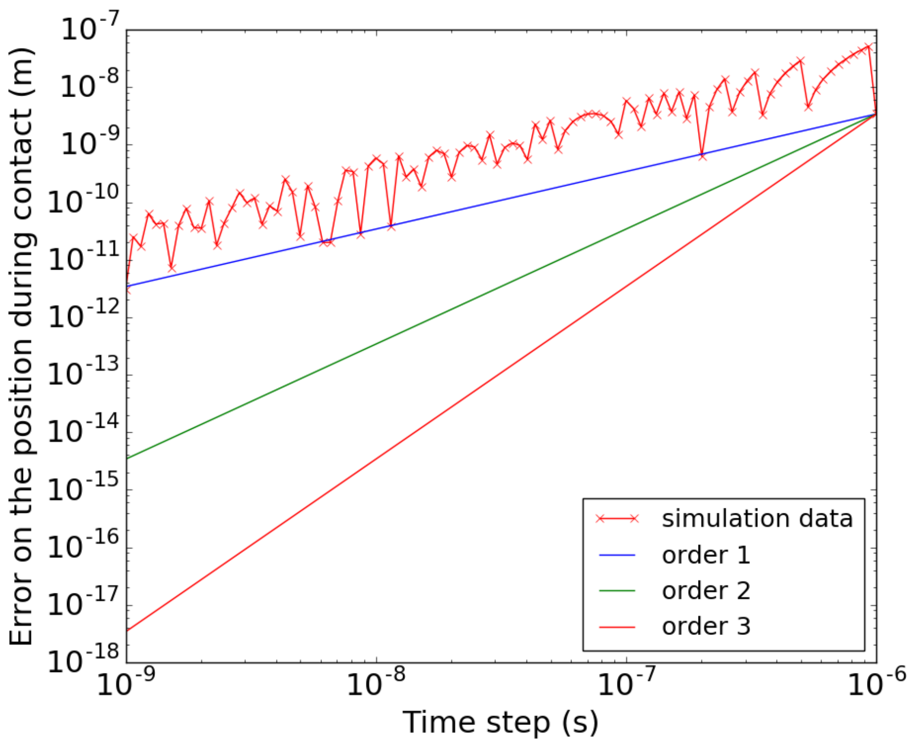
with t_c the time of the beginning of the contact, U_0 the initial velocity norm $\|\overrightarrow{U}(t \leq t_c)\|$ and $\omega = \sqrt{\omega_0^2 - \gamma_n^2}$.

Results

The scaling of our third-order Adams-Bashforth scheme with respect to the position error is shown in FIG. 3.2. Two test-cases are considered: (i) at $t = 0$ the particle already touches the wall and is about to penetrate into it with the initial velocity $\overrightarrow{U}$, and (ii) at $t = 0$ the particle is slightly above the wall – by 10^{-7}m to compare with the particle's radius of 10^{-3}m – and is about to hit the plane with the same velocity $\overrightarrow{U}$. We observe in FIG. 3.2a that the first case leads to the expected third order scaling: the error of the computed position with respect to the analytical position decreases with Δt^3. However, if the sphere does not start on the wall but slightly above, FIG. 3.2b shows an error behavior that is somewhat similar to a noisy first order.



(a) Initial position on the plane



(b) Initial position above the plane

Figure 3.2: Position error scaling of a third order Adams-Bashforth scheme for the rebound problem, with the particle starting on the surface of the plane (a), and above the surface of the plane (b).

APPENDIX. B presents similar results for a variety of other schemes, among them the second-order Adams-Bashforth scheme and the fourth-order Runge-Kutta. This unexpected order decline will be discussed in section 3.3. APPENDIX. C presents the results for a position error test similar to the one on FIG. 3.2a, computed from our code *GRAINS3D* with the newly implemented third order Adams-Bashforth scheme, initiated by the second order Runge-Kutta scheme.

3.2.2 Restitution coefficient error

With the same case test of a sphere hitting a wall in a no-gravity environment, we can also consider the error on other variables than position. For example, the restitution coefficient is particularly easy to analyze as it is simply the ratio of the post-impact velocity U_f over the initial velocity U_0. However, with the error on the restitution coefficient none of the cases where (i) the sphere starts on the wall and (ii) the sphere starts above the wall show a third order error scaling. In FIG. 3.3 we see that they both lead to a noisy first-order tendancy. The results for the same test case applied to other numerical schemes can be found in APPENDIX. B.

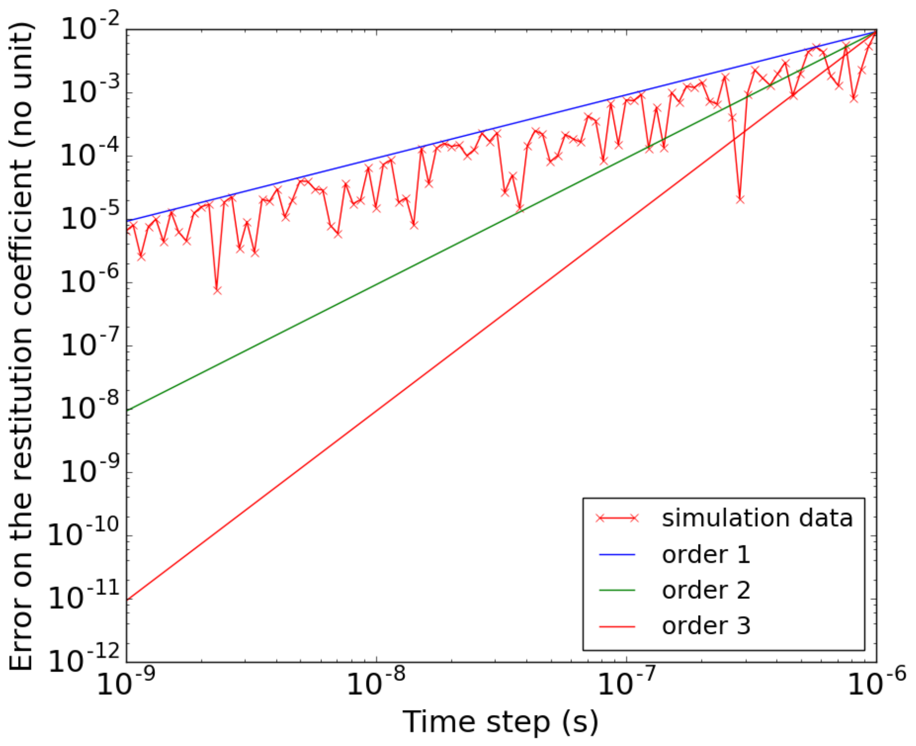
3.3 Discussion

The above error scalings were not expected and are quite uncomfortable because they involve an order degeneration which seems to be case- and error type-specific. As our numerical schemes lead to expected performances in some cases $-$ in the case where there is contact at $t = 0$ for the position error $-$ it clearly does not come from implementation mistakes for all of the schemes tested. In this section we explain why we find a first order tendancy in some configurations and for some errors, and we compare the performances to other DEM methods.

