Comparison of methods computing the distance between two ellipsoids

Ivan Girault, Mohamed-Amine Chadil, Stéphane Vincent

Comparison of methods computing the distance between two ellipsoids

Ivan Girault[a,1], Mohamed-Amine Chadil[a,*], Stéphane Vincent[a]

[a]*Laboratoire MSME, CNRS UMR 8208, Université Gustave Eiffel, 5 Boulevard Descartes Marne-la-Vallée, 77454, France*

Abstract

A review of the existing methods to compute the minimal distance between two ellipsoids has been conducted in order to retain the most adequate one within the context of Particle-Resolved Direct Numerical Simulations for particle-laden flows. First, all methods have been implemented and the corresponding algorithms are reported. Furthermore, a procedure has been systematically suggested to control the error associated with each method, when such control was not explicitly available. In a second phase, a two-ellipsoid configuration where an analytical solution is known has been used to perform an error study. This allows to assess the accuracy and consistency of each method, regarding required criteria defined in this paper. The methods that do not verify these criteria have been ruled out. Finally, the remaining methods have been studied on a benchmark of randomly-generated arrays of mono-dispersed spheroids, with aspect ratios ranging from 1/6 to 6 and volume fraction ranging from 0.05 to 0.25. For each method, the spheroidal packings have been sized to measure a statistically significant computing time. Such procedure enabled to study with generality the computing-time dependency of one method on the aspect ratio, the volume fraction, and the desired accuracy. The most efficient method for a given value of these parameters has then been identified.

Keywords: ellipsoid, non-spherical, distance query, Particle-Resolved Direct Numerical Simulation, convex, optimization

1. Introduction

Computing the Euclidean distance between two ellipsoids is of major interest in problems of collision detection between geometrical objects. Those problems arise in various practical fields such as computer graphics [1], robotics [2] or *Particle-Resolved Direct Numerical Simulations* (PR-DNS) of particle-laden flows [3, 4, 5]. In the context of the last example, numerical models used to describe collisions between two immersed ellipsoids require to compute their minimal separating distance. This is necessary to apply distance-dependant lubrication corrections in the near-field regime. Such resolved simulations being extremely time-consuming, any involved numerical step must be optimized. The goal of this paper is thus to retain the most effective method to accurately compute the distance between two ellipsoids.

This work is part of a larger project aiming to enhance the capabilities of an in-house PR-DNS code, named RESPECT (REsolved Scale Particle and Energy Techniques), which at the present time is capable of computing flows seeded with spherical particles. It is based on the Implicit Tensorial Penalty Method proposed by Vincent *et al.* [6], and the collision model for spheres proposed by Brändle de Motta *et al.* [7]. This code has been used to produce numerical simulations of fixed beds [8, 9], fluidized bed [10] and particles in Homogeneous Isotropic Turbulence [11]. The next achievement would be to reproduce such numerical results with non-spherical particles. The ellipsoidal shape is then a pertinent choice because it can model various real non-spherical particles, from elongated fibers to disks [12].

*Corresponding author

Email addresses: ivan.girault@imft.fr (Ivan Girault), amine.chadil@cnrs.fr (Mohamed-Amine Chadil), stephane.vincent@univ-eiffel.fr (Stéphane Vincent)

[1]Present adress: Institut de Mécanique des Fluides de Toulouse (IMFT), 2 Allée du Professeur Camille Soula Toulouse, 31000, France

The mathematical formulation of the problem is now described. Let $(E_m)_{m\in\{1,2\}}$ be two ellipsoids of $\mathbb{R}^3$, each one represented by (see Appendix A):

- its centre $\mathbf{r}_m$.
- a quaternion $\mathbf{q}_m$.
- a symmetric positive-definite 3×3 matrix $\mathbf{A}_m$.
- a vector of $\mathbb{R}^3$ $\mathbf{b}_m$.
- a constant α_m.
- a quadratic form f_m given by:

$$f_m \quad : \quad \begin{pmatrix} \mathbb{R}^3 \to \mathbb{R} \\ \mathbf{x} \mapsto \dfrac{1}{2}\mathbf{x}^\top \mathbf{A}_m \mathbf{x} + \mathbf{b}_m^\top \mathbf{x} + \alpha_m \end{pmatrix}. \tag{1.1}$$

- if E_m is a *spheroid*, meaning an ellipsoid that features an axis of symmetry, it is convenient to define the *aspect ratio* of the spheroid E_m:

$$A_{r,m} = \frac{\text{length of the semi-axis related to the axis of symmetry of } E_m}{\text{length of the other semi-axes of } E_m}. \tag{1.2}$$

The problem to solve is then:

$$\text{Find the unique } (\mathbf{x}_1^*, \mathbf{x}_2^*) \in E_1 \times E_2 \text{ such that } \|\mathbf{x}_1^* - \mathbf{x}_2^*\| = \min_{\mathbf{x}_1\in E_1, \mathbf{x}_2 \in E_2} \|\mathbf{x}_1 - \mathbf{x}_2\| \tag{1.3}$$

Throughout this article, the notation $(\mathbf{x}_m^*)_{m\in[\![1,2]\!]}$ will refer to the exact solution of problem (1.3). The associated exact distance will then be noted d^*, that verifies:

$$d^* = \|\mathbf{x}_1^* - \mathbf{x}_2^*\| = \min_{\mathbf{x}_1\in E_1, \mathbf{x}_2 \in E_2} \|\mathbf{x}_1 - \mathbf{x}_2\|. \tag{1.4}$$

Of course, if E_1 and E_2 are overlapping, a solution $(\mathbf{x}_1^*, \mathbf{x}_2^*)$ is not necessarily unique. In this case, the only concern is to detect this overlap.

At this stage, it becomes necessary to quantitatively specify the required accuracy when solving problem (1.3). In the context of PR-DNS, the necessity to compute the minimal distance between two ellipsoids arises from the computation of distance-dependent lubrication corrections between two close ellipsoids. In the simpler case of spherical particles, distance thresholds are used in the collision model to modulate the lubrication corrections. For a collision between two identical spheres of radius R, the smallest distance threshold d_1 used in RESPECT, (below which the lubrication correction becomes independent of the distance) equals [7]:

$$d_1 = \epsilon_1 R, \text{ with } \epsilon_1 = 10^{-3}. \tag{1.5}$$

As a first approach, Ardekani *et al.* [3] proposed to treat the collision between two ellipsoids as a collision between two equivalent spheres, which radius coincides with the local Gaussian curvature radius of the corresponding ellipsoid at the point of minimal approach. Following this approach, the value of the smallest distance threshold d_1 (1.5) that is used for the collision of two identical spheres dictates the target accuracy ϵ_d that has to be reached when computing the distance between two ellipsoids $(E_m)_{m\in[\![1,2]\!]}$:

$$\epsilon_d = 0.01\epsilon_1 \min_{m\in[\![1,2]\!]} \left[\min_{\mathbf{x}\in\partial E_m} \left(R_{G,m}(\mathbf{x}) \right) \right] = 10^{-5} \min_{m\in[\![1,2]\!]} \left[\min_{\mathbf{x}\in\partial E_m} \left(R_{G,m}(\mathbf{x}) \right) \right] \tag{1.6}$$

where $R_{G,m}$ is the local Gaussian curvature of ellipsoid E_m (see Appendix B). In this article, the application of Eq. (1.6) will be eased by the choice to work with spheroids.

It is proposed to review and compare the existing methods of the literature that are devoted to solve problem (1.3). The most attractive method will be retained according to the following criteria:

- absolute robustness is required over the range of all possible computed distances, which is of the form $[0, d_{\max}]$, $d_{\max}$ being the upper bound above which the lubrication corrections are not active. In particular, the retained method has to handle the case of very close ellipsoids, without convergence issues.
- absolute robustness is required over the range of investigated aspect ratios $A_r = [1/6\ , 6]$, without convergence issues.
- the distance computation error has to be controlled, to satisfy the condition expressed in Eq. (1.6).
- the method must be insensitive to the working scale, meaning that the method behavior (error, computing time) must be the same if a dilation parameter Λ is applied to a particular two-ellipsoid configuration.
- the ideal method has to be, on average, the less time-consuming as possible.

The reviewed methods to solve problem (1.3) are now briefly introduced.

1. Lin *et al.* [13] proposed in 2001 a geometrical iterative algorithm, sometimes called 'Moving Balls algorithm'. In their paper, all the mathematical proofs relative to the convergence of the method are presented. They found "the algorithm to be very reliable and to work very well generally", but stressed out that "the convergence may become a little slow when the ellipsoids are small, thin, and far apart". Thanks to its conceptual simplicity and its ease of implementation, it has been used by Ardekani *et al.* [3] and Lambert *et al.* [5] in the context of PR-DNS.
2. In 1988, a geometrical iterative algorithm, known as the *Gilbert-Johnson-Keerthi (GJK) algorithm*, has been introduced by the related authors [14]. The GJK algorithm originally applies to the more general problem of computing the distance between two convex sets. According to [12], "the GJK algorithm has been unfairly overlooked by the granular dynamics community", "while popular in the computer graphics community for decades" as a collision detection tool between two convex objects. Rare examples of use in the context of granular dynamics simulations are [15, 16, 17]. More references about the GJK algorithm are available in [12].
3. Abbasov [18] proposed for the first time in 2015 a physics-inspired method, the *Newton-Coulomb method* or *Charged Balls method*. The solution of problem (1.3) is considered to be the equilibrium point of a dissipating dynamical system. A basic numerical scheme enables to find this equilibrium point. The method has been corrected and proofs of convergence have been introduced in 2017 [19] by the same author. In the same article, numerical experiments have been presented, suggesting the effectiveness of the algorithm. A recent publication [20] proposed some variants of the method. Note that the procedure is still relevant for smooth convex objects.
4. Tamasyan *et al.* [21] presented in 2014 a way to transform the constrained problem (1.3) into a global problem thanks to the introduction of an *exact penalty function*. This function is then minimized with a *non-smooth analysis* technique, analog to a gradient descent algorithm, but adapted to the case of a non-differentiable function. The method is quite laborious to master, but not very difficult to implement. The authors concluded that their method is more universal than the Moving Balls Method [13], because it is still relevant when considering non-ellipsoidal quadrics, and is less time-consuming than the algebraic approach of Uteshev *et al.* [22].
5. Jain *et al.* [4] introduced in 2019 a new geometrical iterative algorithm to address problem (1.3), as part of an article dealing with their new collision model for ellipsoidal particles in a viscous flow. They stated that their procedure "converges fast", and "is free from any additional parameter" in opposition to the Moving Balls algorithm. No mathematical proof of convergence has been given by the authors. We demonstrate in the current article that this algorithm does not converge for case

with moderately small distances. After communicating with the authors of [4], they found out that the actual distance algorithm used in their publications [4, 23] was different from the one described in [4]. This led them to publish a Corrigendum [24], where the authors acknowledge that the algorithm introduced in [4] suffers from convergence issues. They also state that the actual algorithm (described in the Corrigendum) used in their publications yields a large number of iterations for case with small distances. It is thus "not recommended for further use due to its excessive computational cost" [24]. For information purposes, the presentation of their initial method [4] and its analysis is conserved in the present paper.

6. Uteshev *et al.* [22] have constructed in a 2008 paper an algebraic method, using elimination theory. In the previously mentioned paper [21], the authors stated that their method is "less labor-consuming than the algebraic approach of Uteshev *et al.* [22]". Consequently, there is no need to consider the latter in this paper. Moreover, its implementation is rather challenging.

This article is organized as follows. First, the numerical procedures of the aforementioned methods are described. Then a validation case is considered. It simple enough to be analytically solved, the evaluation of the accuracy of each method is then possible. At this stage, some methods are ruled out because of convergence issues or excessive computational time. Finally, the most promising methods are tested on assemblies of numerous ellipsoidal particles, whose positions and orientations are randomly generated. The purpose is to ensure the reliability of the algorithms and to establish which one is the fastest on average for a given accuracy.

2. Algorithms description

In this section, the practical details necessary to the implementation of each algorithm are described:

- 2.1: Moving Balls Algorithm [13].
- 2.2: Gilbert-Johnson-Keerthi (GJK) Algorithm [14], [25].
- 2.3: Newton-Coulomb or 'Charged balls' method [18], [19], [20].
- 2.4: Exterior Penalty Function Method [21].
- 2.5: Jain *et al.*' algorithm [4].

Some of the reviewed algorithms can treat natively the particular situation where the ellipsoids are overlapping while some cannot. For the latter algorithms, a preliminary procedure has to be run to detect the overlap. Such a procedure has been provided by Jia *et al.* [26]. It is described in Appendix C. Since this test is exact and analytical, it is fast and can thus be used as a preliminary step before using any of the following methods to compute the distance between the ellipsoids E_1 and E_2. Indeed, in the case where the ellipsoids overlap, it avoids running any iterative procedure.

The reader wishing to learn more about one method is invited to refer to the mentioned papers.

2.1. The "Moving Balls" algorithm [13]

The Moving Balls algorithm [13] is an iterative method that provides, at each iteration k, a pair of points $\left(\mathbf{x}_m^k\right)_{m\in[\![1,2]\!]}$ that are located on the surfaces $\left(\partial E_m\right)_{m\in[\![1,2]\!]}$ of the ellipsoids $\left(E_m\right)_{m\in[\![1,2]\!]}$. Lin *et al.* [13] showed that the sequence $\left(\left(\mathbf{x}_m^k\right)_{m\in[\![1,2]\!]}\right)_{k\in\mathbb{N}}$ converges towards $\left(\mathbf{x}_m^*\right)_{m\in[\![1,2]\!]}$ the solution of Problem (1.3).

At each iteration k, the algorithm [13] begins by constructing two spheres centered on points $\left(\mathbf{c}_m^k\right)_{m\in[\![1,2]\!]}$ as illustrated in Fig. 1. How $\left(\mathbf{c}_m^k\right)_{m\in[\![1,2]\!]}$ are constructed will be precised below (see Eq. (2.1)). These spheres are completely inside the ellipsoids $\left(E_m\right)_{m\in[\![1,2]\!]}$, and tangent to the surfaces $\left(\partial E_m\right)_{m\in[\![1,2]\!]}$ at points $\left(\mathbf{x}_m^k\right)_{m\in[\![1,2]\!]}$. The points $\left(\mathbf{x}_m^{k+1}\right)_{m\in[\![1,2]\!]}$ are then constructed as the intersection of the segment $[\mathbf{c}_1^k, \mathbf{c}_2^k]$

with the ellipsoids surfaces $(\partial E_m)_{m\in[\![1,2]\!]}$ as illustrated in Fig. 1 (the details for the computation of the intersections are given in Appendix D). The algorithm iterates by constructing the points $\left(\mathbf{x}_m^k\right)_{m\in[\![1,2]\!]}$, until the convergence criterion discussed in 2.1.1 is reached.

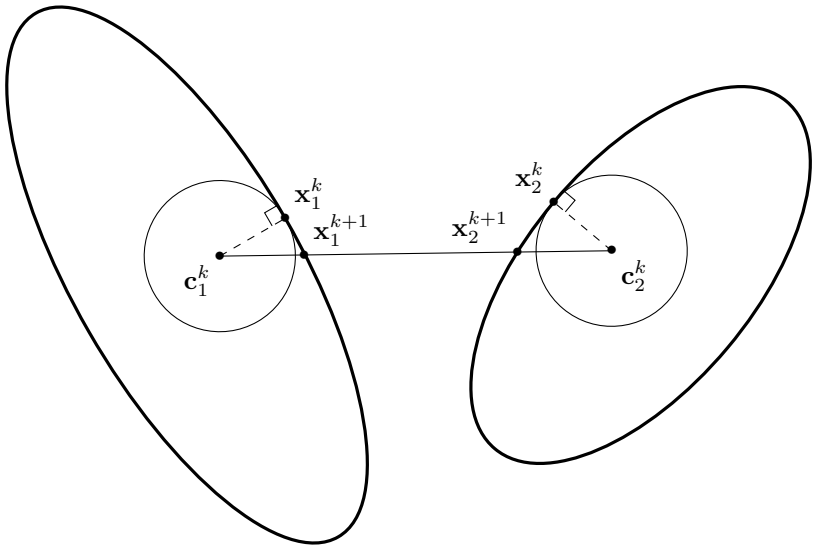


Figure 1: Construction of points $\left(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}\right)$ from $\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right)$.

The main difficulty of this algorithm [13] lies on the construction of the spheres:

- on the one hand, one would want them to be as large as possible to speed up the convergence.
- on the other hand, they must be completely contained in their associated ellipsoid, as it is a sufficient (but not necessary) condition for the algorithm to converge [13].

To that end, Lin *et al.* [13] have demonstrated the following useful result.

Theorem 1. *Let* $E_m = \left\{\mathbf{w} : \frac{1}{2}\mathbf{w}^\top \mathbf{A}_m \mathbf{w} + \mathbf{b}_m^\top \mathbf{w} + \alpha_m \leq 0\right\}$ *be a non-empty ellipsoid. Let* $\mathbf{x}_m^k$ *be a point of* ∂E_m*. Then, for any* $0 < \gamma_m \leq \dfrac{1}{\rho\left(\mathbf{A_m}\right)}$*,*

$$\mathcal{B}\left(\mathbf{x}_m - \gamma_m\left(\mathbf{A}_m\mathbf{x}_m^k + \mathbf{b}_m\right), \gamma_m||\mathbf{A}_m\mathbf{x}_m^k + \mathbf{b}_m||\right) \subset E_m$$

$\rho\left(\mathbf{A}_m\right)$ *being the spectral radius of* $\mathbf{A}_m$*, and* $\mathcal{B}\left(\mathbf{c}, r\right)$ *a ball of center* $\mathbf{c}$ *and radius* r *.*

It is worth noting that:

- The vector $\mathbf{A}_m\mathbf{x}_m^k + \mathbf{b}_m$ is the gradient of the quadratic form representing the ellipsoid E_m. Thus, this vector is normal to the surface ∂E_m, hence the sphere $\mathcal{B}\left(\mathbf{x}_m^k - \gamma_m\left(\mathbf{A}_m\mathbf{x}_m^k + \mathbf{b}_m\right), \gamma_m||\mathbf{A}_m\mathbf{x}_m^k + \mathbf{b}_m||\right)$ is tangent to ∂E_m at point $\mathbf{x}_m^k$.
- The spectral radius $\rho\left(\mathbf{A}_m\right)$ is related to the smallest semi-axis c_m of E_m as $\rho\left(\mathbf{A}_m\right) = \dfrac{1}{c_m^2}$.

Given the previous two remarks, the sphere center $\mathbf{c}_m^k$ can thus be constructed from the surface point $\mathbf{x}_m^k$ as follows:

$$\mathbf{c}_m^k = \mathbf{x}_m^k - \gamma_m\left(\mathbf{A}_m\mathbf{x}_m^k + \mathbf{b}_m\right) \tag{2.1}$$

with γ_m being the parameter introduced in Theorem 1. This parameter value can be fixed by observing that the greater the radii of the constructed spheres are, the faster the algorithm is. It is thus chosen as:

$$\gamma_m = \frac{1}{\rho\left(\mathbf{A}_m\right)} = \text{the square of the smallest semi-axis of the ellipsoid } E_m \tag{2.2}$$

2.1.1. Convergence criterion

A stopping criterion can be established from the fact that the vector $\mathbf{x}_1^* - \mathbf{x}_2^*$ is normal to both the surfaces ∂E_1 and ∂E_2 [13]. That is why, at each iteration $k \to k+1$, the angles $\left(\theta_m^{k+1}\right)_{m\in[\![1,2]\!]}$ between the approached distance vector and the normal vector to the ellipsoids surface $\left(\partial E_m\right)_{m\in[\![1,2]\!]}$ are computed:

$$\begin{cases} \theta_1^{k+1} = \theta\left(\mathbf{x}_2^{k+1} - \mathbf{x}_1^{k+1}, \mathbf{A}_1\mathbf{x}_1^{k+1} + \mathbf{b}_1\right) \\ \theta_2^{k+1} = \theta\left(\mathbf{x}_1^{k+1} - \mathbf{x}_2^{k+1}, \mathbf{A}_2\mathbf{x}_2^{k+1} + \mathbf{b}_2\right) \end{cases}. \tag{2.3}$$

Therefore when the sequence $\left(\left(\mathbf{x}_m^k\right)_{m\in[\![1,2]\!]}\right)_{k\in\mathbb{N}}$ converges towards $\left(\mathbf{x}_m^*\right)_{m\in[\![1,2]\!]}$ then the sequence $\left(\left(\theta_m^k\right)_{m\in[\![1,2]\!]}\right)_{k\in\mathbb{N}}$ converges towards $(0,0)$.

A threshold ϵ_θ is then introduced to stop the program when the following condition is satisfied:

$$\begin{cases} \theta_1^{k+1} \leq \epsilon_\theta \\ \theta_2^{k+1} \leq \epsilon_\theta. \end{cases} \tag{2.4}$$

In Appendix E, an inexpensive implementation of this criterion is proposed.

When the algorithm stops at iteration n, the algorithm returns the value of the current distance and points, which are noted in the following:

$$\begin{cases} \mathbf{x}_1 = \mathbf{x}_1^n \\ \mathbf{x}_2 = \mathbf{x}_2^n \\ d = d^n = \|\mathbf{x}_1^n - \mathbf{x}_2^n\|. \end{cases} \tag{2.5}$$

As mentioned in the introduction, it is desirable to control the error on the computed distance. ϵ_θ has thus to be scaled, to ensure a distance error inferior to a chosen parameter ϵ_d (1.6). To do so, the maximal possible distance error committed with a fixed parameter ϵ_θ is analytically estimated, thanks to geometrical considerations detailed in Appendix F. This analytical estimation uses the local curvature of the ellipsoids surfaces $\left(\partial E_m\right)_{m\in[\![1,2]\!]}$ at the exact solution points $\left(\mathbf{x}_m^*\right)_{m\in[\![1,2]\!]}$

More quantitatively, for $m \in [\![1,2]\!]$, the application

$$R_{\max,m} \quad : \quad \begin{pmatrix} \partial E_m \to \mathbb{R} \\ \mathbf{x}_m \mapsto R_{\max,m}\left(\mathbf{x}_m\right) \end{pmatrix} \tag{2.6}$$

represents the local maximum radius of surface ∂E_m at a given point $\mathbf{x}_m$ (see Appendix B). One of the main results of this paper, demonstrated in Appendix F, is expressed as follows:

$$\boxed{|d - d^*| \leq (\epsilon_\theta)^2 \, \frac{R_{\max,1}\left(\mathbf{x}_1^*\right) + R_{\max,2}\left(\mathbf{x}_2^*\right)}{2}} \tag{2.7}$$

where $d = ||\mathbf{x}_2 - \mathbf{x}_1||$ is the estimated distance obtained after the convergence criterion (2.4) is met, and $d^* = ||\mathbf{x}_2^* - \mathbf{x}_1^*||$ is the exact analytical distance.

This inequality is of particular interest to control the distance error. The parameter ϵ_d can then be introduced to limit this error. To satisfy this control, an expression for ϵ_θ has to be provided, so that when the algorithm stops due to the criterion (2.4), the distance error $|d - d^*|$ verifies:

$$|d - d^*| \leq \epsilon_d. \tag{2.8}$$

From inequality (2.7), the value of ϵ_θ can be chosen as:

$$\epsilon_\theta = \sqrt{\frac{2\epsilon_d}{\max_{\partial E_1}\left(R_{\max,1}\right) + \max_{\partial E_2}\left(R_{\max,2}\right)}}. \tag{2.9}$$

Indeed, this choice will ensure, following inequality (2.7):

$$|d - d^*| \leq \epsilon_d \frac{R_{\max,1}\left(\mathbf{x}_1^*\right) + R_{\max,2}\left(\mathbf{x}_2^*\right)}{\max_{\partial E_1}\left(R_{\max,1}\right) + \max_{\partial E_2}\left(R_{\max,2}\right)} \leq \epsilon_d. \tag{2.10}$$

As explained in the Appendix F, it is possible to implement an adaptive ϵ_θ^k that is computed at each iteration (Eq. (F.6)). Such option enables the algorithm to converge with less iterations (while still ensuring a distance error inferior to ϵ_d), but necessitates to compute the two local curvature radii $\left(R_{\max,m}\left(\mathbf{x}_m^k\right)\right)_{m\in[\![1,2]\!]}$ at each iteration k. For the Moving Balls Algorithm, the speedup due to the reduced number of iterations has been observed to not be sufficient to compensate the overhead due to the curvature computations.