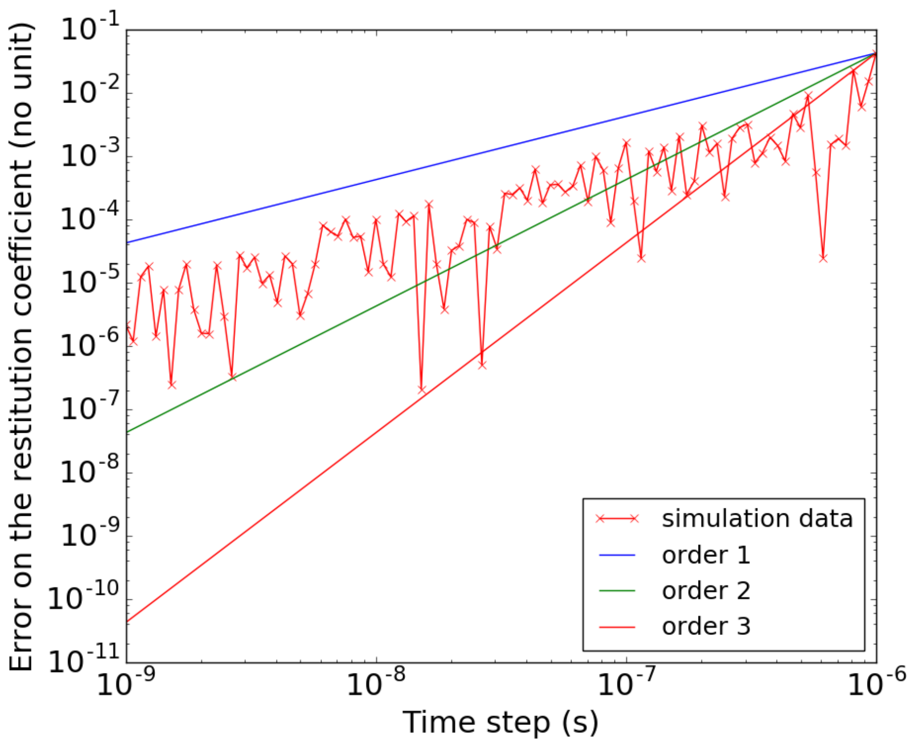
3.3.1 A time discretization inherent error

It turns out that this order degeneration is intrinsically linked to the smooth DEM method, and is unfortunately poorly documented in the literature. Kruggel-Emden briefly mentions it in his performance review of integrators for DEM [8], but does not specify the order depreciations that it can induce.

When a collision occurs, there is an error that is linked to the detection latency of the contact. As shown in FIG. 3.4, when the position of a particle $\mathcal{A}$ is updated during free-fall and when its new position implies a contact with a wall or another particle, during the intermediate time t_{contact} and the next time step $t + \Delta t$ particle $\mathcal{A}$ does not feel the effects of its contact with the obstacle. As a result, at the next time step $t + \Delta t$ its penetration depth and velocity are higher than if the contact was to be detected and taken into account exactly at t_{contact}. The results obtained previously show that this time-step induced error is of order one and dominates the integration error. The noise observed in FIG. 3.2 and FIG. 3.3 is due to the fact that this time-step error depends on the latency of the contact detection. Indeed, sometimes the time discretization leads to a very low latency $-$ when $t + \Delta t$ is exactly on or right after the analytical contact time t_{contact} $-$, involving a groove in the error scaling profile; while sometimes the latency is



(a) Initial position on the wall



(b) Initial position above the wall

Figure 3.3: Velocity error scaling of a third order Adams-Bashforth scheme for the rebound problem, with the particle starting on the surface of the wall (a), and above the surface of the wall (b).

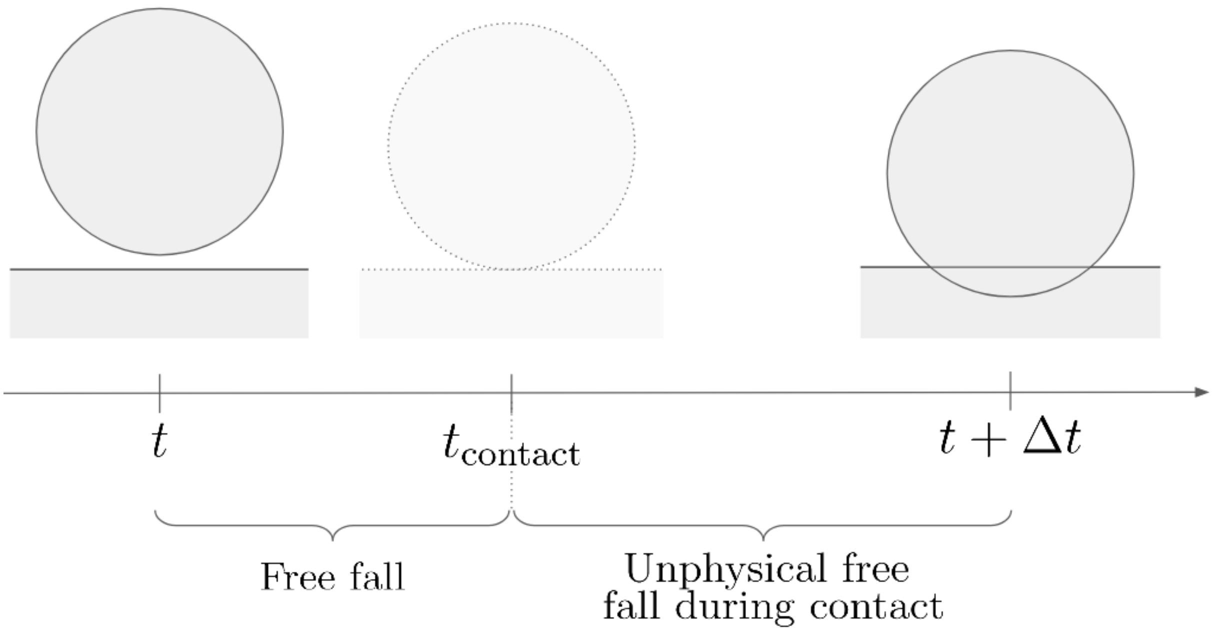


Figure 3.4: Schematic view of the time-discretization induced error during contact detection.