It is stressed out that these considerations about the control of the distance error were not found in the literature, and thus constitute a novel contribution of this article. In Appendix F.3, it is numerically verified on a large number of two-ellipsoid configurations that the proposed values for the parameter ϵ_θ (Eq. (2.9) and Eq. (F.6)) enables the inequality (2.8) to be satisfied.

2.1.2. Overlapping ellipsoids

By definition, when two ellipsoids overlap, the computed distance must be zero. At iteration k, the procedure presented in Appendix D computes the intersection point between an ellipsoid and a segment and provides values for t_1^k and t_2^k such that:

$$\begin{cases} \mathbf{x}_1^{k+1} = \mathbf{c}_1^k + t_1^k\left(\mathbf{c}_2^k - \mathbf{c}_1^k\right) \\ \mathbf{x}_2^{k+1} = \mathbf{c}_1^k + t_2^k\left(\mathbf{c}_2^k - \mathbf{c}_1^k\right). \end{cases} \tag{2.11}$$

If $t_1^k < t_2^k$, the program carries on. Otherwise, the ellipsoids are overlapped and the program returns the distance 0.

2.1.3. Summary of the numerical process

The Moving Balls Algorithm is described in Algorithm 1. The only necessary inputs are the ellipsoids $(E_m)_{m\in[\![1,2]\!]}$ and the parameter ϵ_d.

Algorithm 1 Moving Balls Algorithm

$k = 0$
For $m \in [\![1,2]\!]$, $\mathbf{x}_m^0 = [\mathbf{r}_1, \mathbf{r}_2] \cap \partial E_m$
Set ϵ_θ with Eq. (2.9)
Compute $\left(\theta_m^0\right)_{m\in[\![1,2]\!]}$ with Eq. (2.3)
while $\theta_1^k > \epsilon_\theta$ **or** $\theta_2^k > \epsilon_\theta$ **do**
  Compute $\left(\mathbf{c}_m^k\right)_{m\in[\![1,2]\!]}$ with Eq. (2.1) and Eq. (2.2)
  For $m \in [\![1,2]\!]$, $\mathbf{x}_m^{k+1} = \left[\mathbf{c}_1^k, \mathbf{c}_2^k\right] \cap \partial E_m$
  if $t_1^k \geq t_2^k$ {see 2.1.2} **then**
    return distance $= 0$
  end if
  Compute $\left(\theta_m^{k+1}\right)_{m\in[\![1,2]\!]}$ with Eq. (2.3)
  $k \leftarrow k + 1$
end while
return distance $= \left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\|$

2.2. The Gilbert–Johnson–Keerthi algorithm [14]

The Gilbert-Johnson-Keerthi (GJK) algorithm [14] is an iterative and geometrical method, that leverages on the convexity of the ellipsoidal shape. This special feature enables to reduce problem (1.3) (*ie.* compute the distance between two ellipsoids $(E_m)_{m\in[\![1,2]\!]}$) to the easier task of computing the distance between the origin and a particular convex set C derived from the two ellipsoids. This computation is performed by building a sequence of *simplices* contained in this particular convex set C. Each simplex of the sequence is built to be closer to the origin than the one built at the previous iteration. This process goes on until the distance between the current simplex coincides with the actual distance between C and the origin.

The GJK algorithm manages the computation of the distance between two general convex objects, provided that the *support mapping* of each object can be specified. The reader interested in using the GJK algorithm with objects other than ellipsoids can refer to the corresponding paragraph in [12].

In this section, the GJK algorithm will be explained following the implementation of Van den Bergen [25]. First, the global operation of the method is introduced in 2.2.1. Then, specific topics are exposed in 2.2.2 and 2.2.3. In 2.2.4, all the information necessary to easily implement the algorithm is gathered. Finally, the solution to reconstruct the minimizing points $(\mathbf{x}_m^*)_{m\in[\![1,2]\!]}$ respectively belonging to the ellipsoids $(E_m)_{m\in[\![1,2]\!]}$ is given in 2.2.5.

The presentation of several notions related to convex analysis is deferred to Appendix G.

2.2.1. General operation of the GJK algorithm

Problem (1.3), in the general case of E_1 and E_2 being two convex sets, is equivalent to determining the distance of the Minkowski difference $E_1 - E_2$ (see Appendix G) to the frame origin [12], [25].

The Minkowski difference $E_1 - E_2$ of two convex sets being also convex, the problem is thus reduced to the general problem of projecting the origin onto a convex C:

$$\text{Compute the unique point } \mathbf{v}(C) \in C \text{ such that } ||\mathbf{v}(C)|| = d(C) = \min_{\mathbf{z}\in C} ||\mathbf{z}||. \tag{2.12}$$

Thanks to the *Hilbert Projection Theorem* (Appendix G), the projection $\mathbf{v}(C)$ of the origin onto a convex C is well-defined. In the following, the projection application $\mathbf{v}$ will be often used.

In order to solve the simpler Problem (2.12), with $C = E_1 - E_2$, the GJK algorithm builds a sequence of simplices $\left(S^k\right)_{k\in\mathbb{N}}$ contained in C, with S^{k+1} lying closer to the origin than S_k. Therefore, with the sequence of the projections $\left(\mathbf{v}^k\right)_{k\in\mathbb{N}}$ defined as $\mathbf{v}^k = \mathbf{v}\left(S^k\right)$, the sequence $\left(\|\mathbf{v}^k\|\right)_{k\in\mathbb{N}}$ must be decreasing. In this way, the simplices $\left(S^k\right)_{k\in\mathbb{N}}$ are built to have $||\mathbf{v}^k|| \to d(C)$.

Computationally, each simplex S^k is represented by a set of vertices W^k, with S^k being the convex hull (see Appendix G) of W^k:

$$S^k = \text{conv}\left(W^k\right). \tag{2.13}$$

To build the sequence of simplices $\left(S^k\right)_{k\in\mathbb{N}}$ and associated vertices sets $\left(W^k\right)_{k\in\mathbb{N}}$, the notion of *support mapping* $\mathbf{s}_C$ of a convex set C is used (Appendix G). Initially, the set of vertices W^0, representing the simplex S_0, is set to $\emptyset$. The initial distance vector $\mathbf{v}^0$ is set to be an arbitrary vector in C. From the knowledge of W^k and $\mathbf{v}^k = \mathbf{v}\left(S^k\right)$, the next set of vertices W^{k+1} is built with the following steps:

1. a new vertex $\mathbf{w}^k$ is built thanks to the support mapping $\mathbf{s}_C$ of C:

$$\mathbf{w}^k = \mathbf{s}_C\left(-\mathbf{v}^k\right).$$

2. $\mathbf{v}^{k+1}$ is set as the projection of the origin onto the convex set $\text{conv}\left(W^k \cup \{\mathbf{w}^k\}\right)$:

$$\mathbf{v}^{k+1} = \mathbf{v}\Big(\text{conv}\left(W^k \cup \{\mathbf{w}^k\}\right)\Big).$$

3. W^{k+1} is set as the smallest subset X of $W^k \cup \{\mathbf{w}^k\}$ such that $\mathbf{v}^{k+1}$ belongs to $\text{conv}\,(X)$. This subset X is unique and affinely independent [25].

To provide a stopping criterion, a sequence $\left(\delta^k\right)_{k\in\mathbb{N}}$ is computed at each iteration from the new vertex $\mathbf{w}^{k+1}$ and the new approximated distance vector $\mathbf{v}^{k+1}$:

$$\delta^{k+1} = \frac{\mathbf{w}^{k+1}\cdot\mathbf{v}^{k+1}}{\|\mathbf{v}^{k+1}\|}. \tag{2.14}$$

Geometrically, δ^{k+1} is the distance from the origin to the plane containing $\mathbf{w}^{k+1}$ and normal to $\mathbf{v}^{k+1}$. This distance is always a lower bound on the distance from C to the origin, and converges towards it:

$$\begin{cases} \forall k \in \mathbb{N}, \quad \delta^{k+1} \leq d\,(C) \\ \delta^{k+1} \to d\,(C)\,. \end{cases} \tag{2.15}$$

Contrarily to the sequence $\left(\left\|\mathbf{v}^k\right\|\right)_{k\in\mathbb{N}}$, the sequence $\left(\delta^k\right)_{k\in\mathbb{N}}$ may not be monotonic in k. Hence, a more suitable sequence $\left(\mu^k\right)_{k\in\mathbb{N}}$, monotonically increasing, is computed:

$$\begin{cases} \mu^0 = 0 \\ \mu^{k+1} = \max\left(\mu^k, \delta^{k+1}\right). \end{cases} \tag{2.16}$$

Hence a very practical convergence criterion can be directly established by introducing the parameter ϵ_d:

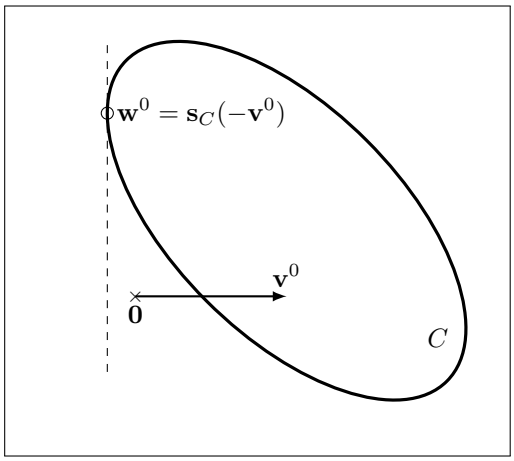
$$\|\mathbf{v}^{k+1}\| - \mu^{k+1} \leq \epsilon_d. \tag{2.17}$$

Indeed, the error on $d\,(C) = d\,(E_1, E_2)$ do not exceed ϵ_d when (2.17) is satisfied.

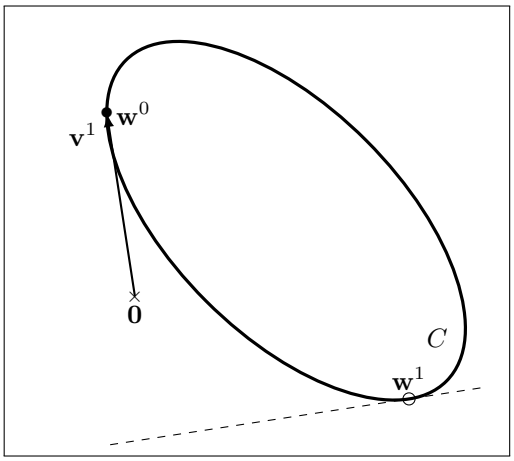
This building procedure is illustrated in Fig. 2.

For clarity purposes, several specific points were not detailed in the previous presentation of this algorithm, and have to be addressed:

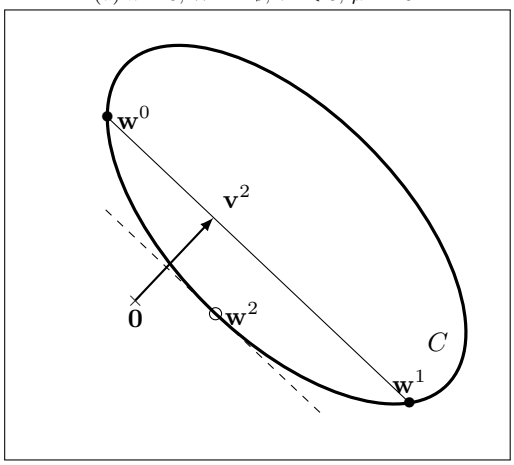
- Until now, the support mapping $\mathbf{s}_C$ of $C = E_1 - E_2$ has not been explicitly defined. Its computation is explained in 2.2.2.
- Performing tasks 2. and 3. is the main difficulty. In practice, they are handled by a single sub-algorithm described in 2.2.3.



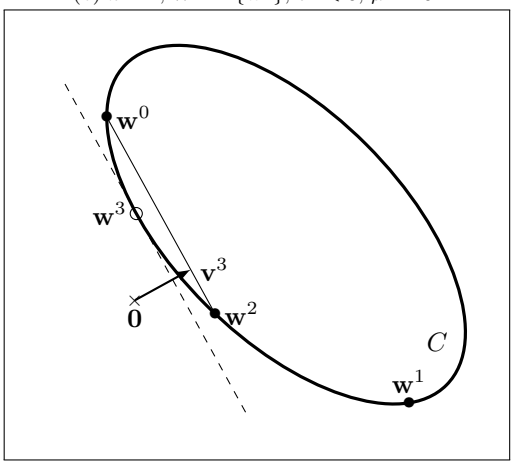
(a) $k = 0$, $W^0 = \emptyset$, $\delta^0 < 0$, $\mu^0 = 0$



(b) $k = 1$, $W^1 = \{\mathbf{w}^0\}$, $\delta^1 < 0$, $\mu^1 = 0$



(c) $k = 2$, $W^2 = \{\mathbf{w}^0, \mathbf{w}^1\}$, $\delta^2 > 0$, $\mu^2 = \delta^2$



(d) $k = 3$, $W^3 = \{\mathbf{w}^0, \mathbf{w}^2\}$, $\delta^3 < \delta^2, \mu^3 = \mu^2$

Figure 2: An example to understand the geometrical process of the GJK algorithm.

2.2.2. Computation of the support mapping

The problem of computing the support mapping $\mathbf{s}_C$ of the set $C = E_1 - E_2$ is now addressed.

From the knowledge of the support mappings $(\mathbf{s}_{E_m})_{m\in[\![1,2]\!]}$ of the two ellipsoids $(E_m)_{m\in[\![1,2]\!]}$, the computation of the support mapping of $\mathbf{s}_C$ C is straightforward (see [25]):

$$\mathbf{s}_C(-\mathbf{v}) = \mathbf{s}_{E_1}(-\mathbf{v}) - \mathbf{s}_{E_2}(\mathbf{v}). \tag{2.18}$$

In the following, the ellipsoids $\left(\tilde{E}_m\right)_{m\in[\![1,2]\!]}$ are centered at the origin, their semi-axes are aligned with the Cartesian axes, and have the same semi-axes length $(a_m, b_m, c_m)_{m\in[\![1,2]\!]}$ as the ellipsoids $(E_m)_{m\in[\![1,2]\!]}$.

From [16], we recall the support mapping of the ellipsoids $\left(\tilde{E}_m\right)_{m\in[\![1,2]\!]}$ is:

$$\mathbf{s}_{\tilde{E}_m}(\mathbf{v}) = \frac{\left((a_m)^2 v_x, (b_m)^2 v_y, (c_m)^2 v_z\right)^\top}{\|(a_m v_x, b_m v_y, c_m v_z)\|}. \tag{2.19}$$

Then, the support mapping of the ellipsoids $(E_m)_{m\in[\![1,2]\!]}$, can be deduced from the knowledge of their associated rotation matrix $(\mathbf{U}_m)_{m\in[\![1,2]\!]}$ and centre $(\mathbf{r}_m)_{m\in[\![1,2]\!]}$. With the definition of the affine transformation $\mathbf{T}_m(\mathbf{x}) = \mathbf{U}_m\mathbf{x} + \mathbf{r}_m$, it yields [25], for $m \in [\![1,2]\!]$:

$$\begin{cases} E_m = \mathbf{T}_m\left(\tilde{E}_m\right) \\ \mathbf{s}_{E_m}(\mathbf{v}) = \mathbf{T}_m\left(\mathbf{s}_{\tilde{E}_m}\left(\mathbf{U}_m^\top \mathbf{v}\right)\right). \end{cases} \tag{2.20}$$

The reader wishing to obtain the adequate support mapping formulae for other geometrical shapes are invited to refer to [16] and [25].

2.2.3. The GJK distance sub-algorithm

This sub-algorithm is introduced to solve the tasks 2. and 3. described in 2.2.1. Its implementation is presented following the article of Van den Bergen [25].

In the following, $Y = W^k \cup \{\mathbf{w}^k\}$. The GJK sub-algorithm is introduced to determine the vector $\mathbf{v}_Y$ and the subset $X \subset Y$ satisfying the following problem:

$$\begin{cases} \mathbf{v}_Y = \mathbf{v}(\mathrm{conv}(Y)) \\ X = \text{ the smallest subset of } Y \text{ such that } \mathbf{v}_Y \in \mathrm{conv}(X) \end{cases} \tag{2.21}$$

where it is recalled that:

- $\mathrm{conv}(Y)$ is the convex hull of the set of points Y.
- $\mathbf{v}(\mathrm{conv}(Y))$ is the projection of the origin onto $\mathrm{conv}(Y)$.

According to [25], Y is affinely independant (if not, the exact distance has been reached at the previous iteration): as a consequence, Y has at most 4 elements and can be indexed as

$$Y = \{\mathbf{y}_0, ..., \mathbf{y}_N\} \quad \text{with} \quad N \leq 3. \tag{2.22}$$

An arbitrary subset X of Y can thus be represented by a subset of indices $I_X \subset I_Y = \{0, ..., N\}$.

For each $Z \subset Y$, the sub-algorithm computes $\mathbf{v}(\text{aff}(Z))$, the distance vector from the origin to the affine hull of Z (see Appendix G). The procedure for computing the distance vector will be explain below. Note that because Y is affinely independent, so are all the subsets $Z \subset Y$. The solution $X = \{\mathbf{y}_i | i \in I_X \subset I_Y\}$ of Problem (2.21) is then characterized by [25]:

$$X = \text{the biggest } Z \subset Y \text{ such that } \begin{cases} \mathbf{v}(\text{aff}(Z)) = \sum_{i \in I_Z} \lambda_i(Z) \mathbf{y}_i \\ \text{with all } \lambda_i(Z) > 0 \text{ and } \sum_{i \in I_Z} \lambda_i(Z) = 1. \end{cases} \tag{2.23}$$

When such X is obtained, we actually have $\mathbf{v}_Y = \mathbf{v}(\text{aff}(X)) = \mathbf{v}(\text{conv}(X))$.

Computing $\mathbf{v}(\text{aff}(X))$ for any $X \subset Y$:

For now, let X be any subset of Y. The following lines details how to compute the vector $\mathbf{v}(\text{aff}(X))$

The vector $\mathbf{v}(\text{aff}(X))$ is represented by the coefficients $(\lambda_i(X))_{i \in I_X}$ such that:

$$\mathbf{v}(\text{aff}(X)) = \sum_{i \in I_X} \lambda_i(X) \mathbf{x}_i, \quad \sum_{i \in I_X} \lambda_i(X) = 1. \tag{2.24}$$

The coefficients $(\lambda_i(X))_{i \in I_X}$ can be obtained from a system of linear equations, deduced from orthogonality conditions [25]. The recursion in Cramer's rules [25] is then used to solve this system, by computing non normalized coefficients $(\Delta_i(X))_{i \in I_X}$, which are closely related to the coefficients $(\lambda_i(X))_{i \in I_X}$ (see below Eq. (2.27)). It then lies the recursion formula, $\forall j \in I_Y$:

$$\Delta_j(\{\mathbf{y}_j\}) = 1 \tag{2.25}$$

$$\forall Z \subset Y \text{ such that } j \notin I_Z, \quad \Delta_j(Z \cup \{\mathbf{y}_j\}) = \sum_{i \in I_Z} \Delta_i(Z) (\mathbf{y}_l \cdot \mathbf{y}_i - \mathbf{y}_j \cdot \mathbf{y}_i) \tag{2.26}$$

where l is a fixed but arbitrary element of I_Z. From the coefficients $(\Delta_j(\{\mathbf{y}_j\}))_{j \in I_Y}$, all the Δ coefficients related to 2-elements subsets of Y can be computed, and so on.

The coefficients $(\lambda_i(X))_{i \in I_X}$ are then related to these non-normalized coefficient with the relation:

$$\forall i \in I_X, \lambda_i(X) = \frac{\Delta_i(X)}{\sum_{j \in I_X} \Delta_j(X)}. \tag{2.27}$$

Exact procedure to solve Problem (2.21):

For each $X \subset Y$, the $(\Delta_i(X))_{i \in I_X}$ are computed, in ascending order of $\text{Card}(X)$. The sub-algorithm identifies the solution X of Problem (2.21) and Eq. (2.23) when both the following conditions are satisfied [25]:

$$\begin{cases} \forall i \in I_X, & \Delta_i(X) > 0 \\ \forall j \notin I_X, & \Delta_j(X \cup \{\mathbf{y}_j\}) \leq 0. \end{cases} \tag{2.28}$$

The sub-algorithm then stops and return the set $W^{k+1} = X$ as well as the new distance vector $\mathbf{v}^{k+1} = \mathbf{v}_Y = \mathbf{v}(\text{aff}(X))$.

Remarks

The following remarks, made by [25], enable to optimize the algorithm:

- the new vertex $\mathbf{w}^k$ is necessarily contained in the solution X of Eq. (2.23), then it reduces the number of subsets contained in Y to be tested for the condition (2.28).
- to apply the recursion formula (2.25), the same dot product of Y vectors might be used multiple times. The coefficients Δ might also be used multiple times. To reduce the computation time, their value can be retained in arrays updated at each iteration (see [25] for how to proceed).

2.2.4. Computational process

The numerical procedure of the GJK algorithm is summarized in Algorithm 2.

Algorithm 2 GJK algorithm

1: $k = 0$
2: $W^0 = \emptyset$
3: $\mathbf{v}^0 = \mathbf{r_1} - \mathbf{r_2}$
4: $\mathbf{w}^0 = \mathbf{s}_C\left(-\mathbf{v}^0\right)$ using Eq. (2.18) and Eq. (2.19)
5: $\mu^0 = 0$
6: **while** $\left\|\mathbf{v}^k\right\| - \mu^k > \epsilon_d$ **do**
7: Sub-algorithm 2.2.3 $\begin{cases}\mathbf{v}^{k+1} = \mathbf{v}\left(\text{conv}\left(W^k \cup \{\mathbf{w}^k\}\right)\right) \\ W^{k+1} = \text{ the smallest subset } X \subset W^k \cup \{\mathbf{w}^k\} \text{ such that } \mathbf{v}^{k+1} \in \text{conv}\left(W^{k+1}\right)\end{cases}$
8: **if** $\left\|\mathbf{v}^{k+1}\right\| = 0$ **then**
9: **return** distance $= 0$ {to avoid evaluating $\mathbf{s}_C$ at $\mathbf{0}$ where it is not defined}
10: **end if**
11: $\mathbf{w}^{k+1} = \mathbf{s}_C\left(-\mathbf{v}^{k+1}\right)$ using Eq. (2.18) and Eq. (2.19)
12: $\delta^{k+1} = \dfrac{\mathbf{v}^{k+1} \cdot \mathbf{w}^{k+1}}{||\mathbf{v}^{k+1}||}$
13: $\mu^{k+1} = \max\left(\mu^k, \delta^{k+1}\right)$
14: $k \leftarrow k + 1$
15: **end while**
16: **return** distance $= \left\|\mathbf{v}^k\right\|$

The significance of the variables is recalled:

- W^{k+1} is a finite subset of points, affinely independent, representing the simplex S^{k+1} (Eq. (2.13)).
- $\mathbf{v}^{k+1}$ is the distance vector of the simplex $S^{k+1} = \text{conv}\left(W^{k+1}\right)$.
- $\mathbf{w}^{k+1}$ is the new vertex built at each iteration.
- $\delta^{k+1} = \dfrac{\mathbf{v}^{k+1} \cdot \mathbf{w}^{k+1}}{||\mathbf{v}^{k+1}||}$ is a lower bound on the distance. Geometrically, it is the distance from the origin to the plane normal to $\mathbf{v}^{k+1}$, containing the point $\mathbf{w}^{k+1}$. It enables to build a monotonic increasing sequence $\left(\mu^k\right)_{k\in\mathbb{N}}$ that converges towards the exact solution for the distance $d\left(C\right)$.
- ϵ_d is the desired distance accuracy.

Note that the GJK algorithm automatically handles the situation where the two ellipsoids collide. In this situation, the sub-algorithm will return at a certain iteration a null distance vector. In this case, the algorithm must stop (see line 8 in Algorithm 2) to avoid evaluating the support mapping $\mathbf{s}_C$ at $\mathbf{0}$ where it is not defined.

The last problem to tackle is the reconstruction of the two minimizing points located on the ellipsoids surfaces. Indeed, the GJK algorithm only seems to provide the difference vector $\mathbf{v}^n \in E_1 - E_2$ defined by these two points. Fortunately, the building process of the GJK algorithm allows to recover $(\mathbf{x}_1^*, \mathbf{x}_2^*) \in E_1 \times E_2$ from the knowledge of $\mathbf{x}_1^*, \mathbf{x}_2^*$.

2.2.5. How to reconstruct the minimizing points belonging to the ellipsoids

After the last iteration n, the returned distance vector $\mathbf{v}^n$ is expressed as a convex combination of the vertices $\mathbf{y}_i \in W^n$:

$$\mathbf{v}^n = \sum_{i \in I_{W^n}} \lambda_i \mathbf{y}_i. \tag{2.29}$$

Now, let us recall that all elements $\mathbf{y}_i \in W^n$ have been computed in the form $\mathbf{y}_i = \mathbf{p}_i - \mathbf{q}_i$, $(\mathbf{p}_i, \mathbf{q}_i) \in E_1 \times E_2$, with Eq. (2.18). This calculation occurs at line 11 in Algorithm 2, when a new vertex $\mathbf{w}^{k+1}$ is computed. Then, by convexity of E_1 and E_2, an estimation of $(\mathbf{x}_1^*, \mathbf{x}_2^*) \in E_1 \times E_2$ such that $\mathbf{v}^n = \mathbf{x}_1^* - \mathbf{x}_2^*$ is obtained [25]:

$$\mathbf{x}_1^* = \sum_{i \in I_{W^n}} \lambda_i \mathbf{p}_i, \quad \mathbf{x}_2^* = \sum_{i \in I_{W^n}} \lambda_i \mathbf{q}_i. \tag{2.30}$$

2.3. Newton-Coulomb Method [18]

This method has been originally introduced by Abbasov in [18]. It is based on a physical principle met by mechanical systems at equilibrium: the minimization of the potential energy.

A fictitious charged point is placed on the surface of each ellipsoid. The principle is to let those two charges evolve under the action of the attractive electrostatic force, with the constraint of staying located on their respective ellipsoid surface. A viscous drag force is added so that convergence occurs. When mechanical equilibrium is reached, the electrostatic potential of interaction between the charged points is minimal, and so the distance between these two points is solution of problem (1.3).

2.3.1. Initial version of the Newton-Coulomb method [18]

Two charged points $m \in [\![1, 2]\!]$ of same mass M and charge q_m are free to move on the surface ∂E_m of the ellipsoid E_m. They are thus represented by the vector $\mathbf{x}_m \in \partial E_m$. The evolution of $(\mathbf{x}_m)_{m \in [\![1,2]\!]}$ is described by Newton's second Law:

$$M_m \ddot{\mathbf{x}}_m = \mathbf{F}_m + \mathbf{N}_m + \mathbf{R}_m \tag{2.31}$$

where:

- $\mathbf{F}_m$ is the electrostatic Coulomb force exerted upon the charged particle m by the other one. It is given by:

$$\mathbf{F}_1 = q_1 q_2 \frac{\mathbf{x}_1 - \mathbf{x}_2}{||\mathbf{x}_1 - \mathbf{x}_2||^3}$$
$$\mathbf{F}_2 = q_1 q_2 \frac{\mathbf{x}_2 - \mathbf{x}_1}{||\mathbf{x}_1 - \mathbf{x}_2||^3}.$$

- $\mathbf{N}_m$ accounts for the normal reaction of the surface ∂E_m, which physically enforces the charge m to stay on it. From equation $\dfrac{d}{dt}\left(\dot{\mathbf{x}}_m \cdot \mathbf{n}_m\right) = 0$, it can be derived:

$$\mathbf{N}_m = -\frac{\mathbf{F}_m \cdot \nabla f_m\left(\mathbf{x}_m\right)}{||\nabla f_m\left(\mathbf{x}_m\right)||^2} \nabla f_m\left(\mathbf{x}_m\right) + \frac{\dot{\mathbf{x}}_m^\top \mathbf{A}_m \dot{\mathbf{x}}_m}{||\nabla f_m\left(\mathbf{x}_m\right)||^2} \nabla f_m\left(\mathbf{x}_m\right).$$

  It is recalled that the matrices $(A_m)_{m \in [\![1,2]\!]}$ and the quadratic forms $(f_m)_{m \in [\![1,2]\!]}$ mathematically describes the ellipsoids $(E_m)_{m \in [\![1,2]\!]}$ (see 1 and Appendix A).

- $\mathbf{R}_m$ is a viscous drag force, with μ a constant:

$$\mathbf{R}_m = -\mu \dot{\mathbf{x}}_m.$$

The set of differential equations representing the system is:

$$\begin{cases} \dot{\mathbf{x}}_1 = \mathbf{z}_1 \\ \dot{\mathbf{x}}_2 = \mathbf{z}_2 \\ \dot{\mathbf{z}}_1 = \boldsymbol{\psi}_1\left(\mathbf{x}_1, \mathbf{y}_2, \mathbf{z}_1, \mathbf{z}_2\right) \\ \dot{\mathbf{z}}_2 = \boldsymbol{\psi}_2\left(\mathbf{x}_2, \mathbf{x}_2, \mathbf{z}_1, \mathbf{z}_2\right) \end{cases} \tag{2.32}$$

with $(\mathbf{x}_m)_{m\in[\![1,2]\!]}$ being the positions of the charged points m respectively located on $(\partial E_m)_{m\in[\![1,2]\!]}$. The $(\boldsymbol{\psi}_m)_{m\in[\![1,2]\!]}$ functions are given by:

$$\boldsymbol{\psi}_1(\mathbf{x}_1,\mathbf{x}_2,\mathbf{z}_1,\mathbf{z}_2) = \frac{p_1}{||\mathbf{x}_1-\mathbf{x}_2||^3}\left(\mathbf{x}_2-\mathbf{x}_1+\frac{(\mathbf{x}_1-\mathbf{x}_2)\cdot\nabla f_1(\mathbf{x}_1)}{||\nabla f_1(\mathbf{x}_1)||^2}\nabla f_1(\mathbf{x}_1)\right) - p_2\mathbf{z}_1 - \frac{\mathbf{z}_1^\top \mathbf{A}_1\mathbf{z}_1}{||\nabla f_1(\mathbf{x}_1)||^2}\nabla f_1(\mathbf{x}_1) \tag{2.33}$$

$$\boldsymbol{\psi}_2(\mathbf{x}_1,\mathbf{x}_1,\mathbf{z}_1,\mathbf{z}_2) = \frac{p_1}{||\mathbf{x}_1-\mathbf{x}_2||^3}\left(\mathbf{x}_1-\mathbf{x}_2+\frac{(\mathbf{x}_2-\mathbf{x}_1)\cdot\nabla f_2(\mathbf{x}_2)}{||\nabla f_2(\mathbf{x}_2)||^2}\nabla f_2(\mathbf{x}_2)\right) - p_2\mathbf{z}_2 - \frac{\mathbf{z}_2^\top \mathbf{A}_2\mathbf{z}_2}{||\nabla f_2(\mathbf{x}_2)||^2}\nabla f_2(\mathbf{x}_2) \tag{2.34}$$

and p_1 and p_2 are parameters related respectively to the electrostatic force and the viscous drag:

$$\begin{cases} p_1 = -\dfrac{q_1 q_2}{M} \\ p_2 = \dfrac{\mu}{M}. \end{cases} \tag{2.35}$$

As proposed in [18], a simple Euler scheme is implemented to solve the problem. To avoid accumulation of errors, the points are projected at each iteration on the surface of each ellipsoid. The associated scheme is, with δ being the time step:

$$\begin{cases} \tilde{\mathbf{x}}_1^k = \mathbf{x}_1^k + \delta\mathbf{z}_1^k \\ \tilde{\mathbf{x}}_2^k = \mathbf{x}_2^k + \delta\mathbf{z}_2^k \\ \mathbf{x}_1^{k+1} = \tilde{\mathbf{x}}_1^k - \dfrac{\nabla f_1\left(\tilde{\mathbf{x}}_1^k\right)}{||\nabla f_1\left(\tilde{\mathbf{x}}_1^k\right)||^2} f_1\left(\tilde{\mathbf{x}}_1^k\right) \\ \mathbf{x}_2^{k+1} = \tilde{\mathbf{x}}_2^k - \dfrac{\nabla f_2\left(\tilde{\mathbf{x}}_2^k\right)}{||\nabla f_2\left(\tilde{\mathbf{x}}_2^k\right)||^2} f_2\left(\tilde{\mathbf{x}}_2^k\right) \\ \mathbf{z}_1^{k+1} = \mathbf{z}_1^k + \delta\boldsymbol{\psi}_1\left(\mathbf{x}_1^k,\mathbf{x}_2^k,\mathbf{z}_1^k,\mathbf{z}_2^k\right) \\ \mathbf{z}_2^{k+1} = \mathbf{z}_2^k + \delta\boldsymbol{\psi}_2\left(\mathbf{x}_1^k,\mathbf{x}_2^k,\mathbf{z}_1^k,\mathbf{z}_2^k\right). \end{cases} \tag{2.36}$$

This algorithm is initialized with $\left(\mathbf{x}_m^0\right)_{m\in[\![1,2]\!]}$ being the intersection between the segment joining the two centers of the ellipsoids, and $(\partial E_m)_{m\in[\![1,2]\!]}$. The stopping criterion is imposed on θ_1 and θ_2 (see 2.1.1 and Eq. (E.1)).

The parameters p_1, p_2 and δ must be tailored to obtain the fastest convergence.

2.3.2. Necessity to modify the initial method

Preliminary studies have shown that the algorithm is not usable in the form (2.36); once the parameters have been adjusted with a particular two-ellipsoids configuration, the algorithm can not handle a new situation where the ellipsoids are closer: the convergence can become prohibitively slow, or even never occurs. This is due to the divergent nature of the electrostatic force when $\|\mathbf{x}_1-\mathbf{x}_2\|\to 0$.

To overcome this difficulty, an attempt has been made in this work to build on this initial method to make it usable. The details of these modifications are described in the next subsection.

2.3.3. Description of the modified Newton-Coulomb Method

"Zeroing speed" scheme [20]: Already introduced in [20], it consists in setting the charged particles speed to zero at each iteration $\left(\mathbf{z}_m^k = 0\right)$. This modification is supposed to avoid convergence issues due to inertia, which can generate oscillations around the equilibrium point. Among all the variants of the Newton-Coulomb method proposed in [20], the "zeroing-speed" technique was proven by the author to be the fastest, which justifies its use here. Note that the numerical experiments realized in [20] were for distance computations between an ellipsoid and a point, and not between two ellipsoids.

This "Zeroing speed" technique leads to new acceleration functions $(\boldsymbol{\xi}_m)_{m\in[\![1,2]\!]}$ analogous to (2.33), but without all the terms depending on the velocities:

$$\begin{cases} \boldsymbol{\xi}_1(\mathbf{x}_1,\mathbf{x}_2) = \dfrac{1}{||\mathbf{x}_1-\mathbf{x}_2||^3}\left(\mathbf{x}_2-\mathbf{x}_1+\dfrac{(\mathbf{x}_1-\mathbf{x}_2)\cdot\nabla f_1(\mathbf{x}_1)}{||\nabla f_1(\mathbf{x}_1)||^2}\nabla f_1(\mathbf{x}_1)\right) \\ \boldsymbol{\xi}_2(\mathbf{x}_1,\mathbf{x}_2) = \dfrac{1}{||\mathbf{x}_1-\mathbf{x}_2||^3}\left(\mathbf{x}_1-\mathbf{x}_2+\dfrac{(\mathbf{x}_2-\mathbf{x}_1)\cdot\nabla f_2(\mathbf{x}_2)}{||\nabla f_2(\mathbf{x}_2)||^2}\nabla f_2(\mathbf{x}_2)\right). \end{cases} \tag{2.37}$$

The numerical scheme to approach the equilibrium point of the dynamical system (2.32) is then:

$$\begin{cases} \tilde{\mathbf{x}}_1^k = \mathbf{x}_1^k + \delta^2\boldsymbol{\xi}_1\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right) \\ \tilde{\mathbf{x}}_2^k = \mathbf{x}_2^k + \delta^2\boldsymbol{\xi}_2\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right) \\ \mathbf{x}_1^{k+1} = \tilde{\mathbf{x}}_1^k - \dfrac{\nabla f_1\left(\tilde{\mathbf{x}}_1^k\right)}{||\nabla f_1\left(\tilde{\mathbf{x}}_1^k\right)||^2} f_1\left(\tilde{\mathbf{x}}_1^k\right) \\ \mathbf{x}_2^{k+1} = \tilde{\mathbf{x}}_2^k - \dfrac{\nabla f_2\left(\tilde{\mathbf{x}}_2^k\right)}{||\nabla f_2\left(\tilde{\mathbf{x}}_2^k\right)||^2} f_2\left(\tilde{\mathbf{x}}_2^k\right). \end{cases} \tag{2.38}$$

It is worth noting that the only remaining parameter here is the time step δ, which has absorbed the parameter p_1 related to the electrostatic force. The proposed scheme (2.38) is different from the one suggested in [20], in which the acceleration at iteration k is computed from the previous positions at iteration $k-1$, whereas here it is computed with the current positions. It has been observed that a faster convergence is obtained by doing so.

Nonetheless, even if the number of parameters is now reduced, a numerical procedure is still needed to scale the time step δ, to avoid convergence issues or too large numbers.

Adaptive time-step: It is proposed here to adapt the time step δ, so that the displacement amplitude $\delta^2\boldsymbol{\xi}_m\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right)$ is adjusted to the local curvature of the ellipsoid surface ∂E_m. This modification is inspired from Jain *et al.* [4], as they introduce a step size that takes into account regions of increased curvature for flattened or elongated shapes. The minimum curvature radius at point $\mathbf{x}_m^k$ belonging to the surface ∂E_m is noted $R_{\min,m}^k$ (see Appendix B). The time-step δ^k, used to compute the positions at time $k+1$ following Eq. (2.38), could be obtained from:

$$\delta^k = \min_{m\in\{1,2\}} \sqrt{\frac{C_{\text{curv}} R_{\min,m}^k}{||\boldsymbol{\xi}_m\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right)||}}. \tag{2.39}$$

Eq. (2.39) enforces the displacement $\left(\delta^k\right)^2\boldsymbol{\xi}_m\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right)$ to be inferior to the limit $C_{\text{curv}}R_{\min,m}^k$, with C_{curv} a constant which must be adjusted (see Eq. (2.45) in the corresponding paragraph "Adaptive C_{curv}").

Since exactly describing the trajectories of the fictitious charged points is not mandatory, two time-steps δ_1^k and δ_1^k can be simultaneously used, each one adapted to its relative ellipsoid:

$$\delta_1^k = \sqrt{\frac{C_{\text{curv}} R_{\min,1}^k}{||\boldsymbol{\xi}_1\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right)||}}, \quad \delta_2^k = \sqrt{\frac{C_{\text{curv}} R_{\min,2}^k}{||\boldsymbol{\xi}_2\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right)||}}. \tag{2.40}$$

The points at time $k+1$ are then computed as follows:

$$\begin{cases} \tilde{\mathbf{x}}_1^k = \mathbf{x}_1^k + \left(\delta_1^k\right)^2\boldsymbol{\xi}_1\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right) \\ \tilde{\mathbf{x}}_2^k = \mathbf{x}_2^k + \left(\delta_2^k\right)^2\boldsymbol{\xi}_2\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right) \\ \mathbf{x}_1^{k+1} = \tilde{\mathbf{x}}_1^k - \dfrac{\nabla f_1\left(\tilde{\mathbf{x}}_1^k\right)}{||\nabla f_1\left(\tilde{\mathbf{x}}_1^k\right)||^2} f_1\left(\tilde{\mathbf{x}}_1^k\right) \\ \mathbf{x}_2^{k+1} = \tilde{\mathbf{x}}_2^k - \dfrac{\nabla f_2\left(\tilde{\mathbf{x}}_2^k\right)}{||\nabla f_2\left(\tilde{\mathbf{x}}_2^k\right)||^2} f_2\left(\tilde{\mathbf{x}}_2^k\right). \end{cases} \tag{2.41}$$

Preliminary tests have shown that the method works, except when the ellipsoids are very close (below $10^{-2} - 10^{-3}$ relative to the dimension of the ellipsoids). In those cases, convergence can become very slow, or never occur. Two additional modifications are still necessary, namely, the adding of a "backtracking loop" and the reduction of the amplitude displacement when the ellipsoids are very close. These modifications are detailed hereafter.

Backtracking loop [20]: This modification was introduced in [20]. It consists in verifying that the time-steps $\left(\delta_m^k\right)_{m\in[\![1,2]\!]}$ ensure the distance to decrease at the $k+1^{\text{th}}$ step . If not, the δ_m^k are corrected with the procedure described in Algorithm 3.

Algorithm 3 Backtracking loop for time-step correction at iteration k

1: For $m \in [\![1,2]\!]$, $\begin{cases} \tilde{\mathbf{z}}_m = \mathbf{x}_m^k + \left(\delta_m^k\right)^2 \boldsymbol{\xi}_m\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right) \\ \mathbf{z}_m = \tilde{\mathbf{z}}_m - \dfrac{\nabla f_m\left(\tilde{\mathbf{z}}_m\right)}{||\nabla f_m\left(\tilde{\mathbf{z}}_m\right)||^2} f_m\left(\tilde{\mathbf{z}}_m\right) \end{cases}$
2: **while** $\|\mathbf{z}_1 - \mathbf{z}_2\| \geq \left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\|$ **do**
3: $\quad \delta_m^k \leftarrow \lambda \delta_m^k$
4: $\quad$ For $m \in [\![1,2]\!]$, $\begin{cases} \tilde{\mathbf{z}}_m = \mathbf{x}_m^k + \left(\delta_m^k\right)^2 \boldsymbol{\xi}_m\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right) \\ \mathbf{z}_m = \tilde{\mathbf{z}}_m - \dfrac{\nabla f_m\left(\tilde{\mathbf{z}}_m\right)}{||\nabla f_m\left(\tilde{\mathbf{z}}_m\right)||^2} f_m\left(\tilde{\mathbf{z}}_m\right) \end{cases}$
5: **end while**
6: For $m \in [\![1,2]\!]$, $\mathbf{x}_m^{k+1} = \mathbf{z}_m$
7: **return** $\left(\mathbf{x}_m^{k+1}\right)_{m\in[\![1,2]\!]}$

where λ is a shrinking factor < 1 .

Adaptive C_{curv}: Some tests have shown that for a constant C_{curv} value, the method could still have convergence issues when the distance between the ellipsoids becomes very small, even when adding a backtracking loop. If the convergence is not obtained with a fixed value of C_{curv}, it has been observed that the C_{curv} value could be diminished *a posteriori* to obtain the convergence.

This observation proves that some arguments are needed to scale the C_{curv} value in the limit of close ellipsoids.

Actually, some tests have shown that the convergence problems arise when the distance between ellipsoids becomes of the same magnitude than the distance e_m^k of the point $\tilde{\mathbf{x}}_m^k$ from the ellipsoid surface ∂E_m. This distance e_m^k, induced by the tangential displacement $C_{\text{curv}} R_{\text{min},m}^k$ from the point $\mathbf{x}_m^k$, is represented in Fig. 3. It can be seen that e_m^k depends on C_{curv}, $R_{\text{min},m}^k$ the minimum radius curvature at point $\mathbf{x}_m^k$ and $R_{\text{loc},m}^k$ the curvature radius at point $\mathbf{x}_m^k$ in the direction of the displacement.

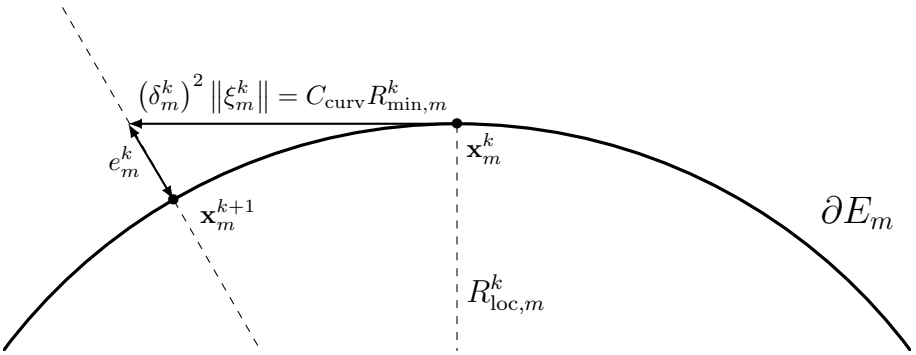


Figure 3: Distance e_m^k from the ellipsoid surface generated by a tangential displacement of magnitude $C_{\text{curv}} R_{\text{min},m}^k$. $R_{\text{loc},m}^k$ is the local curvature radius of the surface ∂E_m at point $\mathbf{x}_m^k$ in the direction of the displacement, and it is recalled that $R_{\text{min},m}^k$ is the minimal local curvature radius of the surface ∂E_m at the point $\mathbf{x}_m^k$.

Of course, this distance e_m^k is compensated by the projection steps described by the third and fourth lines of Eq (2.41), which enable to compute the point $\mathbf{x}_m^{k+1}$. Nonetheless, these steps are first-order projections, and a small error can thus subsist. Assuming that this error is responsible for the observed convergence issues, it seems reasonable to think that it will not play any role if the following inequality is satisfied:

$$e_m^k << ||\mathbf{x}_1^k - \mathbf{x}_2^k||. \tag{2.42}$$

The maximum possible value of e_m^k can be analytically estimated: this limit case occurs when $R_{\text{loc},m}^k = R_{\text{min},m}^k$. To find this maximal value, a second-order equation has to be solved, which solution is:

$$e_m^k \left(R_{\text{loc},m}^k = R_{\text{min},m}^k\right) = R_{\text{min},m}^k \left(\sqrt{1 + C_{\text{curv}}^2} - 1\right) \simeq R_{\text{min},m}^k \frac{C_{\text{curv}}^2}{2}. \tag{2.43}$$

To ensure Eq. (2.42), C_{curv} must then verify:

$$C_{\text{curv}} << \sqrt{\frac{2||\mathbf{x}_1^k - \mathbf{x}_2^k||}{R_{\text{min},m}^k}}. \tag{2.44}$$

This is why it is proposed to compute a different $C_{\text{curv},m}^k$ for each ellipsoid m, at each iteration k, following:

$$C_{\text{curv},1}^k = \min\left(C_{\max}, \tau\sqrt{\frac{2||\mathbf{x}_1^k - \mathbf{x}_2^k||}{R_{\text{min},1}^k}}\right), \quad C_{\text{curv},2}^k = \min\left(C_{\max}, \tau\sqrt{\frac{2||\mathbf{x}_1^k - \mathbf{x}_2^k||}{R_{\text{min},2}^k}}\right) \tag{2.45}$$

with $\tau < 1$, and $C_{\max}$ being the maximum possible value of $C_{\text{curv},m}^k$. $C_{\max}$ is an additional parameter whose value has to be calibrated.

2.3.4. Convergence criterion

The same criterion expressed as in 2.1.1 for the description of the Moving Balls algorithm is used for this algorithm, so that a controlling parameter ϵ_d has to be used, which ensures the error of the computed distance to be inferior to ϵ_d. As explained in Appendix F, two possibilities are available for implementing this convergence criterion. For the Newton-Coulomb method, the modified implementation already necessitates computing the local curvature, so that the adaptive version Appendix F.2.2 of the convergence criterion can be advantageously used.