high – when the contact detection time $t + \Delta t$ is further away from t_{contact} –, involving a bump on the curve. The reason why neither FIG. 3.3a nor FIG. 3.3b show a third order accuracy is that in both cases the final velocity U_f is tainted by this detection error, that occurs at the end of the contact. The post-contact free fall is detected too late and leads to an error on U_f, tainting the restitution coefficient $e_n = \frac{U_f}{U_0}$ with an error. This is also the case in FIG. 3.3a even with an exact value of U_0.

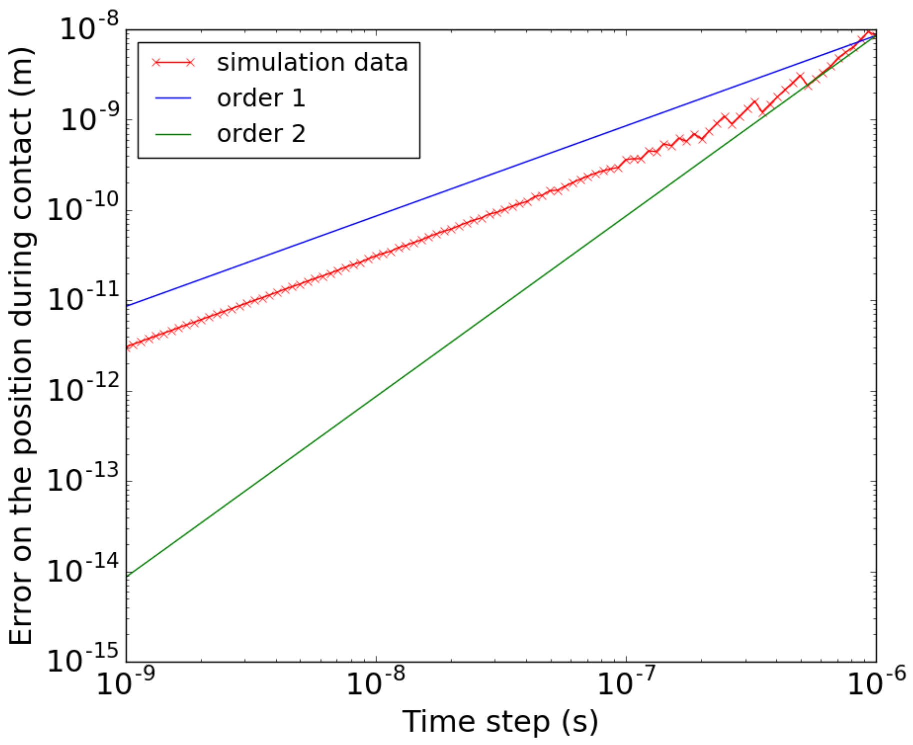
A way to validate this explanation would be to fit the analytical solution δ in EQ (3.3) so that it matches the particle's position and velocity at the time the contact is detected. That is, if U_0 and t_c in EQ (3.3) are chosen to lead to an accordance between the analytical solution δ (resp. $\frac{d\delta}{dt}$) and the particle's position (resp. velocity) at the time of the contact detection; does it lead to an error scaling of the expected order? FIG. 3.5b shows that it is the case for a second order Runge-Kutta scheme.

Interestingly, the error scaling is still of order one in the case of a second order Adams-Bashforth. In fact, APPENDIX. B presents the data from other schemes, and we find all one-step schemes show error scalings of the expected orders, while multi-step schemes do not. Indeed, while one-step algorithms "forget" the data from the previous times, multi-steps algorithms re-use it. However, in this case of a fitted analytical solution, the position and velocities at the step previous to the contact are not consistent with the position and velocity's values after the contact. When multi-step algorithms use this inconsistent data, it leads to a first order error that propagates to further time steps and decreases the global order of the scheme.

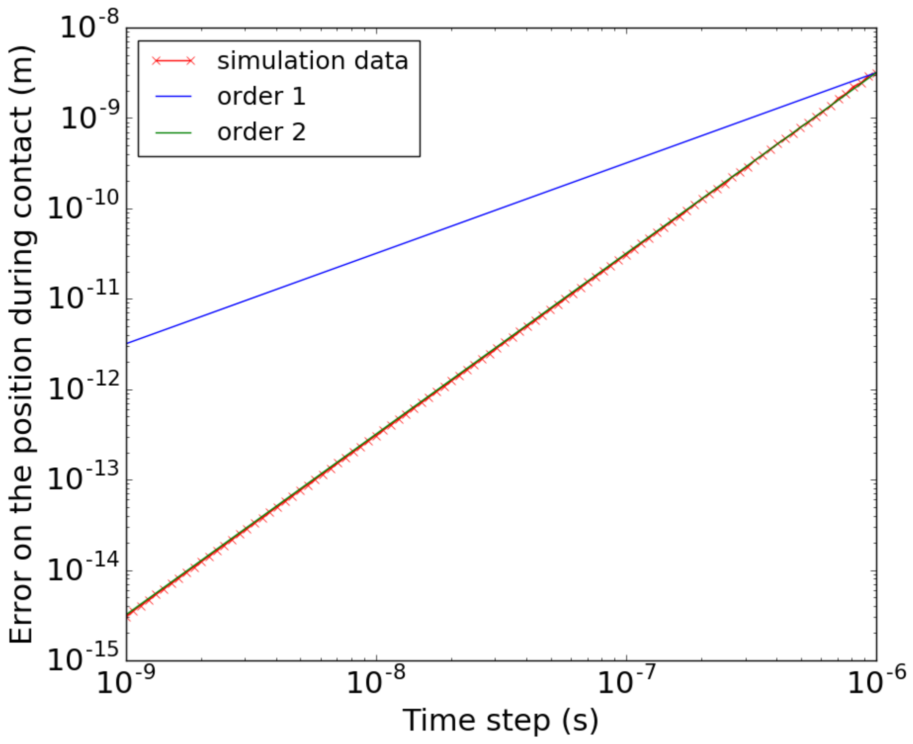
3.3.2 Achieving a global high order in DEM?