2.3.5. Ellipsoids overlap

This algorithm has to be supplemented with an additional algorithm to detect a potential overlap. Indeed, the electrostatic force is singular when $\left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\| = 0$. To avoid this situation, Jia *et al.*'s exact overlap detection algorithm [26], described in Appendix C, is used before executing the Newton-Coulomb Method. If an overlap is detected, the returned distance is equaled to zero. Otherwise, the Newton-Coulomb Method is run to compute the distance.

2.3.6. Summary of the numerical process

Since multiple modifications have been introduced, the numerical steps of the Newton-Coulomb method are recalled, in Algorithm 4. The important parameters are the following:

- ϵ_d the desired error of the distance.
- $C_{\max}$ (Eq. (2.45)).
- τ (Eq. (2.45)).
- λ the parameter associated with the backtracking loop.

Algorithm 4 Newton-Coulomb Method

$k = 0$
if collision = .TRUE. (Collision Detection Algorithm Appendix C) **then**
  return distance = 0
end if
For $m \in [\![1,2]\!]$, $\mathbf{x}_m^0 = [\mathbf{r}_1, \mathbf{r}_2] \cap \partial E_m$
Set $\epsilon_\theta^0 = \epsilon_\theta$ with Eq. (2.9)
Compute $\left(\theta_m^0\right)_{m\in[\![1,2]\!]}$ with Eq. (2.3)
while $\theta_1^k > \epsilon_\theta^k$ **or** $\theta_2^k > \epsilon_\theta^k$ **do**
  Compute $\left(C_{\text{curv},m}^k\right)_{m\in[\![1,2]\!]}$ with Eq. (2.45) and $\left(\delta_m^k\right)_{m\in[\![1,2]\!]}$ with Eq. (2.40)
  Compute the new points $\left(\mathbf{x}_m^{k+1}\right)_{m\in[\![1,2]\!]}$ thanks to Eq. (2.41) and Eq. (2.37)
  Correct the points $\left(\mathbf{x}_m^{k+1}\right)_{m\in[\![1,2]\!]}$ with the backtracking loop described in Algorithm 3
  Compute ϵ_θ^{k+1} with Eq. (F.6)
  Compute $\left(\theta_m^{k+1}\right)_{m\in[\![1,2]\!]}$ with Eq. (2.3)
  $k \leftarrow k+1$
end while
return distance = $\left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\|$

2.4. Exterior Penalty Function Method [21]

This method is an iterative descent method, providing a sequence of points $\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right)^k$ converging towards the solution of Problem (1.3). It involves a reformulation of the constrained Problem 1.3 (*ie.* $\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right) \in \partial E_1 \times \partial E_2$) into an unconstrained problem, thanks to the introduction of a global function defined $\forall \mathbf{z}^k = \left(\mathbf{x}_1^k, \mathbf{x}_2^k\right) \in \mathbb{R}^6$ termed as a *penalty function*. This penalty function, defined from the two quadratic forms describing the ellipsoids, is found to be minimized by the solution $\mathbf{z}^* = \left(\mathbf{x}_1^*, \mathbf{x}_2^*\right)^\top$ of Problem (1.3). Given that the penalty function is not everywhere differentiable, a special gradient method has to be designed to find the minimizing solution. This has been achieved in [21], using *non-smooth analysis* tools. Only the strict computing process associated with the method is presented in the following, all the mathematical justification can be found in the original paper [21].

2.4.1. Reformulating the problem

To stay consistent with the notations in [21], the quadratic forms associated with the ellipsoids $(E_m)_{m\in[\![1,2]\!]}$ (see Appendix A) are designated by $(h_m)_{m\in[\![1,2]\!]}$ instead of $(f_m)_{m\in[\![1,2]\!]}$ given in (1.1).

The ellipsoids $(E_m)_{m\in[\![1,2]\!]}$ are thus defined by:

$$E_m = \left\{\mathbf{x}\in\mathbb{R}^3 \middle| h_m(\mathbf{x}) \leq 0\right\}. \tag{2.46}$$

Problem (1.3) can be formally reformulated in a 6-dimensional formalism, where two points $(\mathbf{x}_1, \mathbf{x}_2)$ are concatenated into a unique variable

$$\mathbf{z} = \begin{pmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{pmatrix} \in \mathbb{R}^6. \tag{2.47}$$

A function norm f is then defined $\forall \mathbf{z}\in\mathbb{R}^6$ as:

$$f(\mathbf{z}) = \frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2}. \tag{2.48}$$

The set $Z\subset\mathbb{R}^6$ is also defined:

$$Z = \left\{\mathbf{z}\in\mathbb{R}^6 \middle| \mathbf{z} = \begin{pmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{pmatrix} \in \partial E_1 \times \partial E_2\right\} \tag{2.49}$$

and can be re-written as:

$$\begin{cases} Z = \{\mathbf{z}\in\mathbb{R}^6 | \phi(\mathbf{z}) = 0\} \\ \text{with } \phi(\mathbf{z}) = |h_1(\mathbf{x_1})| + |h_2(\mathbf{x_2})|. \end{cases} \tag{2.50}$$

In this new 6-dimensional formalism, problem (1.3) is equivalent to:

$$\text{Find } \mathbf{z}_m \text{ such as } f(\mathbf{z}_m) = \min_{\phi(\mathbf{z})=0} f(\mathbf{z}). \tag{2.51}$$

To solve this constrained problem, an *exact penalty function* Φ_λ (see [21]) can be built:

$$\lambda > 0, \quad \forall \mathbf{z}\in\mathbb{R}^6, \quad \Phi_\lambda(\mathbf{z}) = f(\mathbf{z}) + \lambda\phi(\mathbf{z}). \tag{2.52}$$

The use of such function is justified by the following theorem [21]:

Theorem 2. *If we endow $\mathbb{R}^6$ with the Euclidean norm and the associated metric, there exists $\lambda^*\in\mathbb{R}_+$ such as $\forall\lambda > \lambda^*$, a local minimum z_m of f in Z is also a local minimum of Φ_λ in $\mathbb{R}^6$.*

The problem is thus reduced to minimize Φ_λ in $\mathbb{R}^6$ without constraint. Note that the constant λ is called a *penalty*, and can be numerically chosen *a posteriori*, without a theoretical knowledge of λ^*.

Nonetheless, the problem is not simple, as Φ_λ is not differentiable where $h_i(\mathbf{x}_i) = 0$. Tamasyan *et al.* [21] provides an iterative descent procedure consistent with the previous fact, using non-smooth analysis. The next subsection is devoted to describe the numerical procedure that computes the direction of descent of Φ_λ at the point $\mathbf{z}^k = \left(\mathbf{x}_1^k, \mathbf{x}_2^k\right)$.

2.4.2. Computing the direction of descent

Let us consider $\mathbf{z}^k\in\mathbb{R}^6$ the point built at the iteration k. To construct $\mathbf{z}^{k+1}$, the direction of descent must be computed to ensure $\Phi_\lambda(\mathbf{z}^{k+1}) < \Phi_\lambda(\mathbf{z}^k)$.

Since Φ_λ is not necessarily differentiable at the point $\mathbf{z}^k$, its gradient cannot be always defined, and a weaker mathematical notion have to be used. That is why the *hypo-differential* $d\Phi_\lambda$ of Φ_λ is introduced [21]. The hypo-differential at the point $\mathbf{z}^k$, $d\Phi_\lambda\left(\mathbf{z}^k\right)$, is a subset of $\mathbb{R}^7$, and can be expressed in the form:

$$d\Phi_\lambda(\mathbf{z}^k) = \left\{\mathbf{W}\left(\mathbf{z}^k, \mu_1, \mu_2\right)\in\mathbb{R}^7 \middle| \quad \mu_1,\mu_2\in\mathbb{R}, |\mu_1|, |\mu_2| < \lambda\right\}. \tag{2.53}$$

To express the function $\mathbf{W}$ in Eq. (2.53), the quantities $\mathbf{Q}\left(\mathbf{z}^k\right)$, $\mathbf{h'}_1\left(\mathbf{x}_1\right)$, $\mathbf{h'}_2\left(\mathbf{x}_2\right)$ for $\mathbf{z} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^6$ are defined:

$$\mathbf{Q}(\mathbf{z}) = \begin{pmatrix} \mathbf{x}_1 - \mathbf{x}_2 \\ -(\mathbf{x}_1 - \mathbf{x}_2) \end{pmatrix} \in \mathbb{R}^6, \quad \mathbf{h'}_1(\mathbf{x}_1) = \begin{pmatrix} \mathbf{A}_1\mathbf{x}_1 + \mathbf{b}_1 \\ \mathbf{0}_{\mathbb{R}^3} \end{pmatrix} \in \mathbb{R}^6, \quad \mathbf{h'}_2(\mathbf{x}_2) = \begin{pmatrix} \mathbf{0}_{\mathbb{R}^3} \\ \mathbf{A}_2\mathbf{x}_2 + \mathbf{b}_2 \end{pmatrix} \in \mathbb{R}^6. \tag{2.54}$$

Thus, $\mathbf{W}$ is expressed as [21]:

$$\mathbf{W}\left(\mathbf{z}^k, \mu_1, \mu_2\right) = \begin{pmatrix} \mu_1 h_1(\mathbf{x}_1) + \mu_2 h_2(\mathbf{x}_2) - \lambda\phi(\mathbf{z}) \\ \mu_1 \mathbf{h'}_1(\mathbf{x}_1) + \mu_2 \mathbf{h'}_2(\mathbf{x}_2) + \mathbf{Q}(\mathbf{z}) \end{pmatrix}. \tag{2.55}$$

The direction of descent from $\mathbf{z}^k$ is provided by the smallest vector of $d\Phi_\lambda(\mathbf{z^k})$ [21]. We thus have to find $\mathbf{W}^*\left(\mathbf{z}^k\right)$ verifying:

$$\|\mathbf{W}^*\|^2 = \min_{\mathbf{W} \in d\Phi_\lambda(\mathbf{z}^k)} \|\mathbf{W}\|^2 = \min_{\mu_1, \mu_2 \in [-\lambda, \lambda]} \left\|\mathbf{W}\left(\mathbf{z}^k, \mu_1, \mu_2\right)\right\|^2 \tag{2.56}$$

$$= \left\|\mathbf{W}\left(\mathbf{z}^k, \mu_1^*, \mu_2^*\right)\right\|^2. \tag{2.57}$$

To find (μ_1^*, μ_2^*), the partial derivatives of $(\mu_1, \mu_2) \rightarrow ||\mathbf{W}(\mathbf{z}^k, \mu_1, \mu_2)||^2$ must be equaled zero, which implies the following system of equations to be satisfied:

$$\begin{pmatrix} h_1^2(\mathbf{x}_1^k) + \mathbf{h'}_1(\mathbf{x}_1^k) \cdot \mathbf{h'}_1(\mathbf{x}_1^k) & h_1(\mathbf{x}_1^k)h_2(\mathbf{x}_2^k) \\ h_1(\mathbf{x}_1^k)h_2(\mathbf{x}_2^k) & h_2^2(\mathbf{x}_2^k) + \mathbf{h'}_2(\mathbf{x}_2^k) \cdot \mathbf{h'}_2(\mathbf{x}_2^k) \end{pmatrix} \begin{pmatrix} \mu_1^* \\ \mu_2^* \end{pmatrix} = \begin{pmatrix} \lambda\phi\left(\mathbf{z}^k\right) h_1\left(\mathbf{x}_1^k\right) - \mathbf{h'}_1\left(\mathbf{x}_1^k\right) \cdot \mathbf{Q}\left(\mathbf{z}^k\right) \\ \lambda\phi\left(\mathbf{z}^k\right) h_2\left(\mathbf{x}_2^k\right) - \mathbf{h'}_2\left(\mathbf{x}_2\right) \cdot \mathbf{Q}\left(\mathbf{z}^k\right) \end{pmatrix}. \tag{2.58}$$

One can ensure that $|\mu_m^*| < \lambda$ by setting a value of λ large enough [21].

The direction of descent $\mathbf{G}^*(\mathbf{z}^k) \in \mathbb{R}^6$, called the hypogradient of Φ_λ at point $\mathbf{z}^k$, is then obtained as the last 6 components of $\mathbf{W}^*\left(\mathbf{z}^k\right)$, namely:

$$\mathbf{G}^*(\mathbf{z}^k) = \mu_1^* \mathbf{h'}_1(\mathbf{x}_1^k) + \mu_2^* \mathbf{h'}_2(\mathbf{x}_2^k) + \mathbf{Q}(\mathbf{z^k}). \tag{2.59}$$

which can be written as:

$$\mathbf{G}^*\left(\mathbf{z}^k\right) = \begin{pmatrix} \mathbf{G}_1^*\left(\mathbf{x}_1^k\right) \\ \mathbf{G}_2^*\left(\mathbf{x}_2^k\right) \end{pmatrix} = \begin{pmatrix} \mu_1^*\left(\mathbf{A}_1\mathbf{x}_1^k + \mathbf{b}_1\right) + \mathbf{x}_1^k - \mathbf{x}_2^k \\ \mu_2^*\left(\mathbf{A}_2\mathbf{x}_2^k + \mathbf{b}_2\right) + \mathbf{x}_2^k - \mathbf{x}_1^k \end{pmatrix}. \tag{2.60}$$

The vector $-\mathbf{G}^*\left(\mathbf{z}^k\right)$ indicates the direction of descent in the 6-dimensional space . Thus, to compute $\mathbf{z}^{k+1}$, the following problem needs to be solved:

$$\text{Find } \beta^k \text{ such as } \Phi_\lambda(\mathbf{z}^k - \beta^k \mathbf{G}^*) = \min_{\beta \geq 0} \Phi_\lambda(\mathbf{z}^k - \beta \mathbf{G}^*). \tag{2.61}$$

The procedure to solve Problem (2.61) is explained in the next subsection.

2.4.3. Solving Problem (2.61)

In [21], β^k is said to be computed analytically, without further explanations. The procedure that has been applied in this work to compute β^k is thus explained in the next lines.

$$\Gamma : \beta \rightarrow \Phi_\lambda\left(\mathbf{z}^k - \beta \mathbf{G}^*\left(\mathbf{z}^k\right)\right) \tag{2.62}$$

must be computed explicitly and equaled to zero, providing an equation in β to be solved. This reasoning supposes that $\dfrac{d\Gamma}{d\beta}(\beta^k)$ is well defined, **i.e.** Φ_λ differentiable at $\mathbf{z}^k - \beta^k \mathbf{G}^*$. This is not necessarily the case, especially at the target point: indeed the minimum we are looking for is located on Z, where Φ_λ is not

differentiable. Nonetheless, we decide to ignore this for now and compute $\frac{d\Gamma}{d\beta}$ for β such that $h_1(\mathbf{x}_1^k - \beta \mathbf{G_1}^*) \neq 0$ and $h_2(\mathbf{x}_2^k - \beta \mathbf{G_2}^*) \neq 0$:

$$\begin{cases} \Gamma'(\beta) = \dfrac{\partial}{\partial \beta} f\left(\mathbf{z}^k - \beta \mathbf{G}^*\right) + \lambda \dfrac{\partial}{\partial \beta} \phi\left(\mathbf{z}^k - \beta \mathbf{G}^*\right) \\ \dfrac{\partial}{\partial \beta} f\left(\mathbf{z}^k - \beta \mathbf{G}^*\right) = -\left(\mathbf{G}_1^* - \mathbf{G}_2^*\right) \cdot \left(\mathbf{x}_1^k - \mathbf{x}_2^k\right) + \beta \left(\mathbf{G}_1^* - \mathbf{G}_2^*\right)^2 \\ \dfrac{\partial}{\partial \beta} \phi\left(\mathbf{z}^k - \beta \mathbf{G}^*\right) = \beta \displaystyle\sum_{m \in [\![1,2]\!]} \left[s_m(\beta) \left(\mathbf{G}_m^*\right)^\top \mathbf{A}_m \mathbf{G}_m^* \right] - \sum_{m \in [\![1,2]\!]} \left[s_m(\beta) \left(\left(\mathbf{G}_m^*\right)^\top \mathbf{A}_m \mathbf{x}_m^k + \left(\mathbf{G}_m^*\right)^\top \mathbf{b}_m \right) \right], \end{cases} \tag{2.63}$$

where

$$s_m(\beta) = \text{ the sign of } h_m\left(\mathbf{x}_m^k - \beta \mathbf{G}_m^*\right). \tag{2.64}$$

$\Gamma'(\beta) = 0$ implies that:

$$\beta = \frac{\left(\mathbf{G}_1^* - \mathbf{G}_2^*\right) \cdot \left(\mathbf{x}_1^k - \mathbf{x}_2^k\right) + \lambda \sum_{m \in [\![1,2]\!]} s_m(\beta) \left(\left(\mathbf{G}_m^*\right)^\top \mathbf{A}_m \mathbf{x}_m^k + \left(\mathbf{G}_m^*\right)^\top \mathbf{b}_m \right)}{\left(\mathbf{G}_1^* - \mathbf{G}_2^*\right)^2 + \lambda \sum_{m \in [\![1,2]\!]} s_m(\beta) \left(\mathbf{G}_m^*\right)^\top \mathbf{A}_m \mathbf{G}_m^*}. \tag{2.65}$$

Eq. (2.65) enables one to determine β^k. If Eq. (2.65) can not be solved, it is assumed that the minimum of Problem (2.61) is located where Γ is not differentiable, **i.e.** where:

$$h_1\left(\mathbf{x}_1^k - \beta \mathbf{G}_1^*\right) = 0 \text{ or } h_2\left(\mathbf{x}_2^k - \beta \mathbf{G}_2^*\right) = 0.$$

Then, $\left(\beta_m^k\right)_{m \in [\![1,2]\!]} \in \left(\mathbb{R}_+\right)^2$ are computed such that:

$$m \in [\![1,2]\!], \quad h_m\left(\mathbf{x}_m^k - \beta_m^k \mathbf{G}_m^*\right) = 0. \tag{2.66}$$

For $m \in [\![1,2]\!]$, it comes to determine the intersection of the surface ∂E_m with the ray starting from $\mathbf{x}_m^k$ and directed along $-\mathbf{G}_m^*$ (for how to proceed, see Appendix D). As two solutions can exist for the same β_m^k, the smallest positive admissible value for β_m^k in accordance with Eq. (2.66) is chosen. Then β^k is then chosen among $\left(\beta_m^k\right)_{m \in [\![1,2]\!]} \in \left(\mathbb{R}_+\right)^2$ as the one minimizing Γ (see Eq. (2.62)).

2.4.4. Convergence criterion

The algorithm ends when a local minimum of Φ_λ is reached, **i.e.** when:

$$\left\| \mathbf{G}^*(\mathbf{z}^k) \right\| < \epsilon_{\mathbf{G}}. \tag{2.67}$$

2.4.5. Ellipsoids overlap

Preliminary tests have shown that convergence issues could arise with overlapping ellipsoids. Since this algorithm is also time-consuming, it is desirable to first test for the overlap with Jia *et al.*'s analytical overlap test described in Appendix C. If no overlap is detected, the algorithm is run to compute the distance.

2.4.6. Summary of the numerical process

The parameters of the algorithm are recalled:

- the penalty constant λ
- the stopping criterion on the hypogradient norm $\epsilon_{\mathbf{G}}$

The numerical procedure of the Exterior Penalty Function Method is summarized in Algorithm 5.

Algorithm 5 Exterior Penalty Function Method

$k = 0$
if collision = .TRUE. (Collision Detection Algorithm Appendix C) **then**
  return distance = 0
end if
For $m \in [\![1, 2]\!]$, $\mathbf{x}_m^0 = [\mathbf{r}_1, \mathbf{r}_2] \cap \partial E_m$
$\mathbf{z}^0 = \left(\mathbf{x}_1^0, \mathbf{x}_2^0\right)^\top$
while Solution is not reached **do**
  Compute $\mathbf{Q}\left(\mathbf{z}^k\right)$, $\mathbf{h}_1'\left(\mathbf{x}_1^k\right)$ and $\mathbf{h}_2'\left(\mathbf{x}_2^k\right)$ with Eq. (2.54)
  Compute (μ_1^*, μ_2^*) by solving the system (2.58)
  Compute $\mathbf{G}^*\left(\mathbf{z}^k\right) = \begin{pmatrix} \mathbf{G}_1^*\left(\mathbf{x}_1^k\right) \\ \mathbf{G}_2^*\left(\mathbf{x}_2^k\right) \end{pmatrix}$ with Eq. (2.60)
  if $\left\|\mathbf{G}^*(\mathbf{z}^k)\right\| < \epsilon_{\mathbf{G}}$ **then**
    return distance $= \left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\|$
  end if
  Compute β^k by solving Problem (2.61) following 2.4.3
  $\mathbf{z}^{k+1} = \left(\mathbf{x}_1^k, \mathbf{x}_2^k\right)^\top = \mathbf{z}^k - \beta^k \mathbf{G}^*\left(\mathbf{z}^k\right)$
  $k \leftarrow k + 1$
end while

2.5. Jain *et al.*' algorithm [4]

A geometrical iterative algorithm has been suggested by Jain *et al.* [4], in an article that introduces a methodology to perform Particle-Resolved Direct Numerical Simulations with ellipsoidal particles. The method is quite simple, but no proof of convergence was given.

2.6. Geometrical procedure

This algorithm consists in parameterizing the surface of the ellipsoids with ellipsoidal coordinates $(\nu, \phi) \in [0, \pi] \times [-\pi, \pi]$. In its own frame reference, for an ellipsoid with semi axes (a, b, c), it yields:

$$\begin{cases} x = a\cos\phi\sin\nu \\ y = b\sin\phi\sin\nu \\ z = c\cos\nu. \end{cases} \tag{2.68}$$

Those relations can be inverted:

$$\begin{cases} \nu = \arccos\dfrac{z}{c} \\ \phi = \begin{cases} \arctan\dfrac{ay}{bx} & \text{if } x > 0 \\ \pi + \arctan\dfrac{ay}{bx} & \text{if } x < 0 \quad \text{and } y \geq 0 \\ -\pi + \arctan\dfrac{ay}{bx} & \text{if } x < 0 \quad \text{and } y < 0 \\ \dfrac{\pi}{2} & \text{if } x = 0 \quad \text{and } y > 0 \\ -\dfrac{\pi}{2} & \text{if } x = 0 \quad \text{and } y < 0. \end{cases} \end{cases} \tag{2.69}$$

At each iteration, by assuming the knowledge of the previous points $(\mathbf{x}_1^k, \mathbf{x}_2^k) \in E_1 \times E_2$, points $(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}) \in E_1 \times E_2$ are computed as follows:

$$\begin{cases} \nu_m^{k+1} = \nu_m^k + C_m \dfrac{\mathbf{d}_m^k \cdot \mathbf{t}_{\nu_m}^k}{||\mathbf{d}_m^k||.||\mathbf{t}_{\nu_m}^k||} \\ \phi_m^{k+1} = \phi_m^k + C_m \dfrac{\mathbf{d}_m^k \cdot \mathbf{t}_{\phi_m}^k}{||\mathbf{d}_m^k||.||\mathbf{t}_{\phi_m}^k||} \end{cases} \tag{2.70}$$

where:

- $m = 1, 2$ is referring to the considered ellipsoid.
- $\mathbf{t}_{\nu_m}^k$ (respectively $\mathbf{t}_{\phi_m}^k$) is the tangent vector to ∂E_m at point $\mathbf{x}_m^k$ obtained by differentiating Eq. (2.68) with respect to ν (respectively ϕ).
- C_m are constants, their choice is explained below.
- $\mathbf{d}_1^k = \mathbf{x}_2^k - \mathbf{x}_1^k$ and $\mathbf{d}_2^k = \mathbf{x}_1^k - \mathbf{x}_2^k$.

The algorithm is initialized in the same manner than the Moving Balls algorithm 2.1: $\mathbf{x}_m^0$ is chosen as the intersection point between the segment joining the two mass centers and the surface ∂E_m.

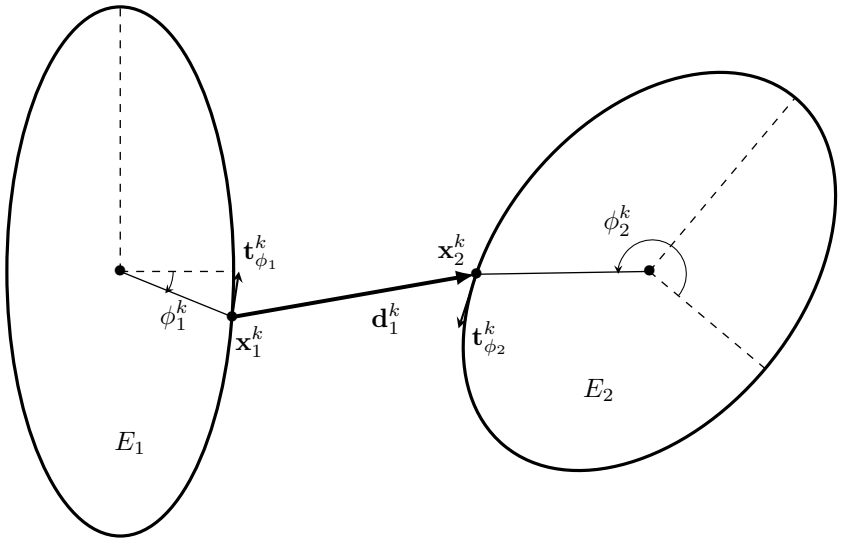


Figure 4: 2D illustration for the working of the algorithm proposed by Jain *et al.* [4].

2.6.1. Choice of the constants C_m

In Jain *et al.*' article [4], the constants $(C_m)_{m\in 1,2}$ are prescribed as

$$C_m^{\text{Jain}} = \frac{D_{eq,m}}{\max(a_m, b_m, c_m)} \tag{2.71}$$

where $D_{eq,m}$ is the volumetrically equivalent diameter of the ellipsoid E_m. The authors stated that it allows to take into account the increased curvature generated by non-sphericity.

2.6.2. Convergence criterion

The same criterion expressed in 2.1.1 for the Moving Balls algorithm is also used for this algorithm. A controlling parameter ϵ_d has to be used, which ensures that the error of the computed distance is inferior to ϵ_d.

2.6.3. Ellipsoids overlap

This algorithm does not handle the situation where E_1 and E_2 collide, so it has to be supplemented with an additional preliminary algorithm to detect a potential overlap. Jia's *et al.*'s analytical overlap test [26] is then run before executing the algorithm. This test is described in Appendix C. If no overlap is detected, the algorithm is run and the distance is computed.

2.6.4. Summary of the numerical process

In Algorithm 6, the numerical process of Jain *et al.*' method is summarized.

Algorithm 6 Jain *et al.*' Distance Algorithm

$k = 0$
if collision = .TRUE. (Collision Detection Algorithm Appendix C) **then**
  return distance = 0
end if
For $m \in [\![1,2]\!]$, $\mathbf{x}_m^0 = [\mathbf{r}_1, \mathbf{r}_2] \cap \partial E_m$
Compute $\left(\nu_m^0, \phi_m^0\right)_{[\![1,2]\!]}$ with Eq. (2.69)
Set ϵ_θ with Eq. (2.9)
Compute $\left(\theta_m^0\right)_{m\in[\![1,2]\!]}$ with Eq. (2.3)
while $\theta_1^k > \epsilon_\theta$ **or** $\theta_2^k > \epsilon_\theta$ **do**
  Compute $\left(\nu_m^{k+1}, \phi_m^{k+1}\right)_{[\![1,2]\!]}$ with Eq. (2.70)
  Deduce $\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right)$ with Eq. (2.68)
  Compute $\left(\theta_m^{k+1}\right)_{m\in[\![1,2]\!]}$ with Eq. (2.3)
  $k \leftarrow k + 1$
end while
return distance = $\left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\|$

3. Algorithms validation

3.1. Goals

It is aimed here to test and validate the algorithms described in 2, to ensure that they:

- Respect the desired precision.
- Are insensitive to the employed scale.
- Are not excessively time-consuming. When the CPU time of one method is considered, the GJK will be taken as reference.

A dedicated analytical example is set and described in 3.2. In 3.4, a precision study is realised for every method with this specific configuration, following a common methodology explained in 3.3. Additional considerations will be brought for the Newton-Coulomb method (3.4.3), the Exterior Penalty Function Method (3.4.4) and Jain's *et al.*'s method (3.4.5) to rule out these methods.

Note that the goal of this section is not to draw general conclusions about the methods, especially about CPU time (except when it is excessive): it is aimed in the following to carry out a first validation step, and to eliminate the methods that do not verify the above-mentioned points.

3.2. Description of the validation case

A two-ellipsoid configuration can be constructed where the solution of problem (1.3) is analytically known: the case where one ellipsoid is the symmetric of the other with respect to a plane. In this case, the problem

is equivalent to compute the distance of an ellipsoid to the plane of symmetry. This can be exactly solved thanks to *support mappings* introduced with the GJK algorithm in 2.2 and Appendix G.

An ellipsoid E_1 of semi-axes (a, b, b) is considered, with its equivalent diameter $D_{eq} = (abb)^{1/3}$ set to 1. The lengths (a, b) are thus completely determined by the aspect ratio $A_r = a/b$:

$$\begin{cases} a = A_r^{\frac{2}{3}} \dfrac{D_{eq}}{2} \\ b = A_r^{-\frac{1}{3}} \dfrac{D_{eq}}{2}. \end{cases} \tag{3.1}$$

Its mass center $\mathbf{r}_1$ can move along the x axis, and is then characterised by $x_c > 0$ (see Fig. 5). A rotation of $\dfrac{\pi}{4}$ around the z-axis is applied between its major axis and the x-axis. Its symmetric companion E_2 with respect to the yz plane is constructed. The exact solution of problem 1.3 for E_1 and E_2 is then:

$$\begin{cases} \mathbf{x}_1^* = \mathbf{s}_{E_1}(-\mathbf{n}) \\ \mathbf{x}_2^* = \mathbf{s}_{E_2}(\mathbf{n}) \\ d^* = ||\mathbf{x}_1^* - \mathbf{x}_2^*|| \end{cases} \tag{3.2}$$

where $\mathbf{n}$ is the normal vector to yz-plane directed towards E_1, and $\mathbf{s}_{E_m}$ is the support mapping of the ellipsoid E_m introduced in 2.2 and Appendix G.

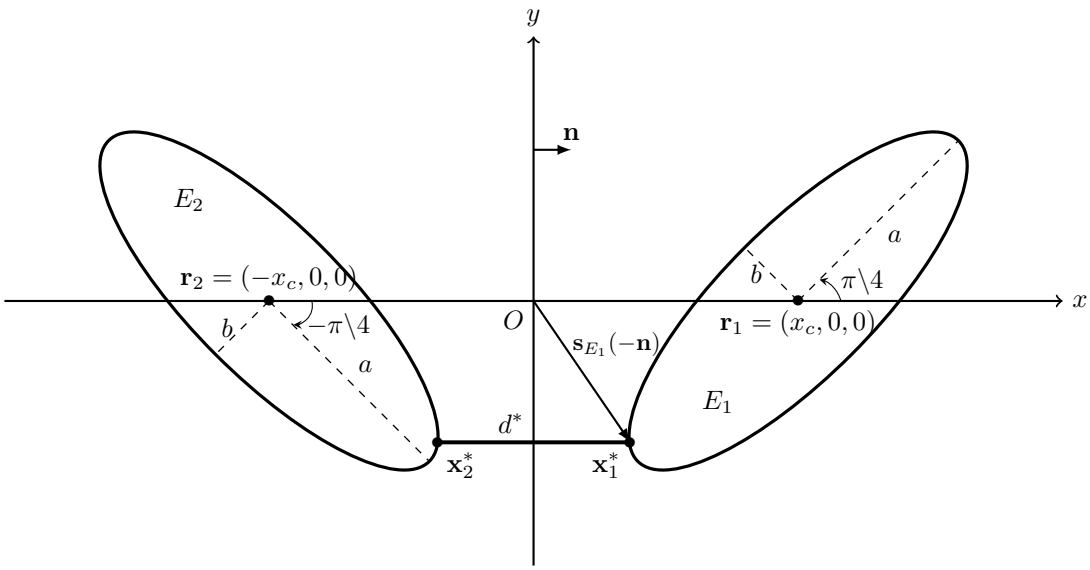


Figure 5: Description of the analytical validation case.

Three parameters characterize this two-ellipsoid configuration:

- the analytical distance d^* (or abscissa x_c).
- the aspect ratio $A_r = a/b$.
- a global dilation parameter Λ, acting on all the dimensions (D_{eq}, a, b, x_c, d^*) illustrated in Fig.5.

3.3. Approach for the numerical study on the validation case 3.2

The solution provided by a particular algorithm for the validation case 3.2 will be noted $(\mathbf{x}_1, \mathbf{x}_2, d = ||\mathbf{x}_2 - \mathbf{x}_1||)$.

For all the numerical tests in 3.4, lengths are then rendered dimensionless by dividing them with the dimension D_{eq}.

For a particular value of the aspect ratio $A_r = a/b$, the dilation parameter Λ, along with the internal parameters of one method fixed, the following quantities will be studied as functions of the exact distance d^*/D_{eq}:

- the accuracy of the distance $|d^* - d|/D_{eq}$.
- the accuracy of the minimizing points $\|\mathbf{x}_m - \mathbf{x}_m^*\|/D_{eq}$ where m is indifferently equal to 1 or 2 thanks to the symmetry of the considered configuration (see Fig.5).

The parameters related to the two-ellipsoids configuration 3.2 will be chosen as follows:

- the dilation parameter $\Lambda \in \left\{10^{-6}, 10^{-3}, 1, 10^3, 10^6\right\}$.
- the analytical distance $d^*/D_{eq} \in \left\{10^{-k} | k \in [\![0, 6]\!]\right\}$.
- the aspect ratio $A_r = a/b \in \{1/6\ ,\ 1/3\ ,\ 2/3\ ,\ 3/2\ ,3, 6\}$.

For each algorithm, the approach will be the same:

1. the first parameter to be varied will be the dilation parameter Λ, to ensure that the considered algorithm is independent of a particular choice of scale.
2. After the algorithm is proven to be scale independent, the value of Λ is set to 1, and all the others parameters of interest are then varied.

3.4. Numerical results

Following the approach described in 3.3, numerical results are presented for all algorithms detailed in 2. Indeed, a parametric study is carried for all algorithms by varying the dilation parameter Λ, the aspect ratio A_r, and the stopping parameter ϵ_d. Additional considerations are brought for the Newton-Coulomb method 3.4.3, the Exterior Penalty Function Method 3.4.4 and Jain *et al.*'s method 3.4.5.

3.4.1. Moving Balls algorithm

In this part, the numerical results concerning the validation case 3.2 are shown for the Moving Balls algorithm 2.1.

The only free parameter for this algorithm is the stopping parameter controlling the accuracy of the computed distance d.

Precision study with different values of the dilation parameter Λ

In Fig. 6, it is shown that for this particular example 3.2, the Moving Balls algorithm is fairly insensitive to the working scale. All the curves collapse into a single one for all considered values of the dilation parameter Λ.

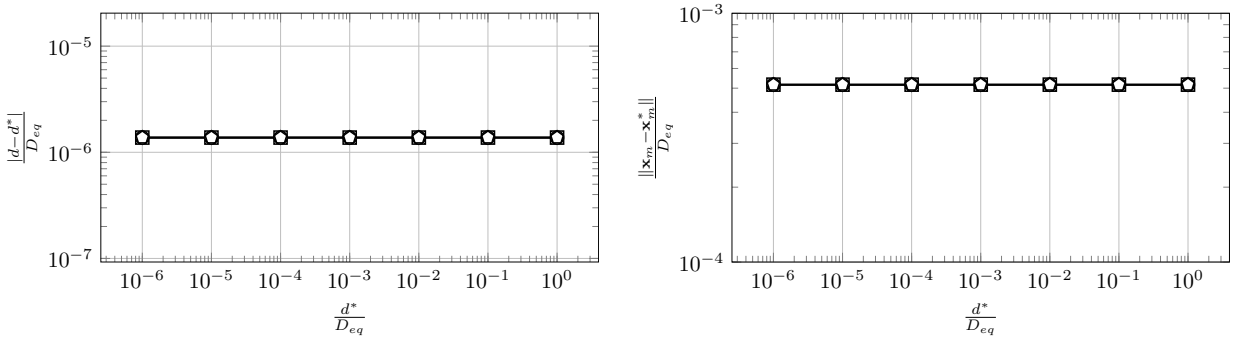


Figure 6: Quantities of interest against the distance d^*/D_{eq} for different values of the dilation parameter Λ: (□) $\Lambda = 10^{-6}$, (◇) $\Lambda = 10^{-3}$ (○) $\Lambda = 1$, (△) $\Lambda = 10^3$, (○) $\Lambda = 10^6$. A_r=3 and $\epsilon_d = 10^{-5} D_{eq}$.

Precision study with different values of the aspect ratio A_r

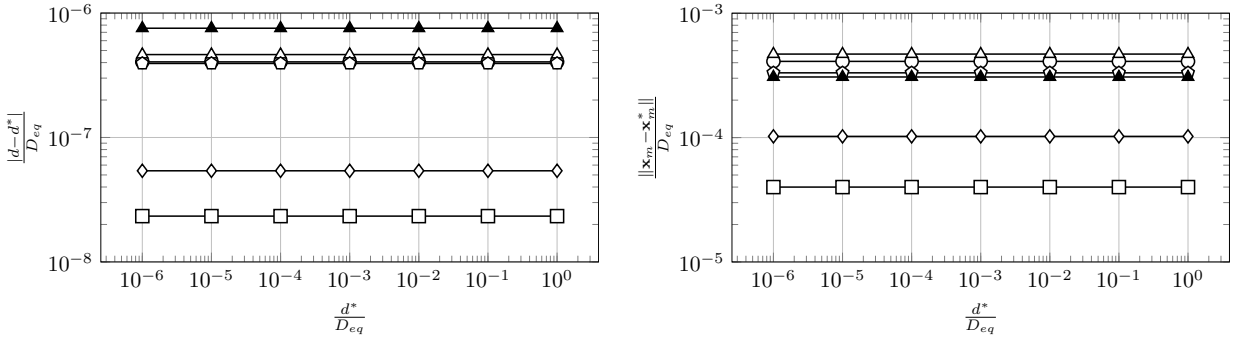


Figure 7: Quantities of interest against the distance d^*/D_{eq} for different values of the aspect ratio A_r: (□) $A_r = 1/6$, (◇) $A_r = 1/3$, (○) $A_r = 2/3$, (△) $A_r = 3/2$, (○) $A_r = 3$ and (▲) $A_r = 6$ The ϵ_d value is set to $10^{-6}D_{eq}$.

Fig. 7 shows that the Moving Balls algorithm always meet the desired inequality $|d - d^*| \leq \epsilon_d$ for any value of the aspect ratio A_r.

Precision study with different values of the stopping parameter ϵ_d

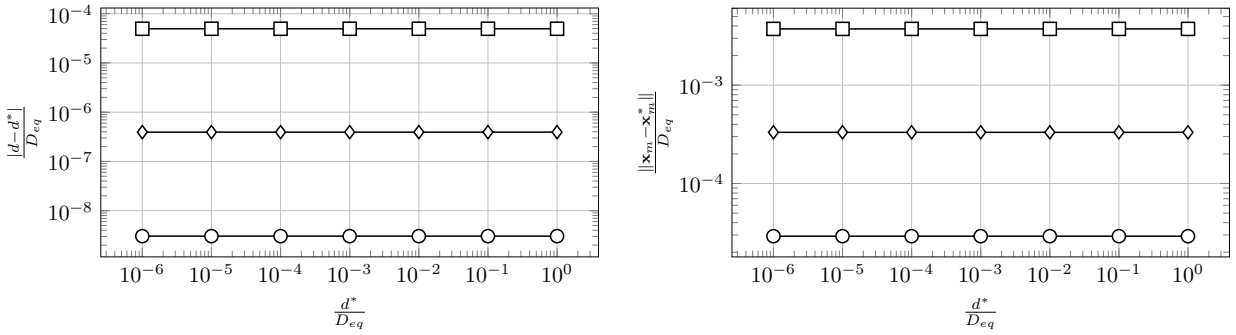


Figure 8: Quantities of interest against the distance d^*/D_{eq} for different values of the stopping parameter ϵ_d: (□) $\epsilon_b = 10^{-4}D_{eq}$, (◇) $\epsilon_d = 10^{-6}D_{eq}$ and (○) $\epsilon_d = 10^{-8}D_{eq}$. A_r is set to 3.

In Fig. 8, it can be verified that the desired inequality $|d - d^*| \leq \epsilon_d$ holds over the whole range of distances d^*/D_{eq}.

Conclusion

For this particular validation case, The Moving Balls algorithm meets all the required criteria expressed in 3.1.

3.4.2. GJK algorithm

In this paragraph, the results of the numerical experiments on the validation case 3.2 are shown, for the GJK algorithm 2.

The only free parameter of the GJK algorithm is the desired maximum error ϵ_d on the distance.

Note that if the initial vector $\mathbf{v}^0$ is chosen as the centers difference $\mathbf{r}_1 - \mathbf{r}_2$ as specified in Algorithm 2, then the GJK algorithm returns the analytical result within 1 iteration. That is why an arbitrary initial condition

$$\mathbf{v}^0 = (2x_c, 2b, 2b)^\top \in E_1 - E_2 \tag{3.3}$$

has been used in this subsection for this particular validation case.

Precision study for different values of the scaling parameter Λ

In Fig. 9, it is verified that the relative accuracy of the GJK algorithm is rigorously independent on the scaling parameter Λ with this particular example. All the curves $\text{error}/D_{eq} = f\,(d^*/D_{eq})$ collapse into a single one.

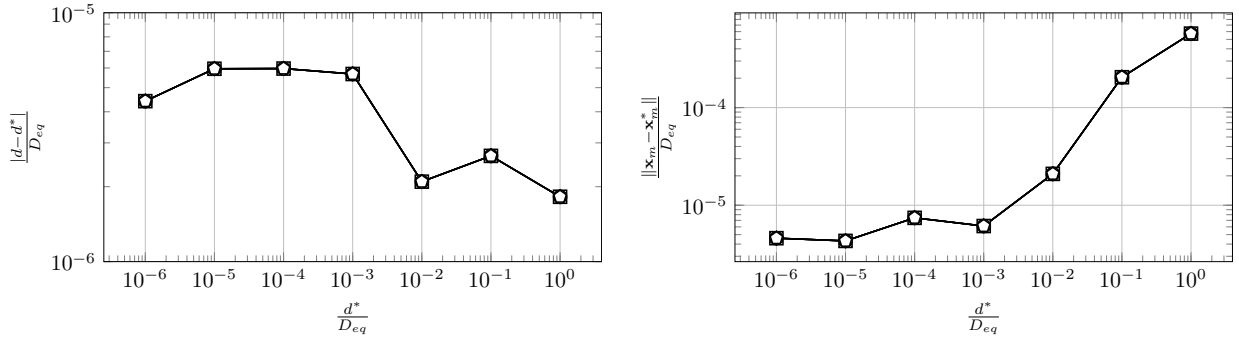


Figure 9: Quantities of interest against the distance d^*/D_{eq} for different values of the dilation parameter Λ: (□) $\Lambda = 10^{-6}$, (◇) $\Lambda = 10^{-3}$ (○) $\Lambda = 1$, (△) $\Lambda = 10^{3}$, (○) $\Lambda = 10^{6}$. A_r=3 and $\epsilon_d = 10^{-5} D_{eq}$.

Precision study with different values of the aspect ratio A_r

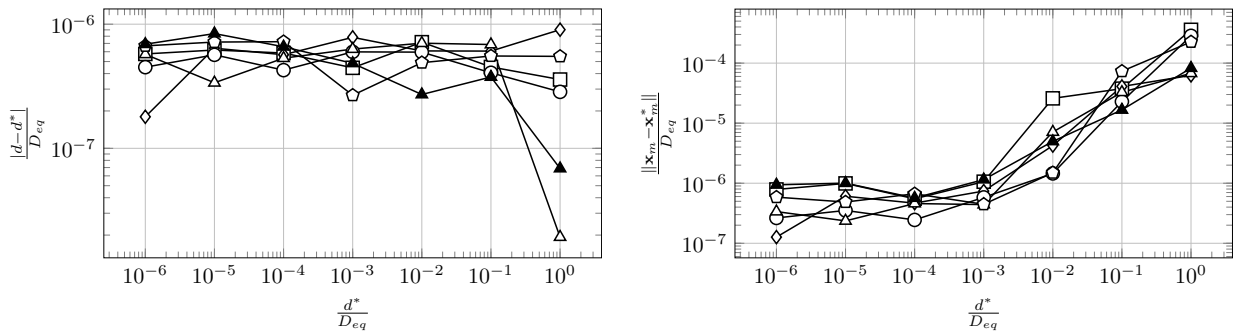


Figure 10: Quantities of interest against the distance d^*/D_{eq} for different values of the aspect ratio A_r: (□) $A_r = 1/6$, (◇) $A_r = 1/3$, (○) $A_r = 2/3$, (△) $A_r = 3/2$, (○) $A_r = 3$ and (▲) $A_r = 6$. ϵ_d is set to $10^{-6} D_{eq}$.

Figure 10 shows that the global precision is almost independent on the aspect ratio on this particular example. The precision of the distance is coherent with the assigned value of the parameter ϵ_d, meaning that the inequality $|d - d^*| \leq \epsilon_d$ holds. A peculiar behaviour is observed for the error $\|\mathbf{x}_m - \mathbf{x}_m^*\|/D_{eq}$ associated with the minimizing point $\mathbf{x}_m$ (right Fig. 10): this error increases with the distance d^*/D_{eq} while the distance error $|d - d^*|/D_{eq}$ (left Fig. 10) remains constant or even decreases with the distance d^*/D_{eq}. No explanation has been found.

Precision study with different values of the stopping criterion ϵ_d

Fig. 11 shows that the desired accuracy of the distance, set by ϵ_d, is always verified (*i.e.* $|d - d^*| < \epsilon_d$). The same curious trend observed in the previous test is also observed here: when the distance d^*/D_{eq} increases, the distance error $|d - d^*|/D_{eq}$ stays constant but the point error $\|\mathbf{x}_m - \mathbf{x}_m^*\|/D_{eq}$ increases.

Conclusion

For this particular validation case, the GJK algorithm meets the required criteria expressed in 3.1. An unexplained opposite behavior has been observed between the distance error and the minimizing point error. Nonetheless, it is worth noting that the particular initial condition used in this numerical analysis (Eq. (3.3)) can be the origin of such behavior.

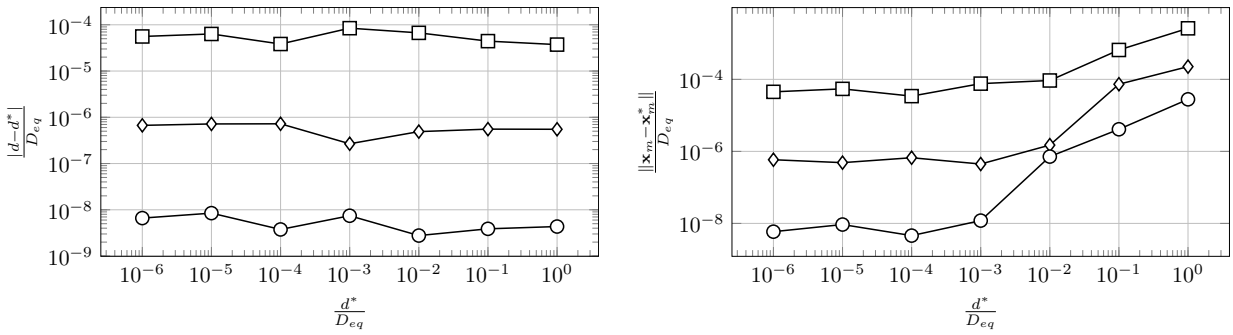


Figure 11: Quantities of interest against the distance d^*/D_{eq} for different values of the stopping parameter ϵ_d: (□) $\epsilon_d = 10^{-4}D_{eq}$, (◇) $\epsilon_d = 10^{-6}D_{eq}$ and (○) $\epsilon_d = 10^{-8}D_{eq}$. A_r is set to 3.