In the light of the time discretization inherent error that has just been described, it is legitimate to ask if a DEM simulation can be of high order at all. In fact, very few error versus time step curves can be found in the literature. One could think of abandonning the soft sphere model for the hard-sphere model [4], but this latter model cannot achieve large amounts of particles, which makes it unapplicable to the typically large systems simulated by the smooth DEM community.

A promising approach has been recently proposed by Kempe & Fröhlich [10]. Although their study is focused on collision of spheres in fluids, their Adaptative Collision Time Model (ATCM) is applicable for dry collisions. In this model, the stiffness and damping coefficients are no longer material constants but are computed for each collision using an optimization procedure, so that the rebound velocity and the collision time match the physical expectations. Their comparisons on experimental data from Gondret et al. [7] look very encouraging. Implementing it in *GRAINS3D* is beyond the scope of this project, but is to be considered in the future.



(a) Second order Adams-Bashforth (two-step scheme)



(b) Second order Runge-Kutta (one step scheme)

Figure 3.5: Position error scaling of a second order Adams-Bashforth scheme (a) and a second order Runge Kutta scheme (b) for the rebound problem, with the particle starting above the surface of the wall and the analytical solution being fitted to match the particle's position and velocity at the instant the contact is detected.

Conclusion

A new contact force model was implemented in the DEM code *GRAINS3D*, as well as a third order Adams-Bashforth integration scheme. On the one hand, validation for the contact force model was not successful due to another phenomenon altering the validation tests. A more accurate rolling friction model should allow the validation of the implemented contact force model, and is to be carried out in the near future.

On the other hand, validation tests of the Adams-Bashforth integration scheme show a third-order behavior when the position is compared to the analytical solution during contact, as expected. However, an interesting behavior has been identified for all integration schemes. It occurs at the interface between the free fall and the contact problems. It is believed to be due to an error inherent to the time discretization, and was confirmed by the different behavior of one-step and multi-step algorithms in the case of an analytical solution fitting.

For furthur interests, methods to bypass the order decrease can be investigated, such as the Adaptative Collision Time Model. A focus point can be put on the computational performances of the code in order to keep up with the fast moving field of computer science. The most crucial aspects are probably the implementation of a dynamic load balancing and a hybrid OpenMP-MPI architecture. Finally, once the contact model has been validated in dry conditions, validation tests can be carried out for particles immersed in a fluid. Such particle-laden flows have tremendous applications in many industries and in academia.

Bibliography

[1] P.A. Cundall and O.D.L. Strack. “A discrete numerical model for granular assemblies”. In: *Geotechnique* 29.1 (1979), pp. 47–65.

[2] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. “A fast procedure for computing the distance between complex objects in three-dimensional space”. In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203.

[3] G. Van den Bergen. “A fast and robust GJK implementation for collision detection of convex objects”. In: *Journal of Graphics, Gpu, and Game Tools* 4 (1999), pp. 7–25.

[4] Michel Jean. “The non-smooth contact dynamics method”. In: *Computer methods in applied mechanics and engineering* 177.3-4 (1999), pp. 235–257.

[5] A. Dviugys and B. Peters. “An approach to simulate the motion of spherical and non-spherical fuel particles in combustion chambers”. In: *Granular Matter* 3.4 (2001), pp. 231–266.

[6] G. Van Den Bergen. “Proximity queries and penetration depth computation on 3d game objects”. In: *Game Developers Conference*. Vol. 170. Citeseer. 2001.

[7] P. Gondret, M. Lance, and L. Petit. “Bouncing motion of spherical particles in fluids”. In: *Physics of fluids* 14.2 (2002), pp. 643–652.

[8] H. Kruggel-Emden et al. “Selection of an appropriate time integration scheme for the discrete element method (DEM)”. In: *Computers & Chemical Engineering* 32.10 (2008), pp. 2263–2279.

[9] J. Ai et al. “Assessment of rolling resistance models in discrete element simulations”. In: *Powder Technology* 206.3 (2011), pp. 269–282.

[10] T. Kempe and J. Fröhlich. “Collision modelling for the interface-resolved simulation of spherical particles in viscous fluids”. In: *Journal of Fluid Mechanics* 709 (2012), pp. 445–489.

[11] Christoph Kloss et al. “Models, algorithms and validation for opensource DEM and CFD–DEM”. In: *Progress in Computational Fluid Dynamics, an International Journal* 12.2-3 (2012), pp. 140–152.

[12] A. Wachs et al. “Grains3D, a flexible DEM approach for particles of arbitrary convex shape - Part I: numerical model and validations”. In: *Powder Technology* 224 (2012), pp. 374–389.

[13] F Zhao and BGM van Wachem. “A novel quaternion integration approach for describing the behaviour of non-spherical particles”. In: *Acta Mechanica* 224.12 (2013), pp. 3091–3109.

[14] Pedro Costa et al. “Collision model for fully resolved simulations of flows laden with finite-size particles”. In: *Physical Review E* 92.5 (2015), p. 053012.

[15] Alessandro Tasora et al. “Chrono: An open source multi-physics dynamics engine”. In: *International Conference on High Performance Computing in Science and Engineering*. Springer. 2015, pp. 19–49.

[16] Jia Yan-Bin. *Quaternions and Rotations*. `https://pdfs.semanticscholar.org/a008/49dd6075976296c963112bf4f37940e243c9.pdf`. Accessed: 2018-09-30. 2015.

[17] LJH Seelen, JT Padding, and JAM Kuipers. “Improved quaternion-based integration scheme for rigid body motion”. In: *Acta Mechanica* 227.12 (2016), pp. 3381–3389.

[18] G. Lu, J.R. Third, and C.R. Müller. “The parameters governing the coefficient of dispersion of cubes in rotating cylinders”. In: *Granular Matter* 19.1 (2017), p. 12.

[19] A.D. Rakotonirina and A. Wachs. “Grains3D, a flexible DEM approach for particles of arbitrary convex shape-Part II: Parallel implementation and scalable performance”. In: *Powder Technology* 324 (2018), pp. 18–35.