3.4.3. Newton-Coulomb Method

In this part, the results of the numerical experiments on the validation case 3.2 are shown, for the Newton-Coulomb method 4.

The parameters of the Newton-Coulomb method are recalled:

- ϵ_d the stopping parameter limiting the distance error.
- $C_{\max}$, being the maximum value of the $C^k_{\text{curv},m}$ coefficients (Eq. (2.45)).
- τ, being the parameter controlling the magnitude of $C^k_{\text{curv},m}$ in small-distance configurations (Eq. (2.45)).
- λ, being the shrinking factor appearing in the backtracking loop.

Calibration of $C_{\max}$, τ and λ

These three internal parameters need to be calibrated. Until further notice, their value is set to:

- $C_{\max} = 0.01$
- $\tau = 1$
- $\lambda = 0.5$

Precision study with different values of the dilation parameter Λ

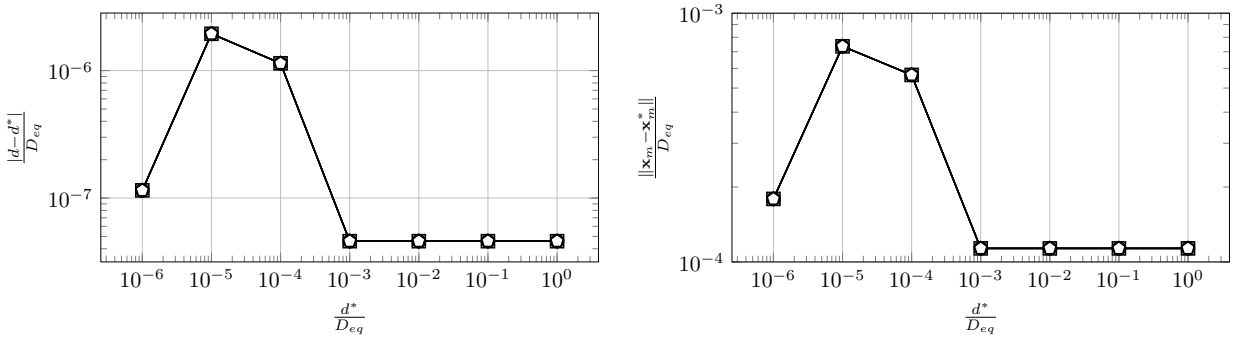


Figure 12: Quantities of interest against the distance d^*/D_{eq} for different values of the dilation parameter Λ: (□) $\Lambda = 10^{-6}$, (◇) $\Lambda = 10^{-3}$ (○) $\Lambda = 1$, (△) $\Lambda = 10^3$, (○) $\Lambda = 10^6$. A_r=3 and $\epsilon_d = 10^{-5}D_{eq}$

In Fig. 12, it is shown that algorithm is independent of the dilation parameter Λ, since here again all the curves $|d - d^*|/D_{eq} = f(d^*/D_{eq})$ collapse into one. This is an expected consequence of the modifications

brought to the method in 2.3.2, because the displacement at each iteration is scaled proportionally to the dimensions of the ellipsoid (see Eq. (2.40) and Eq. (2.41)).

Precision study with different values of the aspect ratio A_r

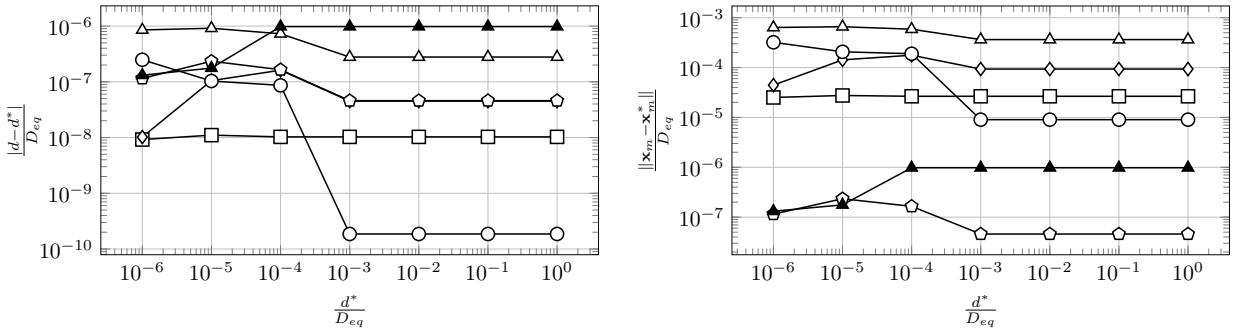


Figure 13: Quantities of interest against the distance d^*/D_{eq} for different values of the aspect ratio A_r: (□) $A_r = 1/6$, (◇) $A_r = 1/3$, (○) $A_r = 2/3$, (△) $A_r = 3/2$, (○) $A_r = 3$, and (▲) $A_r = 6$. ϵ_d is set to $10^{-6}D_{eq}$

Figure 13 shows the distance error $|d - d^*|$ stays inferior to the desired limit ϵ_d, for all aspect ratios A_r and distances d^*/D_{eq} considered. High variations of both errors with the distance d^*/D_{eq} are observed.

Precision study with different values of the stopping parameter ϵ_d

Figure 14 shows that that for several values of ϵ_d, the desired inequality $|d^* - d| \leq \epsilon_d$ holds over the range of tested distances $\dfrac{d^*}{D_{eq}}$.

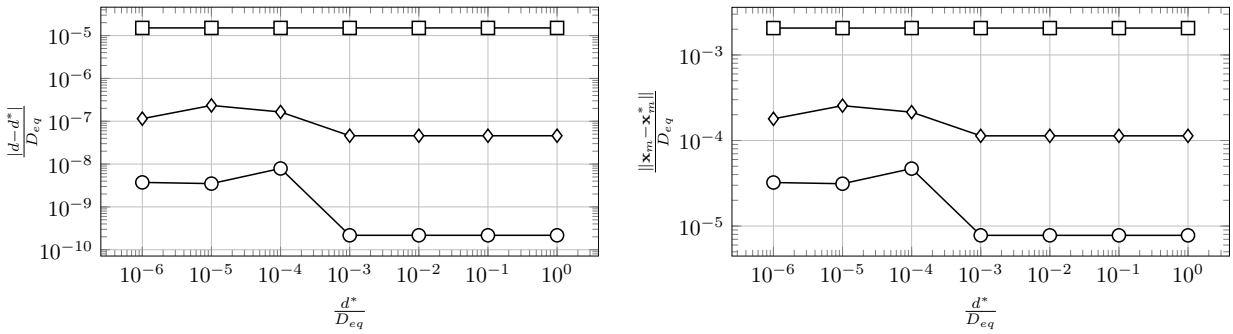


Figure 14: Quantities of interest against the distance d^*/D_{eq} for different values of the stopping parameter ϵ_d: (□) $\epsilon_d = 10^{-4}D_{eq}$, (◇) $\epsilon_d = 10^{-6}D_{eq}$ and (○) $\epsilon_d = 10^{-8}D_{eq}$. A_r is set to 3.

Comparison of the computing-cost with the GJK algorithm

Despite good performances in terms of computing time on the validation case (not shown) compared to widely used algorithms such as the MB algorithm or the GJK algorithm, the implementation 4 of the Newton-Coulomb method revealed to be disappointing *a posteriori* in terms of both computing-time and robustness. The method also suffers from an high number of parameters that makes it tedious to calibrate.

To illustrate these difficulties, the computing-time of the Newton-Coulomb Method has been compared to the GJK algorithm on a small randomly-generated array of mono-dispersed spheroids with aspect ratio $A_r = \dfrac{1}{6}$ (see 4.1.1 for the details regarding the array generation). The values of the parameters that have been used in this comparison are compiled in Table 1, as well as the results of the comparison.

Both algorithms	
ϵ_d	$10^{-5}R_{eq}$
NCM	
$C_{\max}$	0.01
τ	0.01
λ	0.5
Array parameters	
aspect ratio A_r	1/6
array size L/D_{eq}	5
volume fraction α_p	0.25
Test results	
Number of distance queries	606
Number of convergence failures	3
$\frac{\text{computing-time NCM}}{\text{computing-time GJK}}$	216

Table 1: Computing-time comparison between the GJK algorithm and Newton-Coulomb method: parameters and results.

The parameters of the Newton-Coulomb method have been chosen to enable a consequent number of distance queries to succeed (small values for $C_{\max}$ and τ). Yet, on the total number of 606 distance queries, the algorithm failed to converge 3 times (a limit has been set to 10^5 iterations). In the meanwhile, the Newton-Comlomb method has spent 216 times more CPU time than the GJK algorithm to solve these 606 distance queries.

This overhead can be reduced by increasing $C_{\max}$ and τ, but the number of problematic distance queries will also increase.

For informative purposes, one among the 3 problematic two-ellipsoid configurations is reported:

	$m=1$	$m=2$
a_m	$A_r^{\frac{2}{3}}\ D_{eq}/2$	
b_m	$A_r^{-\frac{1}{3}}\ D_{eq}/2$	
c_m	$A_r^{-\frac{1}{3}}\ D_{eq}/2$	
$\mathbf{r}_m^\top$	$(2.933, 8.555, 0.831)$	$(0.231, 9.120, 1.451)$
$q_{p,m}$	$(0.116, -0.561, -0.087, -0.815)$	$(0.351, -0.701, -0.525, -0.332)$

Table 2: Two-ellipsoid configuration where the Newton-Coulomb fails to converge within 10^5 iterations.

Conclusion

The numerical results obtained with the validation case 3.2 did not disqualify the Newton-Coulomb Method. However, this algorithm will not be considered in the statistical study 4, for the following reasons:

- high number of parameters that makes the method tedious to calibrate.
- robustness issues.
- high computing-time in comparison with the widely used GJK algorithm.

However, it is hoped that the proposed implementation of this paper can constitute a basis for one willing to improve the method.

3.4.4. Exterior Penalty Function method

In this part, the results of the numerical experiments for the validation case 3.2 are shown, for the Exterior Penalty Function Method 2.4.

The parameters of the algorithm are recalled:

- the penalty constant λ.
- the stopping criterion limiting the hypo-gradient norm $\epsilon_{\mathbf{G}}$.

Precision study with different values of the dilation parameter Λ

It has been tried to scale the internal parameters λ and $\epsilon_{\mathbf{G}}$ following dimensional analysis. Indeed, these parameters have the following dimensions:

- $[\lambda] = \text{length}^2$, as can be seen in Eq. (2.52).
- $[\epsilon_{\mathbf{G}}] = \text{length}$, as can be seen in Eq. (2.59).

Thus, it would seem natural to set these parameters λ and $\epsilon_{\mathbf{G}}$ proportional respectively to D_{eq}^2 and D_{eq}, following the convention that has been used so far to work with dimensionless lengths. By doing so, the obtained results are shown in Fig.15. Note that the values of the dilation parameter Λ chosen for this figure differs from the ones used so far for the other algorithms in Fig. 6, 9 and 12, because for certain of these values the convergence of the Exterior Penalty Function Method is not obtained.

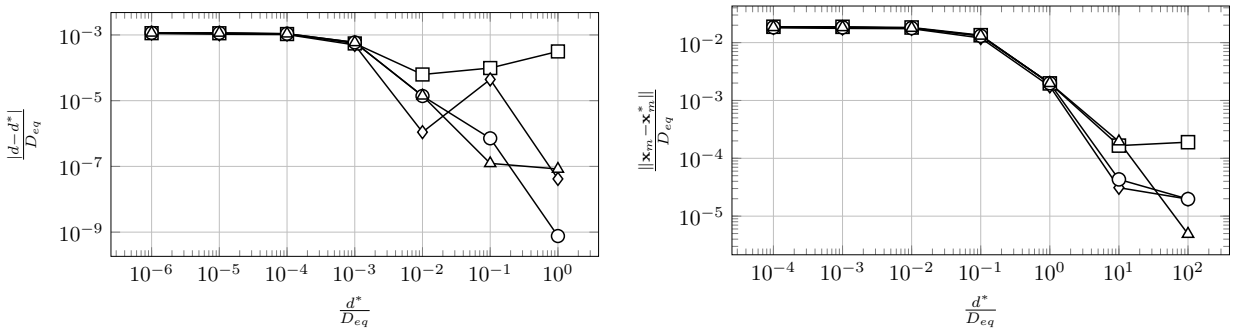


Figure 15: Quantities of interest against the distance d^*/D_{eq} for different values of the dilation parameter Λ: (□) $\Lambda = 10^{-3}$, (◇) $\Lambda = 1$, (○) $\Lambda = 10$ and (△) $\Lambda = 10^2$. The $\epsilon_{\mathbf{G}}$ value is set to $10^{-4}D_{eq}$, and the λ constant is set to $10^3 D_{eq}^2$.

It can be seen that the results of the precision study differ when changing the value of Λ. For the previously used values $\Lambda = 10^{-6}$ or 10^3, the convergence of the Exterior Penalty Function Method is not obtained. These observations demonstrate that the adequate scaling of the internal parameters for the implementation of this algorithm described in 2.4 is not trivial.

Nonetheless, the parameter study is carried out in the following paragraphs with $\Lambda = 1$.

Precision study with different values of the aspect ratios A_r

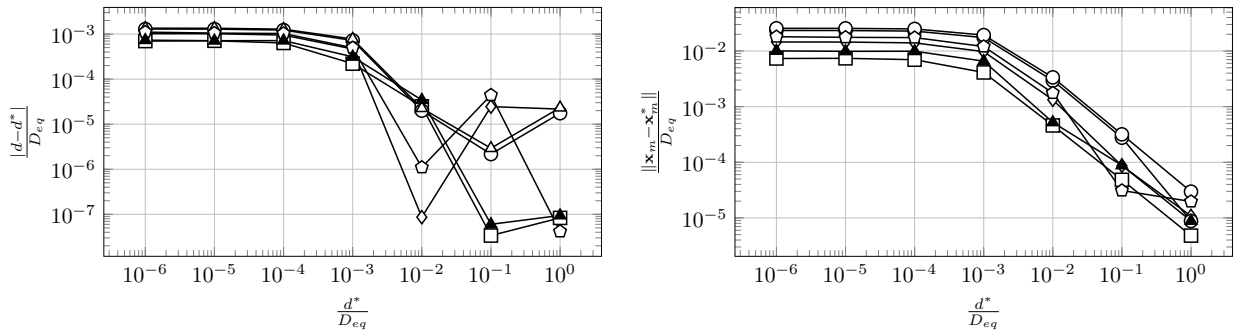


Figure 16: Quantities of interest against the distance d^*/D_{eq} for different values of the aspect ratio A_r: (□) $A_r = 1/6$, (◇) $A_r = 1/3$, (○) $A_r = 2/3$, (△) $A_r = 3/2$, (⬠) $A_r = 3$, and (▲) $A_r = 6$. $\epsilon_{\mathbf{G}}$ is set to $10^{-4}D_{eq}$, and the λ constant is set to $10^4 D_{eq}^2$.

In Fig. 16, several curves $|d - d^*|/D_{eq} = f\,(d^*/D_{eq})$ and $\|\mathbf{x}_m - \mathbf{x}_m^*\|/D_{eq} = f\,(d^*/D_{eq})$ are plotted, each one corresponding to a particular aspect ratio A_r. All these curves illustrates a loss of precision when the distance tends to zero.

Precision study with different values of the stopping parameter $\epsilon_{\mathbf{G}}$

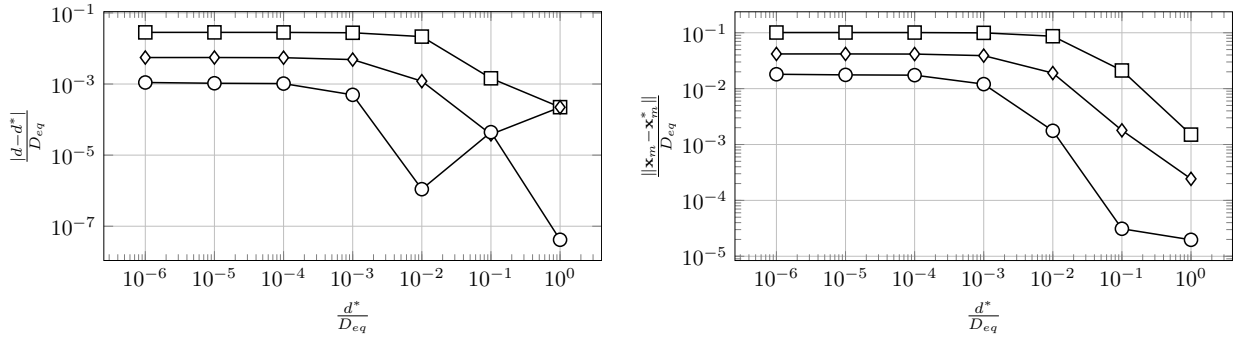


Figure 17: Quantities of interest against the distance d^*/D_{eq} for different values of the stopping parameter $\epsilon_{\mathbf{G}}$: (□) $\epsilon_{\mathbf{G}} = 10^{-2}D_{eq}$, (◇) $\epsilon_{\mathbf{G}} = 10^{-3}D_{eq}$ and (○) $\epsilon_{\mathbf{G}} = 10^{-4}D_{eq}$. A_r is set to 3 and λ to $10^3 D_{eq}^2$.

In Fig. 17, different curves $|d - d^*|/D_{eq} = f\,(d^*/D_{eq})$ and $\|\mathbf{x}_m - \mathbf{x}_m^*\|/D_{eq} = f\,(d^*/D_{eq})$ are plotted for different values of the stopping parameter $\epsilon_{\mathbf{G}}$. It can be seen that the error decreases slowly when decreasing the value of $\epsilon_{\mathbf{G}}$. It seems, that this algorithm cannot reach high precision for distances $d^*/D_{eq} \leq 10^{-3}$, since further decreasing the value of $\epsilon_{\mathbf{G}}$ (below $10^{-4}D_{eq}$) can generates convergence issues. This is a serious drawback, in comparison with the algorithms studied in 3.4.1 and 3.4.2 that can easily reach far lower errors. Fig. 17 also shows that with the smallest possible value of $\epsilon_{\mathbf{G}} = 10^{-4}D_{eq}$ (below this value the algorithm does not converge), the error on the distance for small d^*/D_{eq} cannot reach the level of accuracy easily attained with the Moving Balls algorithm for instance (Fig. 8).

Precision study with different values of the penalty constant λ

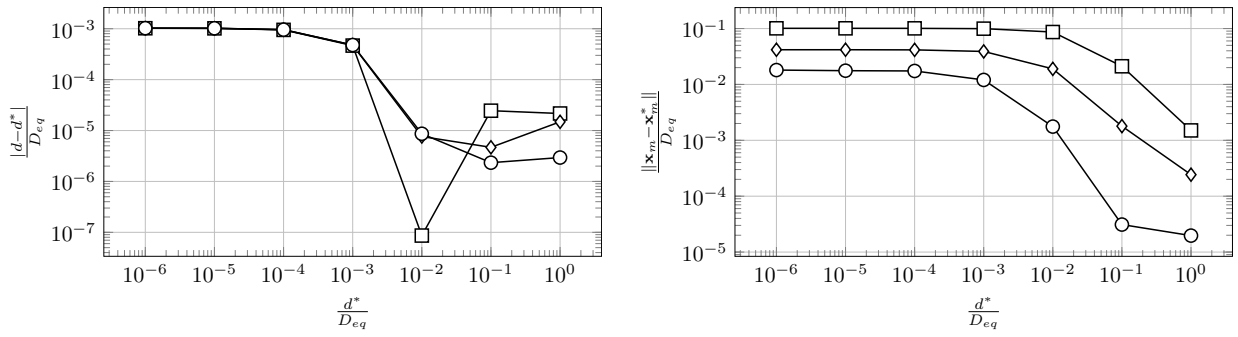


Figure 18: Quantities of interest against the distance d^*/D_{eq} for different values of the penalty constant λ: (□) $\lambda = 1000$, (◇) $\lambda = 5000$ and (○) $\lambda = 10.000$. A_r is set to 3 and $\epsilon_{\mathbf{G}}$ to $10^{-4}D_{eq}$.

Figure 18 shows that, on this particular validation case, the committed error on the solution seems to be weekly dependant on the value of the penalty constant λ. This constant must then be chosen large enough to ensure the convergence and ensure the constraint to be satisfied, but not too large to avoid a unreasonnable computational cost.

Computing time comparison with the GJK algorithm

This algorithm is found to be very time-consuming. To illustrate this fact, it is shown in Fig. 19 that in this particular example the Exterior Penalty Function method is two to four orders more time-consuming than the widely used GJK algorithm, with parameters chosen so that the GJK algorithm is way more accurate ($\epsilon_d = 10^{-6}D_{eq}$) than the Exterior Penalty Function Method over the whole range of tested distances.

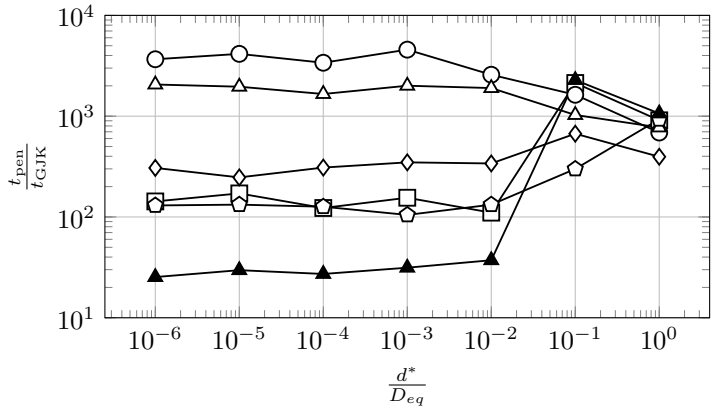


Figure 19: Computing time ratio between the Exterior Penalty Function method and the GJK algorithm $t_{\text{pen}}/t_{\text{GJK}}$ against the exact distance d^*/D_{eq}, for different values of the aspect ratio A_r: (□) $A_r = 1/6$, (◇) $A_r = 1/3$, (○) $A_r = 2/3$, (△) $A_r = 3/2$, (⬠) $A_r = 3$, and (▲) $A_r = 6$. $\epsilon_{\mathbf{G}} = 10^{-4}D_{eq}$, $\lambda = 10^3 D_{eq}^2$ for the Exterior Penalty Function Method, and $\epsilon_d = 10^{-6}D_{eq}$ for the GJK algorithm.

Conclusion

Different reasons make the Exterior Penalty Function Method a less viable option, and thus it will not appear in the statistical study 4. Indeed:

- The choice for the internal parameters λ and $\epsilon_{\mathbf{G}}$ is not obvious, in particular in order to make the algorithm insensitive to the dilation parameter Λ.
- It is difficult to obtain high accuracy in comparison with other methods.
- The computing time is excessive in comparison with other methods such as the GJK algorithm.

3.4.5. Jain *et al.*'s algorithm

In this part, some numerical results on the validation case 3.2 are presented, for Jain *et al.*'s algorithm.

It is recalled that this algorithm features the constants $(C_m)_{m\in[\![1,2]\!]}$ appearing in (2.70). In the original article [4], these constants are set with Eq. (2.71).

It has been noticed that Jain *et al.*'s algorithm fails to converge on the validation case for small distances between the two ellipsoids. To address this issue, it has been first tried to reduce the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants with respect to the scaling (2.70) introduced by Jain *et al.* [4]. Nonetheless, as will be shown in the next paragraphs, this measure is not sufficient to correct this robustness issue.

Minimal distance of convergence with different values of the aspect ratio A_r, and different values of the $(C_m)_{m\in[\![1,2]\!]}$ constants

Since Jain *et al.*'s algorithm does not converge for all investigated distances d^*/D_{eq}, full curves similar to the ones presented in Figs. 7, 10, 13 and 16 for the previous algorithms are not reported. Instead, the observed distance d^* below which Jain *et al.*'s algorithm fails to converge on the validation case is reported, with different values of the aspect ratio A_r, and different values for the $(C_m)_{m\in[\![1,2]\!]}$ constants. It is observed that prescribing a value of $(C_m)_{m\in[\![1,2]\!]}$ twice as small as in Eq. (2.71) greatly enhances the range of convergence of the algorithm (the algorithm converges down to smaller distances d^*) for all aspect ratios, while still not covering the full range of investigated distances d^*/D_{eq} (conversely to the Moving Ball algorithm in Fig. 7 or to the GJK algorithm in Fig. 10 for instance). Curiously, further diminishing the value of $(C_m)_{m\in[\![1,2]\!]}$ increases the value of the minimal distance of convergence up to an apparent plateau. This effect is thought to be very specific to this two-ellipsoid configuration.