[20] Andriarimina Daniel Rakotonirina et al. “Grains3D, a flexible DEM approach for particles of arbitrary convex shape?Part III: extension to non-convex particles modelled as glued convex particles”. In: *Computational Particle Mechanics* 5.4 (2018), pp. 1–30.

[21] L.J.H. Seelen, J.T. Padding, and J.A.M. Kuipers. “A granular Discrete Element Method for arbitrary convex particle shapes: method and packing generation”. In: *Chemical Engineering Science* 189 (2018), pp. 84–101.

Appendices

Appendix A

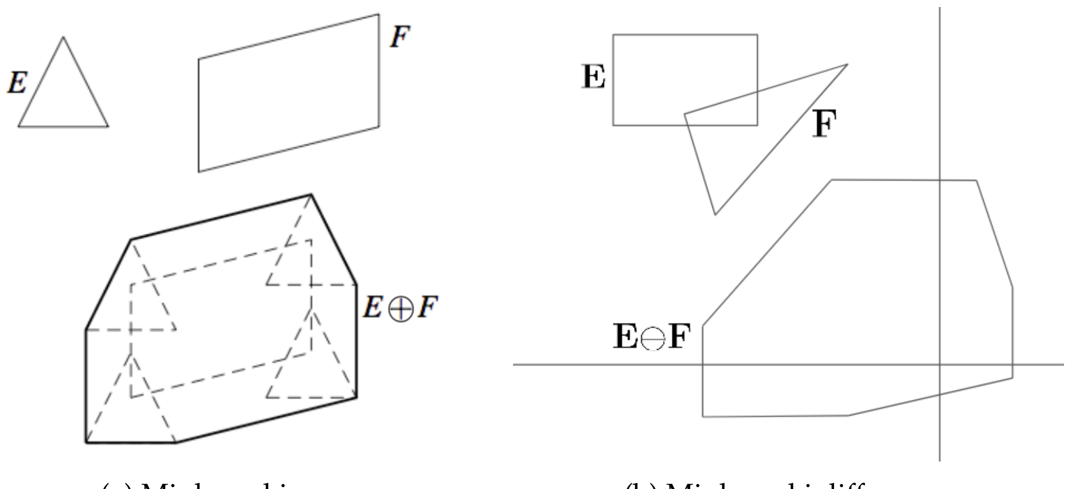
Key concepts for the GJK distance algorithm

Minkowski sum

The Minkowski sum is an operation on vector sets. A and B being two sets of vectors their Minkowski sum $A \oplus B$ is defined as:

$$A \oplus B := \{\mathbf{a} + \mathbf{b}, \quad \mathbf{a} \in A \text{ and } \mathbf{b} \in B\} \tag{A.1}$$

Similarly, $A \ominus B := \{\mathbf{a} - \mathbf{b}, \quad \mathbf{a} \in A \text{ and } \mathbf{b} \in B\}$. Geometric result of Minkowski operations are shown below in FIG. A.1:



(a) Minkowski sum (b) Minkowski difference

Figure A.1: Geometric example of Minkowski operations

Support mapping

The support mapping – or support function – $f_{\mathcal{C}}$ of a convex set $\mathcal{C}$ is defined as

$$f_{\mathcal{C}}(\mathbf{x}) := \sup\{\mathbf{x} \cdot \mathbf{c}, \quad \mathbf{c} \in \mathcal{C}\} \tag{A.2}$$