A_r \ C_m/C_m^{Jain}	1	1/2	1/4	1/10	1/20
1/6	10^{-1}	10^{-2}	10^{-3}	10^{-2}	10^{-3}
1/3	10^{0}	10^{-4}	10^{-2}	10^{-2}	10^{-2}
2/3	10^{0}	10^{-6}	10^{-2}	10^{-2}	10^{-2}
3/2	10^{-1}	10^{-4}	10^{-2}	10^{-2}	10^{-2}
3	10^{-2}	10^{-3}	10^{-2}	10^{-2}	10^{-2}
6	10^{0}	10^{-3}	10^{-2}	10^{-2}	10^{-2}

Table 3: Value of d^*/D_{eq} below which Jain *et al.*'s algorithm is observed not to converge, for different values of the aspect ratio A_r and different values of the $(C_m)_{m\in[\![1,2]\!]}$. ϵ_d/D_{eq} is set to 10^{-6}.

Minimal distance of convergence with different values of the stopping parameter ϵ_d, and different values of the $(C_m)_{m\in[\![1,2]\!]}$ constants

As in the previous paragraph, we report in Table 4 the observed distance d^* below which Jain *et al.*'s algorithm fails to converge on the validation case, but this time with different values of the stopping parameter ϵ_d, and different values for the $(C_m)_{m\in[\![1,2]\!]}$ constants. Firstly, for the value of C_m prescribed in Eq. (2.71), it is observed that the smaller the stopping criterion ϵ_d, the bigger the minimal distance of convergence. Secondly, the same trend than in Table 3 is observed when diminishing the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants: the minimal distance of convergence d^* goes smaller when dividing the value C_m^{Jain} by two, but then increases when further decreasing the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants.

ϵ_d/D_{eq} \ C_m/C_m^{Jain}	1	1/2	1/4	1/10	1/20
10^{-4}	10^{-3}	10^{-4}	10^{-3}	10^{-2}	10^{-2}
10^{-6}	10^{-2}	10^{-3}	10^{-2}	10^{-2}	10^{-2}
10^{-8}	10^{-1}	10^{-2}	10^{-2}	10^{-2}	10^{-2}

Table 4: Value of d^*/D_{eq} below which Jain *et al.*'s algorithm is observed not to converge, for different values of the stopping parameter ϵ_d and different values of the $(C_m)_{m\in[\![1,2]\!]}$. The aspect ratio A_r is set to 3.

Therefore, a lack of robustness seems to be exhibited by these two last paragraphs. To further investigate this point, the algorithm has been tested on a small randomly-generated array of ellipsoids to observe the algorithm behaviour in a more practical context. The results are shown in the next paragraph.

Test on a small randomly-generated array, and computing cost comparison with the GJK algorithm

Jain *et al.*'s algorithm has been run on a small randomly-generated array of mono-dispersed spheroids with aspect ratio $A_r = 1/6$ (see 4.1.1 for the details regarding the array generation). Note that this choice for the aspect ratio is arbitrary. The value of the $(C_m)_{m\in[\![1,2]\!]}$ has been varied, and for each value all distance queries in the array are treated (successfully or not) between pairs of spheroids that verify an augmented bounding sphere test (4.2). In Table 5 are reported the percentage of convergence failure of Jain *et al.*'s algorithm and the computing cost comparison with the GJK algorithm for the successful distance queries. All involved parameters are also reported. These results exhibits a persistent robustness issue of Jain *et al.*'s algorithm despite considerably diminishing the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants with respect to the value C_m^{Jain} (2.71) recommended by Jain *et al.* [4]. Moreover, decreasing the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants increases the computing cost of Jain *et al.*'s algorithm, as shows the computing cost comparison with the GJK algorithm.

Array parameters					
aspect ratio A_r	1/6				
array size L/D_{eq}	5				
volume fraction α_p	0.25				
Algorithm parameters					
ϵ_d	$10^{-5}R_{eq}$				
C_m/C_m^{Jain}	1/10	1/20	1/40	1/100	1/200
Numerical results					
number of distance queries	606				
% of convergence failure	22	13	9	5	3
$\frac{\text{computing time Jain } et\ al.}{\text{computing time GJK}}$	2.0	3.9	7.4	18	35

Table 5: Numerical test of Jain *et al.* algorithm on a small randomly-generated array of oblate spheroids, and computing cost comparison with the GJK algorithm for the successful distance queries. Both algorithms are set with $\epsilon_d = 10^{-5}R_{eq}$. The value of the $(C_m)_{m\in[\![1,2]\!]}$ constants in Jain *et al.* algorithm is varied.

To overcome this lack of robustness while maintaining a small computing time, it has been tried to adaptively scale at each iteration the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants with a "backtracking loop", on the

model of the one introduced by Abbasov [20]. This "backtracking loop" consists of the following step at each iteration k:

- at the beginning of the iteration k, an initial value $C_m = C_m^{\text{Jain}}$ (Eq. (2.71)) is used to compute a new provisional distance vector.
- if the norm of this provisional distance vector is not inferior to the distance d^{k-1} computed at the previous iteration $k-1$, a smaller value λC_m is used to compute a second provisional distance vector.
- if the norm of this second provisional distance vector is not inferior to the distance d^{k-1}, the process is repeated with a value $\lambda^l C_m$ (l is the index of the backtracking loop) until the new provisional distance vector becomes smaller than the distance d^{k-1}.

This attempt of improvement revealed unfruitful, and the reason for this has been analysed on the exemple pictured in Fig. 20. For this example, Jain *et al.*'s algorithm does not converge with the $(C_m)_{m\in[\![1,2]\!]}$ constants set with Eq. (2.71). It has been observed that a value $C_m = C_m^{\text{Jain}}/3$ enables the algorithm to converge (without resorting to the backtracking loop). However, when run with the backtracking loop, the algorithm becomes trapped in it, because at some iteration k, the algorithm fails to generate new points $\left(\mathbf{x}_1^k, \mathbf{x}_2^k\right) \in \partial E_1 \times \partial E_2$ that verifies

$$\left\|\mathbf{x}_1^k - \mathbf{x}_2^k\right\| \leq \left\|\mathbf{x}_1^{k-1} - \mathbf{x}_2^{k-1}\right\|. \tag{3.4}$$

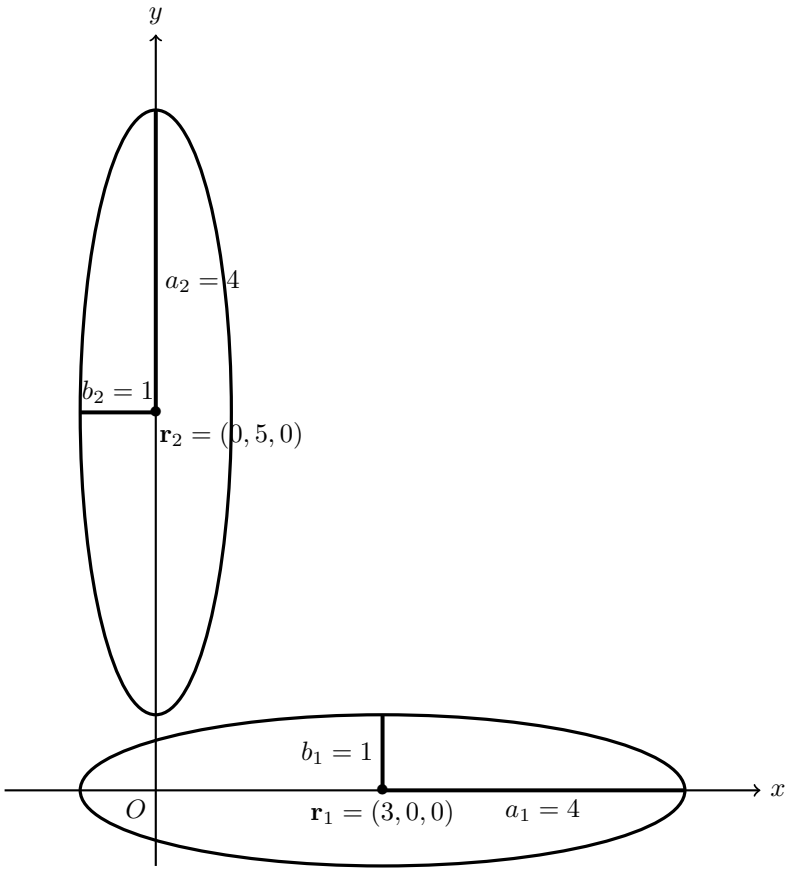


Figure 20: A configuration where the algorithm proposed by Jain *et al.* [4] does not converge. The third axis of both ellipsoids is set to $c_1 = c_2 = 1$.

Conclusion

Jain *et al.* will not be considered in the statistical study 4 for the following reasons:

- convergence issues on the validation case in the regime of small distances, despite diminishing the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants with respect to the value (2.71) introduced in [4].
- convergence issues observed on solving distance queries inside a moderately dense array of oblate spheroids. Diminishing the value of the $(C_m)_{m\in[\![1,2]\!]}$ constants enables to improve the ratio of converged distance queries. However, the computing cost then becomes way bigger than widely known algorithms such that the GJK algorithm.
- adding a backtracking loop does not solve the problem of appropriately scaling the $(C_m)_{m\in[\![1,2]\!]}$ due to the non-monotonic character of the algorithm.

3.5. Conclusions

The purpose of this study was to assess the ability of the methods to satisfy criteria expressed in 3.1, on a particular two-ellipsoid configuration 3.2. In this regard, the Moving Balls 2.1 and GJK 2.2 algorithms successfully passed this test; these two algorithms will be further investigated in the next section.

The Newton-Coulomb method 2.3, in its modified version 2.3.3, is also shown to successfully pass this simple validation case. Nonetheless, this algorithm revealed unusable *a posteriori* when studied on randomly-generated arrays of ellipsoids, because of a high number of parameters to calibrate, robustness issues, and high computation-time.

The Exterior Penalty Function Method 2.4 was ruled out by this first test. Indeed, no adequate scaling of the method's internal parameters was found to obtain scale independence, and the error control was shown to be poor. Even if considerations about computing time were not conducted on the validation case in this section (because general conclusions could not be made with a particular example), an exception was made for this algorithm to illustrate its excessive computing cost.

Finally, Jain *et al.*' algorithm 2.5 was shown to suffer from a lack of robustness. This problem is shown to be partly solved by reducing the value of the constants controlling the step size at each iteration, but at the detriment of the computing cost that becomes prohibitively slow compared to the GJK algorithm. Moreover, adding a backtracking loop to ensure robustness and a small computing cost is not possible due to the non-monotonic character of the algorithm.

4. Statistical study of the computing time on randomly-generated arrays of ellipsoids

In the last Section, the algorithms described in Section 2 were studied on a particular two-ellipsoid configuration 3.2. No considerations were given regarding the computing time on this validation case, since any conclusion would have been limited to this particular configuration (exception is found in 3.4.4, to illustrate the excessive computation cost of the Exterior Penalty Function method 2.4).

In this section, it is therefore proposed to study the methods validated in 3.2 (*i.e.* the Moving Balls 2.1 and GJK 2.2 algorithms) on randomly-generated arrays of mono-dispersed spheroids, in order to assess their robustness and make statistically true conclusions about the computing cost of these methods.

Firstly, the methodology used to carry this computing-time study is detailed in 4.1. Thereafter, a statistical convergence study is led algorithm by algorithm in 4.2, to determine the minimal size of the arrays that necessitates being used to get statistically significant results. Finally, the computing-costs of the algorithms are compared in 4.3.

4.1. Methodology

In this subsection, the methodology that will be applied to compare the computing times of the algorithms is described.

In 4.1.1, the characteristics of the randomly-generated arrays are given.

In 4.1.2, the approach to measure a mean computing-cost that is statistically significant is detailed.

4.1.1. Arrays generation

For all situations, the equivalent diameter D_{eq} of the ellipsoids is kept constant for every aspect ratio A_r. The semi-axes lengths (a, b, b) of a spheroid which aspect ratio equals A_r are then given by:

$$\begin{cases} a = A_r^{\frac{2}{3}} \dfrac{D_{eq}}{2} \\ b = A_r^{-\frac{1}{3}} \dfrac{D_{eq}}{2}. \end{cases} \tag{4.1}$$

Each generated array of mono-dispersed spheroids is characterized by the following quantities:

- the spheroids aspect ratio A_r.
- the non-dimensional length $l = L/D_{eq}$ of the cubix box containing the ellipsoids.
- the volume fraction $\alpha_p = N_p \, \dfrac{4}{3}\pi R_{eq}^3 \Big/ L^3$.

Since in PR-DNS particles repel each other (due to solid repulsion), arrays are generated without any overlap, to work in the conditions under which the distance algorithms would be used.

To generate random packings without overlap between the spheroids, a Random Sequential Adsorption (RSA) algorithm is used. Each array is filled particle after particle. The position and orientation of the new particle to insert are randomly picked, and the overlap is tested with every particle already generated with the Jia *et al.*' algorithm [26], that is described in Appendix C. If no overlap is detected with these "old particles", the new particle is retained. Otherwise, another position and orientation are picked, and the procedure is repeated until the new particle can fit in. The algorithm goes on until the desired volume fraction α_p is reached. With this method, volume fractions α_p can be obtained up to 0.25.

Another feature of these packings is periodicity. It suppresses edge effects due to the lack of neighbors for spheroids located near the surfaces of the cubical domain. The periodicity of the arrays is taken into account by adapting the overlap test in the Random Sequential Algorithm described above.

4.1.2. Approach for the computing time statistical study

Distance queries

For a particular array (A_r, α_p, l), all possible distance queries are not solved; the total number of these queries grows approximately as $N_p\,(N_p - 1)/2$, the approximation coming from the periodic character of the array. Instead, the distance is computed for pairs that potentially necessitate a lubrication correction. It means that, for two spheroids with centers $(\mathbf{r}_m)_{m\in[\![1,2]\!]}$ and same semi-axes lengths (a, b, b), the distance between them is computed if the following bounding sphere test augmented by a small distance is verified:

$$\|\mathbf{r}_1 - \mathbf{r}_2\| \leq 2\max(a, b) + \epsilon_{al} R_{eq}. \tag{4.2}$$

where ϵ_{al} is a small parameter setting the distance threshold below which distance-dependent lubrication corrections would be applied during a PR-DNS. The order of magnitude of ϵ_{al} can be found in [7], and in the following, it is fixed to:

$$\epsilon_{al} = 0.2. \tag{4.3}$$

Definition of a converged mean computing time

For each distance method, working with tall enough arrays enables to make statistically true statements about the dependency of its computing cost on:

- the geometrical parameters, meaning the aspect ratio $A_r \in [1/6\,, 6]$ and the volume fraction $\alpha_p \in [0.05, 0.25]$.
- the desired accuracy ϵ_d.

For a particular value of $(A_r, \alpha_p, l, \epsilon_d)$, the total computing time to solve all the distance queries in the corresponding array can be measured. From this total computing time, a *mean computing time* $\langle \Delta t \rangle (A_r, \alpha_p, l, \epsilon_d)$ per proximity query can be defined

$$\langle \Delta t \rangle (A_r, \alpha_p, l, \epsilon_d) = \frac{\text{total computing time for solving all distance queries}}{\text{total number of distance queries}}. \tag{4.4}$$

If the size l of the domain is tall enough (equivalently if enough random binary configurations of ellipsoids are treated), the mean computing time (4.4) becomes independent of l. Therefore, there exists a size threshold l_t such that

$$\forall l \geq l_t, \quad \langle \Delta t \rangle (A_r, \alpha_p, l, \epsilon_d) \equiv \overline{\langle \Delta t \rangle} (A_r, \alpha_p, \epsilon_d). \tag{4.5}$$

$\overline{\langle \Delta t \rangle}$ is called *converged mean computing time*.

For each method $X \in \{\text{MB}, \text{GJK}\}$, a size threshold $l_{t,X}$ will be identified in 4.2. It will enable to define converged mean computing times $\left(\overline{\langle \Delta t \rangle}_X\right)_{X \in \{\text{MB,GJK}\}}$, that will be then compared in 4.3 for different values of the aspect ratio A_r, the volume fraction α_p and the precision parameter ϵ_d.

4.2. Identification of the convergence thresholds

In this subsection, the goal is, for each algorithm, to identify a domain-size threshold l_t, defined by the property (4.5).

At this stage, since no comparison between the algorithms is performed, the mean number of iterations $\langle N_{\text{it}} \rangle$ will be studied instead of directly measuring the mean computing-time $\langle \Delta t \rangle$. Indeed, absolute computing times are submitted to high fluctuations over a long period, engendering difficulties to analyze data when the solving of all distance queries takes several hours.

4.2.1. MB algorithm

In Fig. 21, the mean number of iterations per distance query $\langle N_{\text{it}} \rangle$ of the MB algorithm is plotted against the non dimensional-size of the domain, for a single value of $\epsilon_d = 10^{-5} D_{eq}$. Each subfigure corresponds to a value of the aspect ratio A_r, and each curve corresponds to a value of the volume fraction α_p. It can be seen that an appropriate value of the threshold $l_{t,\text{MB}}$ for defining a converged mean computing time can be taken as:

$$l_{t,\text{MB}} = 30 \tag{4.6}$$

Indeed, Fig. 21 remains qualitatively the same for $\epsilon_d / R_{eq} \in [10^{-8}, 10^{-4}]$ (not shown).

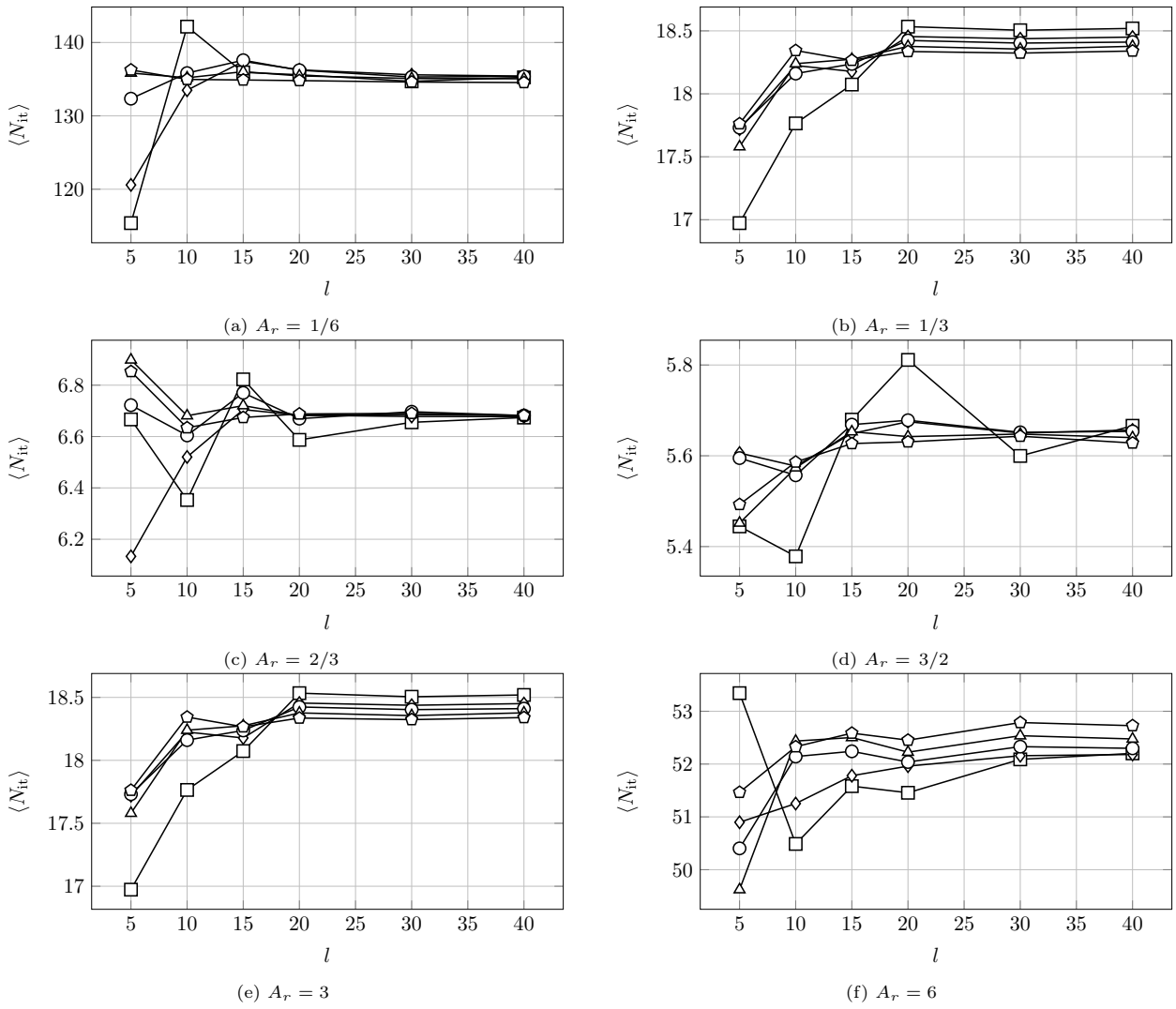


Figure 21: Mean number of iterations per distance query of the MB algorithm against the dimensionless domain size l for different values of the volume fraction α_p and A_r. (□) $\alpha_p = 0.05$, (◇) $\alpha_p = 0.10$ (○) $\alpha_p = 0.15$, (△) $\alpha_p = 0.20$, (○) $\alpha_p = 0.25$. ϵ_d is set to $10^{-5} R_{eq}$. To one subfigure corresponds a single value of A_r.

4.2.2. GJK algorithm

In Fig. 21, the mean number of iterations per distance query $\langle N_{\mathrm{it}} \rangle$ of the GJK algorithm is plotted against the non dimensional-size of the domain, for a single value of $\epsilon_d = 10^{-5} D_{eq}$. Each subfigure corresponds to a value of the aspect ratio A_r, and each curve corresponds to a value of the volume fraction α_p. It can be seen that an appropriate value of the threshold $l_{t,\mathrm{GJK}}$ for defining a converged mean computing time can be taken as:

$$l_{t,\mathrm{GJK}} = 30 \tag{4.7}$$

Indeed, Fig. 22 remains qualitatively the same for $\epsilon_d / R_{eq} \in [10^{-8}, 10^{-4}]$ (not shown).