and is illustrated in FIG. A.2

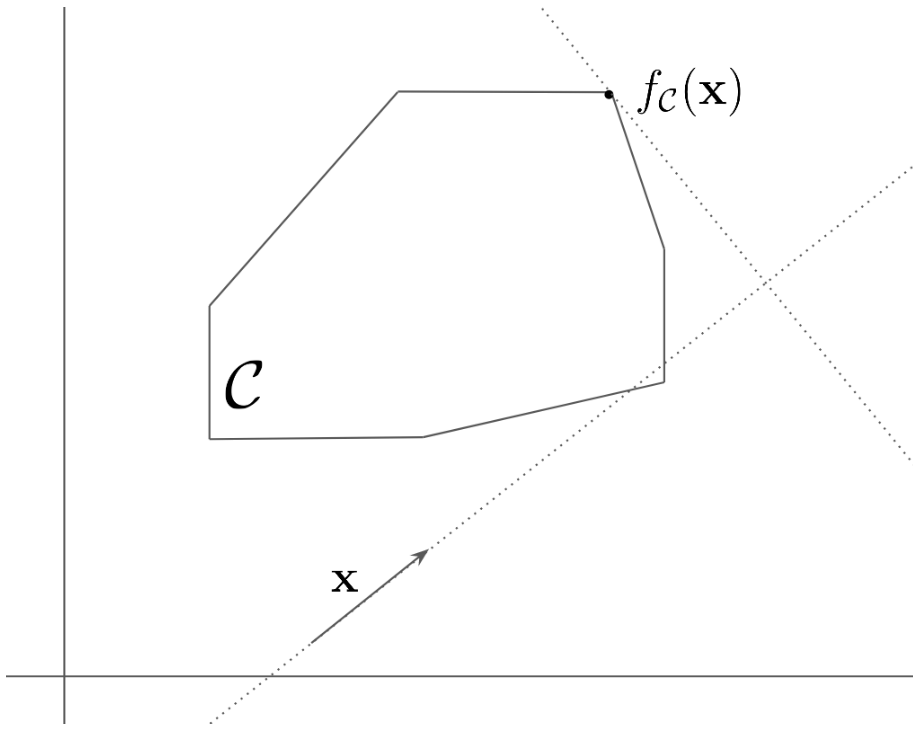


Figure A.2: Geometric example of a support mapping function

Simplex

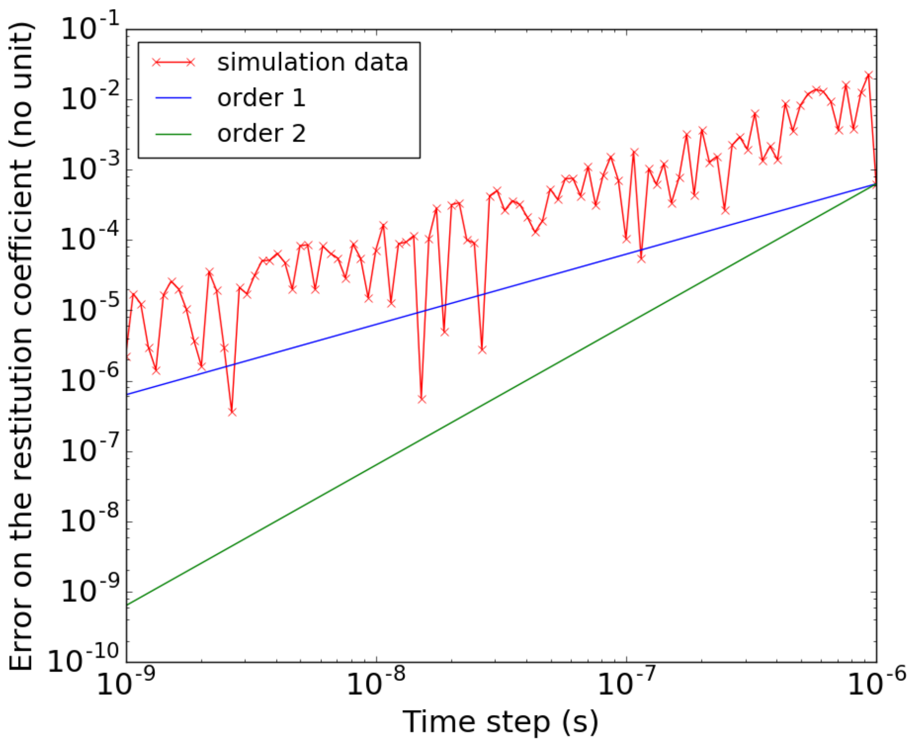
A k-simplex $\mathcal{S}_k$ is the convex hull of a set of k vertices $\{\mathbf{v}_i\}_{1\leq i\leq k}$:

$$S_k = \left\{ \sum_{i=0}^{k} a_i \mathbf{v}_i, \quad \text{with} \sum_{i=0}^{k} a_i = 1 \right\} \tag{A.3}$$

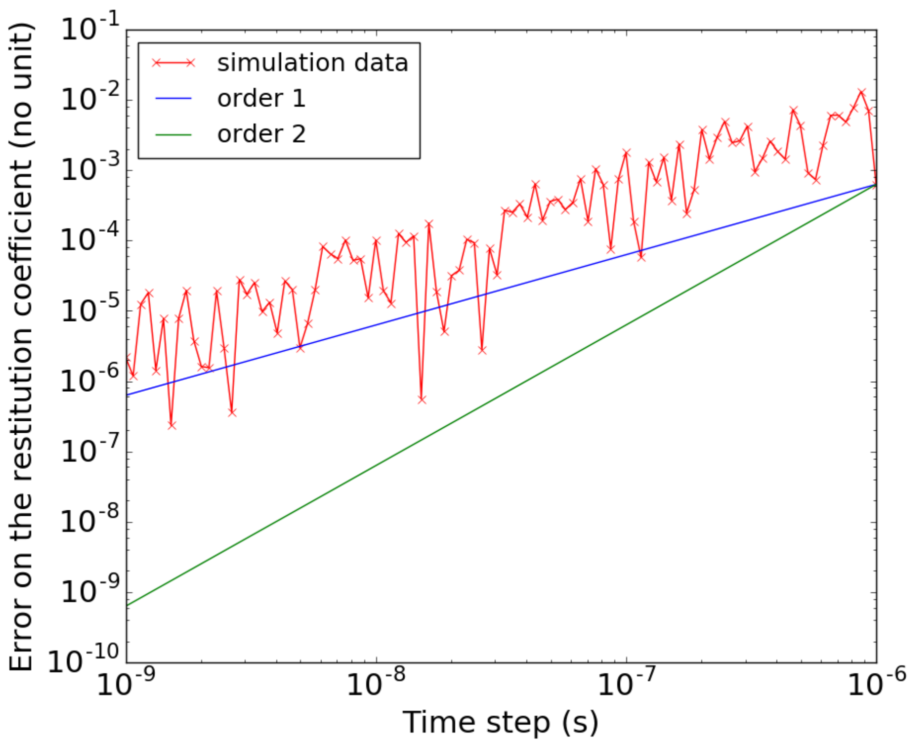
In the GJK distance algorithm, only simplices up to four vertices are considered, that is: a point, a segment, a triangle and a tetrahedron.

Appendix B

Order scaling for other numerical schemes

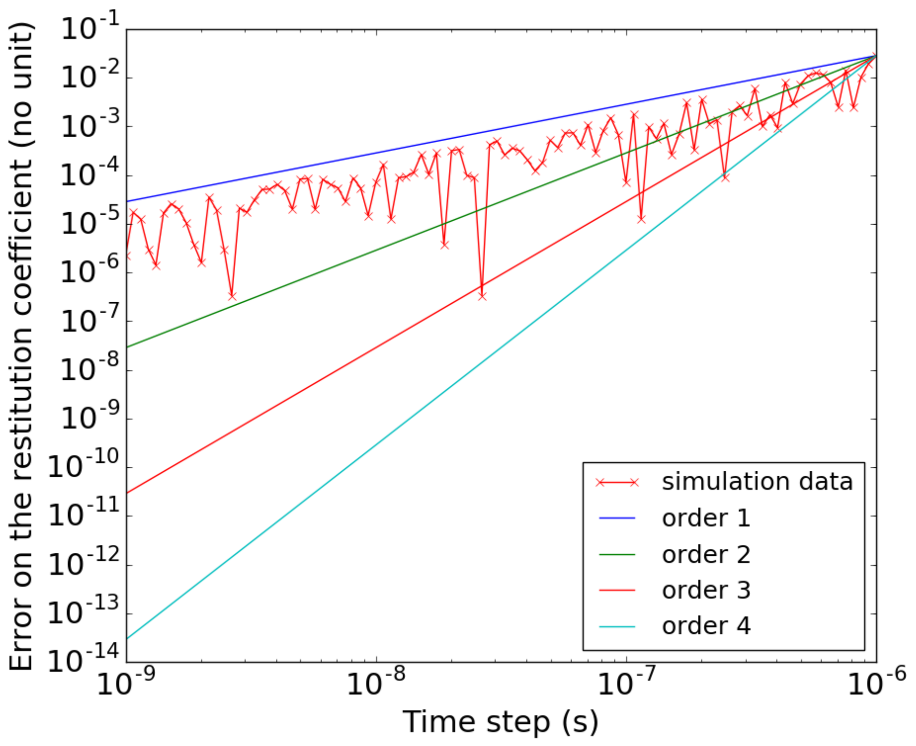


(a) Initial position on the wall

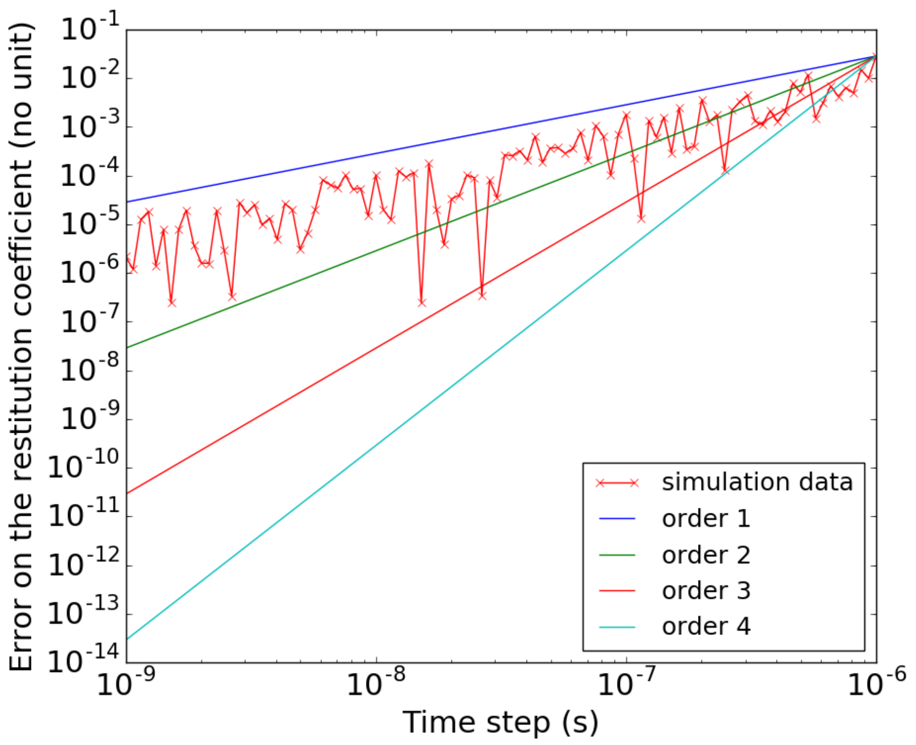


(b) Initial position above the wall

Figure B.1: Restitution coefficient error scaling of a second order Runge-Kutta scheme for the rebound problem, with the particle starting on the surface of the wall (a), and above the surface of the wall (b).

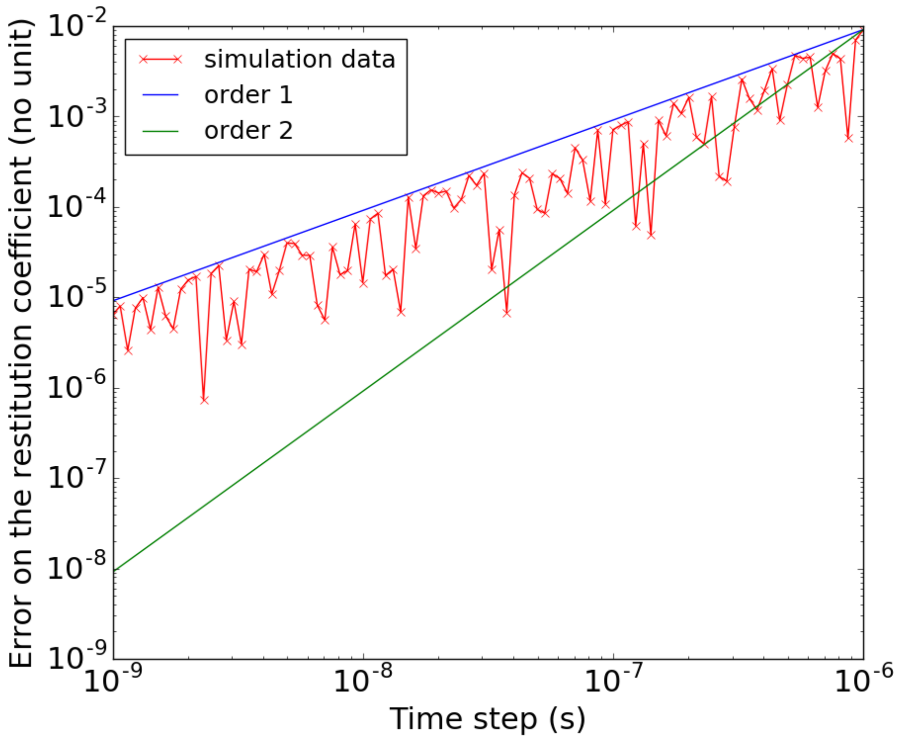


(a) Initial position on the wall

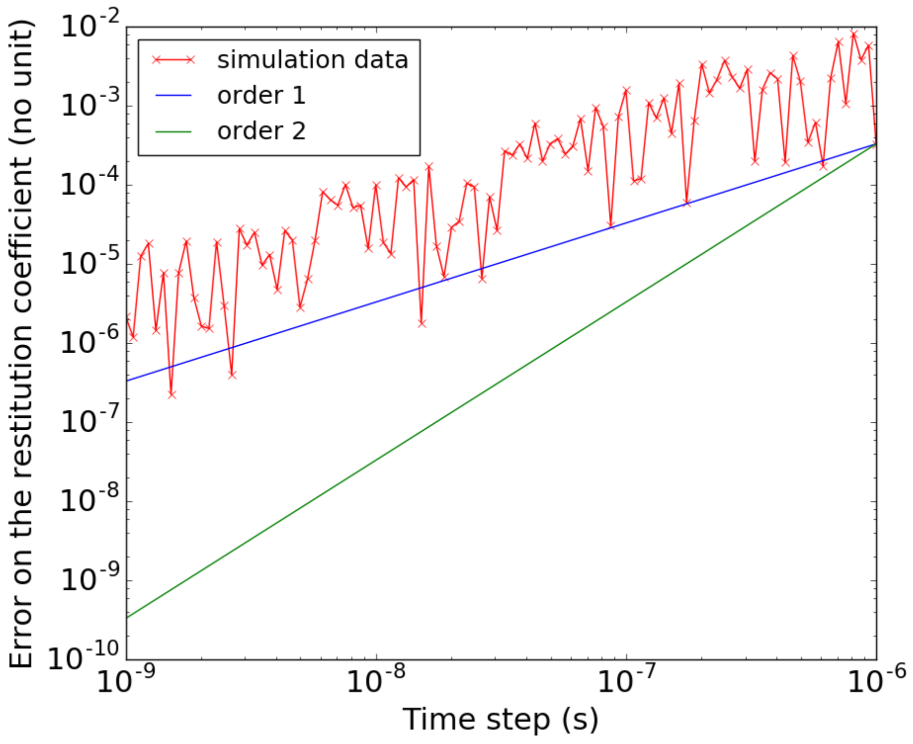


(b) Initial position above the wall

Figure B.2: Restitution coefficient error scaling of a fourth order Runge-Kutta scheme for the rebound problem, with the particle starting on the surface of the wall (a), and above the surface of the wall (b).



(a) Initial position on the wall

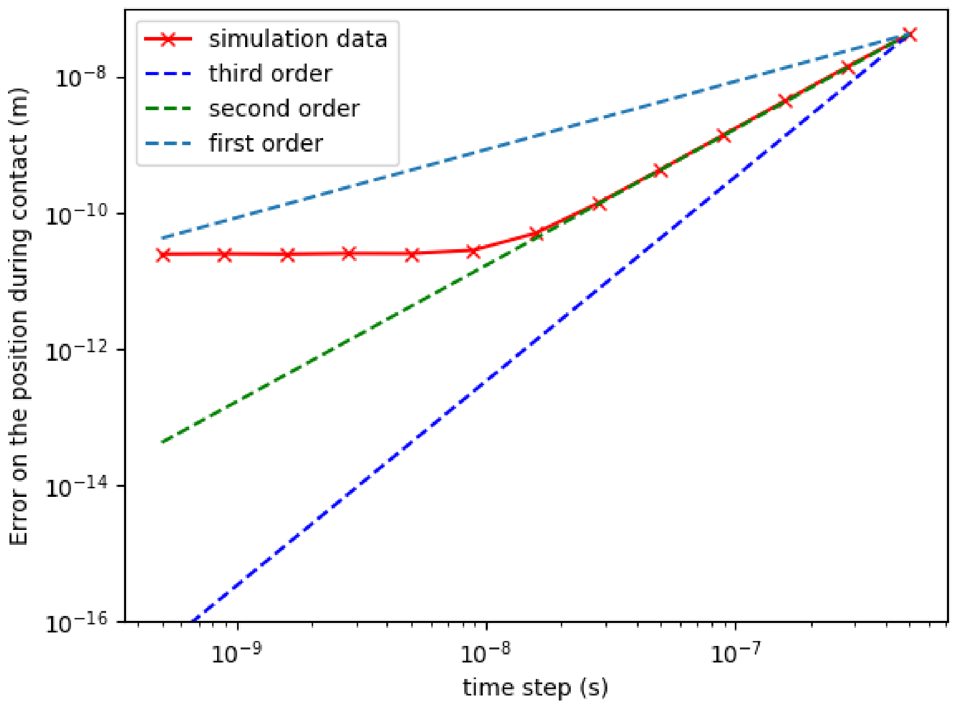


(b) Initial position above the wall

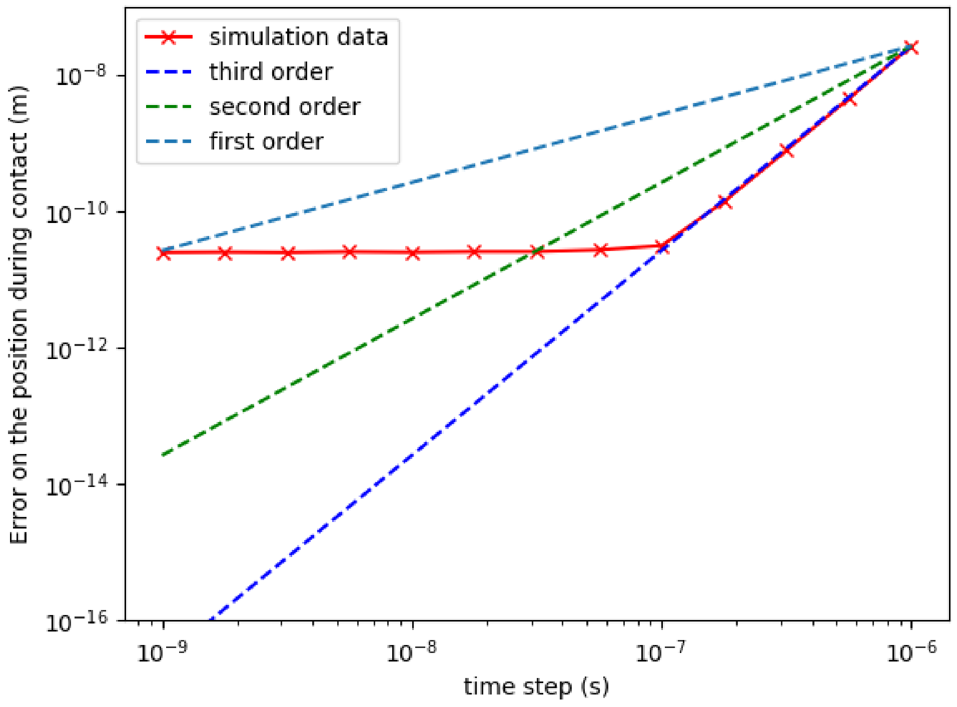
Figure B.3: Restitution coefficient error scaling of a second order Adams-Bashforth scheme for the rebound problem, with the particle starting on the surface of the wall (a), and above the surface of the wall (b).

Appendix C

Order analysis of the newly implemented schemes in *Grains3D*



(a) Second order Runge-Kutta



(b) Third order Adams-Bashforth

Figure C.1: Position error scaling during contact for the second order Runge-Kutta scheme (a) and the third order Adams-Bashforth scheme (b). The contact already exists at t=0 (the penetration depth at t=0 is 10^{-6}m). The plateau around 10^{-11}m is not scheme-related, but is due to the precision storage of values in text files by the code *Grains3D*. The schemes (a) and (b) are therefore true second and third order schemes respectively.