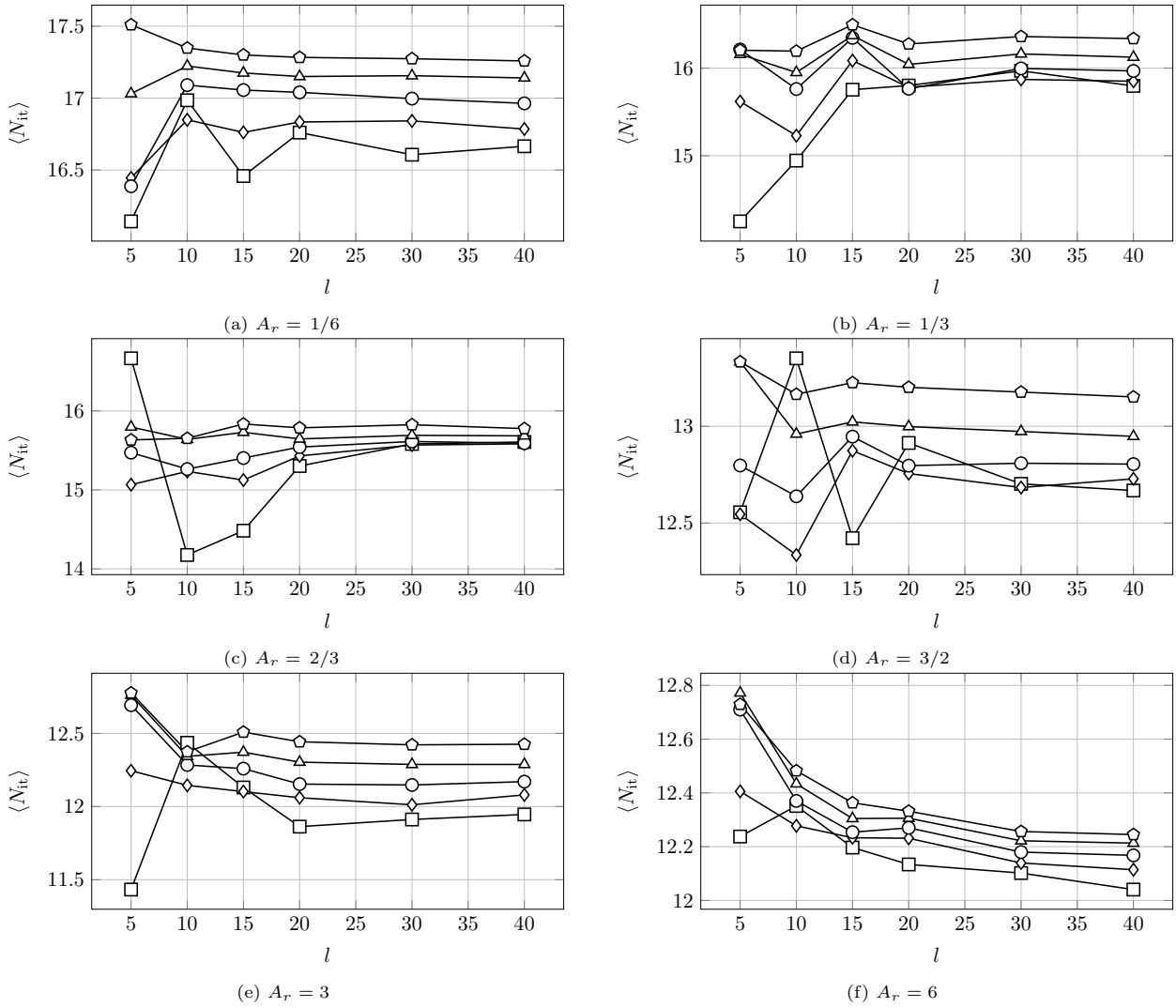


Figure 22: Mean number of iterations per distance query of the GJK algorithm against the dimensionless domain size l for different values of the volume fraction α_p. (□) $\alpha_p = 0.05$, (◇) $\alpha_p = 0.10$ (○) $\alpha_p = 0.15$, (△) $\alpha_p = 0.20$, (⬠) $\alpha_p = 0.25$. ϵ_d is set to $10^{-5} R_{eq}$. To one subfigure corresponds a single value of A_r.

4.3. Computing-costs comparison

A converged mean computing-time per distance query $\overline{\langle \Delta t \rangle}_X$ is now defined for each method $X \in \{\text{MB}, \text{GJK}\}$. In the following subsection, the ratio $\overline{\langle \Delta t \rangle}_{\text{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\text{GJK}}$ will be studied as a function of the remaining parameters, that are recalled to be:

- the aspect ratio A_r.
- the volume fraction α_p.
- the accuracy ϵ_d.

In 4.3.1, the independence from α_p of the ratio $\overline{\langle \Delta t \rangle}_{\text{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\text{GJK}}$ will be outlined, so that the parametric study is actually reduced to varying the aspect ratio A_r and the desired accuracy ϵ_d.

In 4.3.2, the ratio $\overline{\langle \Delta t \rangle}_{\text{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\text{GJK}}$ is studied as a function of the remaining parameters A_r and ϵ_d.

4.3.1. Independence of the ratio $\frac{\overline{\langle \Delta t \rangle}_{\mathbf{MB}}}{\overline{\langle \Delta t \rangle}_{\mathbf{GJK}}}$ from the volume fraction α_p

In Fig. 23, the computing-time ratio $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}}$ is plotted for multiple values of the aspect ratio A_r and the accuracy ϵ_d as a function of the volume fraction α_p. It shows that the computing-time ratio $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}}$ is almost independent from the volume fraction, since the curves $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}} = f(\alpha_p)$ at constant ϵ_d and A_r are nearly horizontal lines.

This enables to simplify the parametric study of the computing-time ratio by eliminating the volume fraction α_p. What remains to be done is to study the dependence of the computing-time ratio on the remaining parameters, namely the aspect ratio A_r and the desired accuracy ϵ_d. The results of this study are shown in the next paragraph.

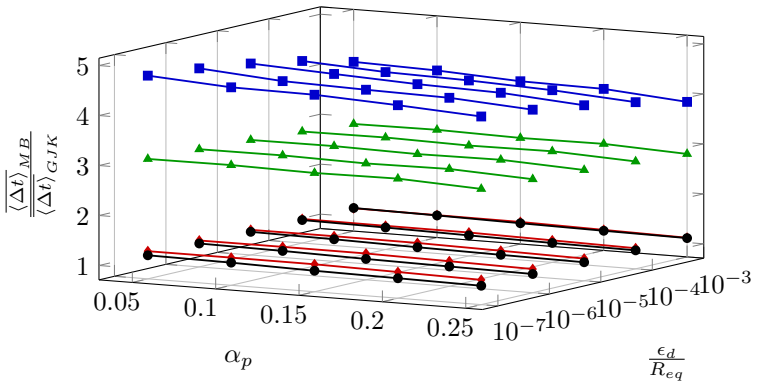


Figure 23: Computing-time ratio $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}}$ as a function of α_p. Each curve corresponds to a value of ϵ_d, and each color corresponds to a value of A_r: (■) $A_r = \frac{1}{6}$, (◆) $A_r = \frac{1}{3}$, (●) $A_r = 3$, and (▲) $A_r = 6$.

4.3.2. Study of the ratio $\frac{\overline{\langle \Delta t \rangle}_{\mathbf{MB}}}{\overline{\langle \Delta t \rangle}_{\mathbf{GJK}}}$ as a function of aspect ratio A_r and desired accuracy ϵ_d

In Fig. 24, the computing-time ratio $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}}$ is plotted against the aspect ratio A_r for different values of the desired accuracy ϵ_d. For the range of investigated values $\epsilon_d / R_{eq} \in \left[10^{-7}; 10^{-3}\right]$, a general observation can be done:

- in a range of aspect ratios $A_r \in [1/3\ , 3]$, the computing-time ratio $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}}$ is shown to be inferior to 1: in this range of aspect ratios, the MB algorithm is thus faster than the GJK algorithm. For the sphere-like aspect ratios $A_r = 2/3$ and $3/2$, the MB algorithm is for instance 2 to 3 times faster than the GJK algorithm.

- for aspect ratios $A_r \leq 1/3$ or $A_r \geq 3$, the computing time ratio $\overline{\langle \Delta t \rangle}_{\mathrm{MB}} \Big/ \overline{\langle \Delta t \rangle}_{\mathrm{GJK}}$ is shown to be superior to 1: in this range of aspect ratios, the GJK algorithm is thus faster than the MB algorithm. The gain in computing-time can be considerable, with the GJK algorithm being more than 4 times faster than the MB algorithm at $A_r = 1/6$ for instance.

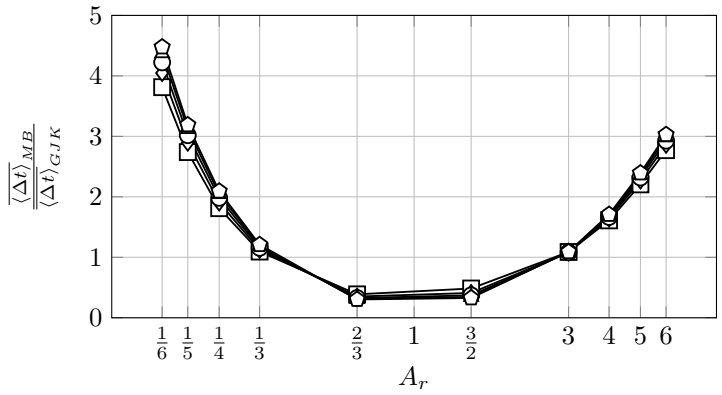


Figure 24: Computing-time ratio $\overline{\langle\Delta t\rangle}_{\mathrm{MB}}\Big/\overline{\langle\Delta t\rangle}_{\mathrm{GJK}}$ as function of aspect ratio A_r. Each curve corresponds to a value of the desired accuracy ϵ_d: (□) $\frac{\epsilon_d}{R_{eq}} = 10^{-3}$, (◇) $\frac{\epsilon_d}{R_{eq}} = 10^{-4}$, (○) $\frac{\epsilon_d}{R_{eq}} = 10^{-5}$, (△) $\frac{\epsilon_d}{R_{eq}} = 10^{-6}$, and (⬠) $\frac{\epsilon_d}{R_{eq}} = 10^{-7}$.

5. Conclusion

Five methods have been extensively studied to retain the most adequate one to perform Particle-Resolved Direct Numerical Simulation featuring ellipsoidal particles immersed in a viscous fluid. This study was guided by the criteria expressed in the introduction.

Three methods appeared to be ruled out by these criteria:

- Newton-Coulomb method: as initially introduced by Abbasov [18, 19, 20], this algorithm is unusable. Indeed, the divergent nature of the electrostatic force for low distances makes the parameter calibration impossible. An effort to improve this method has been reported in this article and is summarized in Algorithm 4. But as outlined in 3.4.3, the modified version of the Newton-Coulomb did not prove as efficient as expected, in terms of robustness and computing cost. The high number of parameters makes the method tedious to calibrate.

- Exterior Function Penalty Method: introduced by Tamasyan *et al.* [21] and described in 2.4, has been ruled out for multiple reasons: the algorithm is not scale-independent, no control of the accuracy has been observed and the algorithm has been observed to be excessively time-consuming in comparison to other methods.

- Jain *et al.*'s algorithm: introduced in [4] and described in 2.5, has been ruled out because of a lack of robustness exhibited in 3.4.5. An attempt to scale the parameters of the method while preserving a competitive computing cost revealed unsuccessful because the monotonic decreasing of the distance with the iteration number k is not assured.

The two remaining methods, namely the Moving Balls algorithm 2.1 and the GJK algorithm 2.2 have been studied on randomly-generated, mono-dispersed arrays of spheroids. The robust character of these two methods has thus been firmly established. An explicit stopping criterion to control the committed error with the Moving Balls method has been derived since it was not available in the original paper by Lin *et al.* [13]. After sizing the arrays to make statistically significant statements about the computing-time of the methods, the mean computing-time per distance query of the Moving Balls algorithm and the GJK algorithm have been compared for multiple values of the aspect ratio, volume fraction, and desired accuracy. It revealed that the GJK algorithm is faster for flattened ($A_r \leq 1/3$) or elongated ($A_r \geq 3$) shapes. For intermediate aspect ratios $1/3 \leq A_r \leq 3$, the Moving Balls algorithm is the fastest method. A possible prospect for future work would be to pursue the statistical study in random arrays of ellipsoids with various sizes and aspect ratios.

Declaration of competing interest

The authors have no competing interest to declare rgarding the publication of this article.

Aknowledgements

This work was granted access to the HPC resources of IDRIS, CINES and CCRT under the allocation A0072b06115 proposed by GENCI (Grand Equipement National de Calcul Intensif).

Appendix A. Mathematical description of ellipsoids

From a mathematical point of view, an ellipsoid E_m of $\mathbb{R}^3$ can be represented in a Cartesian frame by a quadratic form f_m:

$$f_m(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}_m \mathbf{x} + \mathbf{b}_m^\top \mathbf{x} + \alpha_m \tag{A.1}$$

where $\mathbf{A}_m$ is 3×3 symmetric definite positive matrix, $\mathbf{b}_m$ a vector of $\mathbb{R}^3$, α_m a constant and $\mathbf{x} \in \mathbb{R}^3$.

The ellipsoid E_m is then defined as

$$E_m = \left\{ \mathbf{x} \in \mathbb{R}^3 | f_m(\mathbf{x}) \leq 0 \right\}. \tag{A.2}$$

In the frame defined by the three semi-axis of E_m, with the origin that coincides with the centroid of E_m, the ellipsoid E_m can be represented as:

$$\frac{1}{2}\tilde{\mathbf{x}}^\top \mathbf{\Sigma}_m \tilde{\mathbf{x}} - \frac{1}{2} \leq 0 \tag{A.3}$$

where $\mathbf{\Sigma}_m$ is the 3×3 matrix defined as:

$$\mathbf{\Sigma}_m = \begin{pmatrix} \left(\frac{1}{a_m}\right)^2 & 0 & 0 \\ 0 & \left(\frac{1}{b_m}\right)^2 & 0 \\ 0 & 0 & \left(\frac{1}{c_m}\right)^2 \end{pmatrix} \tag{A.4}$$

a_m, b_m, and c_m being the length of the three semi-axes of E_m.

To explicit $\mathbf{A}_m$, $\mathbf{b}_m$ and α_m in Eq. (A.1), the position of the centre $\mathbf{r}$ and the orientation of E_m must be taken into account. The use of *Euler angles* is avoided, thanks to the *Unitary Quaternions Group*. The reader wishing to grasp more about Quaternions can refer to Wach's review [12] as a starting point, but all the strict necessary information is introduced below.

If the orientation of the ellispoid E_m in $\mathbb{R}^3$ is represented by a unitary axis of rotation $\hat{\mathbf{n}}_m$, and a angle of rotation θ_m, a Unitary Quaternion $\boldsymbol{q}_m$ can be defined from these quantities as:

$$\boldsymbol{q}_m = (q_{m,0}, q_{m,1}, q_{m,2}, q_{m,3}) = \left(\cos\left(\frac{\theta_m}{2}\right), \sin\left(\frac{\theta_m}{2}\right)\hat{\mathbf{n}}_m\right). \tag{A.5}$$

The Quaternion $\boldsymbol{q}_m$ completely describes the orientation of E_m in $\mathbb{R}^3$. Indeed, the rotation matrix $\mathbf{U}_m$ associated with the orientation of E_m can be reconstructed from it:

$$\mathbf{U}(\boldsymbol{q}_m) = \begin{pmatrix} 1 - 2\left((q_{m,2})^2 + (q_{m,3})^2\right) & 2\left(q_{m,1}q_{m,2} - q_{m,0}q_{m,3}\right) & 2\left(q_{m,0}q_{m,2} + q_{m,1}q_{m,3}\right) \\ 2\left(q_{m,1}q_{m,2} + q_{m,0}q_{m,3}\right) & 1 - 2\left((q_{m,1})^2 + (q_{m,3})^2\right) & 2\left(q_{m,2}q_{m,3} - q_{m,0}q_{m,1}\right) \\ 2\left(q_{m,1}q_{m,3} - q_{m,0}q_{m,2}\right) & 2\left(q_{m,0}q_{m,1} + q_{m,2}q_{m,3}\right) & 1 - 2\left((q_{m,1})^2 + (q_{m,2})^2\right) \end{pmatrix}. \tag{A.6}$$

The components $\tilde{\mathbf{x}}$ of the position vector in the body-fixed frame associated with E_m can be related to the components $\mathbf{x}$ of the position vector in the general frame with the formula:

$$\tilde{\mathbf{x}} = \mathbf{U}_m^\top (\mathbf{x} - \mathbf{r}_m) \tag{A.7}$$

with $\mathbf{r}_m$ being the centre of the ellipsoid in the general frame.

Thus, Eq. (A.3) can be re-written

$$\frac{1}{2}\mathbf{x}^\top \mathbf{U}_m \mathbf{\Sigma}_m \mathbf{U}_m^\top \mathbf{x} - (\mathbf{U}_m \mathbf{\Sigma}_m \mathbf{U}_m^\top \mathbf{r}_m)^\top \mathbf{x} + \frac{1}{2}\mathbf{r}_m^\top (\mathbf{U}_m \mathbf{\Sigma}_m \mathbf{U}_m^\top)\mathbf{r}_m - \frac{1}{2} \leq 0. \tag{A.8}$$

By identification with Eq. (A.1), it yields:

$$\mathbf{A}_m = \mathbf{U}_m \mathbf{\Sigma}_m \mathbf{U}_m^\top \tag{A.9}$$

$$\mathbf{b}_m = -\mathbf{A}_m \mathbf{r}_m \tag{A.10}$$

$$\alpha_m = \frac{1}{2}\mathbf{r}_m^\top \mathbf{A}_m \mathbf{r}_m - \frac{1}{2} \tag{A.11}$$

where we recall that $\mathbf{\Sigma}_m$ is the matrix defined in Eq. (A.4), and $\mathbf{r}_m$ is the center of the ellipsoid in the general frame.

Appendix B. Curvature of an ellipsoid

For an ellipsoid E_m, centered on the origin, and with its semi-axes (a_m, b_m, c_m) aligned with the Cartesian axes, the Gaussian curvature K_m and the mean curvature H_m of the surface ∂E_m are [27]:

$$\begin{cases} K_m = \dfrac{1}{R_{\min,m} R_{\max,m}} = \left(\dfrac{1}{a_m b_m c_m \left(\frac{x^2}{a_m^4} + \frac{y^2}{(b_m)^4} + \frac{z^2}{(c_m)^4} \right)} \right)^2 \\ H_m = \dfrac{1}{2}\left(\dfrac{1}{R_{\min,m}} + \dfrac{1}{R_{\max,m}} \right) = \dfrac{(a_m)^2 + (b_m)^2 + (c_m)^2 - x^2 - y^2 - z^2}{2(a_m b_m c_m)^2 \left(\frac{x^2}{(a_m)^4} + \frac{y^2}{(b_m)^4} + \frac{z^2}{(c_m)^4} \right)^{\frac{3}{2}}} \end{cases} \tag{B.1}$$

with $R_{\min,m}$ and $R_{\max,m}$ respectively being the smallest and tallest curvature radius of the surface ∂E_m at the point (x, y, z).

From Eq. (B.1), it can be established that both $R_{\min,m}$ and $R_{\max,m}$ satisfy the second-order equation in X

$$K_m X^2 - 2H_m X + 1 = 0 \tag{B.2}$$

which solutions are

$$\begin{cases} R_{\min,m} = \dfrac{H_m - \sqrt{(H_m)^2 - K_m}}{K_m} \\ R_{\max,m} = \dfrac{H_m + \sqrt{(H_m)^2 - K_m}}{K_m}. \end{cases} \tag{B.3}$$

The local Gaussian radius of curvature $R_{G,m}$ is defined as:

$$R_{G,m} = \frac{1}{\sqrt{K_m}} = a_m b_m c_m \left(\frac{x^2}{(a_m)^4} + \frac{y^2}{(b_m)^4} + \frac{z^2}{(c_m)^4} \right). \tag{B.4}$$

Appendix C. Overlap detection algorithm

Appendix C.1. The algorithm of Jia *et al.* [26]

In this subsection, an exact method to detect if two ellipsoids collide is presented. This method is based on the analytical criterion of Wang *et al.* [28], demonstrated in 2001. On his original form, this analytical criterion is not adequate for a fast implementation, since it involves locating in the complex plane the roots of a fourth-order polynomial. Fortunately, this criterion has been reformulated by Jia *et al.* [26] in 2011 so that only a few algebraic expressions have to be computed to determine if the two ellipsoids overlap or not.

Appendix C.1.1. The original criterion of Wang [28]

In [28], the ellipsoids $(E_m)_{m\in[\![1,2]\!]}$ are described in *homogeneous coordinates*, meaning the equation describing each ellipsoid $m \in [\![1,2]\!]$ reformulated as:

$$X^T \mathcal{A}_m X \leq 0 \tag{C.1}$$

where $X = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}$, and $(\mathcal{A}_m)_{m\in[\![1,2]\!]}$ are 4×4 matrices. How to compute these matrices is explained in Appendix C.2.

With the use of $(\mathcal{A}_m)_{m\in[\![1,2]\!]}$, one can define the characteristic polynomial P associated with the two ellipsoids E_1 and E_2:

$$P(\lambda) = \det\Big(\lambda I_4 + (\mathcal{A}_1)^{-1}\,\mathcal{A}_2\Big). \tag{C.2}$$

Remarks:

- P is a fourth-order monic polynomial. In [28] or [26], the characteristic polynomial is designated by $\det(\lambda\mathcal{A}_1 + \mathcal{A}_2)$, which is not necessarily a monic polynomial. Eq. (C.2) provides directly a polynomial with the leading coefficient equal to 1.
- In order to reduce the cost of computing the determinant in Eq. (C.2), the matrices $(\mathcal{A}_m)_{m\in[\![1,2]\!]}$ can be computed in the reference frame associated with the ellipsoid E_1. In that way, the matrix $\mathcal{A}_1$ is diagonal, and is easily inverted.

The following theorem, demonstrated in [28], constitutes an exact criterion for the overlapping of E_1 and E_2.

Theorem 3 (Wang *et al.*' criterion for the overlap of two ellipsoids, Wang *et al.* 2001). *Let P be the characteristic polynomial associated with E_1 and E_2.*

1. *The characteristic polynomial P has at least two real negative roots.*
2. *The two ellipsoids E_1 and E_2 are separated if and only if P has two distinct real positive roots.*
3. *The two ellipsoids E_1 and E_2 touch each other externally if and only if P has a real positive double root.*

Computing the roots of P to determine if E_1 and E_2 is an expensive operation. Moreover, the precise value of these roots is not necessary to verify the overlap criterion expressed in Theorem 3. That is why this criterion has to be adapted in order to be used in a fast algorithm. This task has been achieved in [26].

Appendix C.1.2. Reformulation of Wang *et al.*' criterion

In the following, P is supposed to have been computed:

$$P(\lambda) = \lambda^4 + a\lambda^3 + b\lambda^2 + c\lambda + d. \tag{C.3}$$

A first definition is necessary.

Definition 1 (Number of sign variations in a finite sequence). *Let be s a finite sequence of real numbers.* $\mathrm{Var}(s)$ *is defined as the number of sign changes between two consecutive non-zero elements of s.*

For instance:

- $\mathrm{Var}(-1,1,2) = 1$
- $\mathrm{Var}(1,2,3) = 0$

- $\mathrm{Var}(1, 0, -1) = 1$

The reformulation of the Wang *et al.*' criterion also involves five quantities $\mathbf{sr}_{22}$,$\mathbf{sr}_{20}$, $\mathbf{sr}_{11}$, $\mathbf{sr}_{10}$ and $\mathbf{sr}_0$, computed from the coefficients (a, b, c, d) of the characteristic polynomial P following the procedure:

$$\begin{cases} \bar{b} = -a/4 \\ \bar{c} = b/6 \\ \bar{d} = -c/4 \\ \bar{e} = d \end{cases}, \quad \begin{cases} \Delta_2 = \bar{b}^2 - \bar{c} \\ \Delta_3 = \bar{c}^2 - \bar{b}\bar{d} \\ W_1 = \bar{d} - \bar{d}\bar{c} \\ W_2 = \bar{b}\bar{e} - \bar{c}\bar{d} \\ W_3 = \bar{e} - \bar{b}\bar{d} \end{cases}, \quad \begin{cases} T = -9W_1^2 + 27\Delta_2\Delta_3 - 3W_3\Delta_2 \\ A = W_3 + 3\Delta_3 \\ B = -\bar{d}W_1 - \bar{e}\Delta_2 - \bar{c}\Delta_3 \\ T_2 = AW_1 - 3\bar{b}B \\ \Delta_1 = A^3 - 27B^2 \end{cases}, \quad \begin{cases} \mathbf{sr}_{22} = \Delta_2 \\ \mathbf{sr}_{20} = -W_3 \\ \mathbf{sr}_{11} = T \\ \mathbf{sr}_{10} = T_2 \\ \mathbf{sr}_0 = \Delta_1. \end{cases} \tag{C.4}$$

Finally, the practical criterion, demonstrated in [26], is given by the following theorem.

Theorem 4 (Algebraic criterion for the overlap of two ellipsoids, Jia *et al.* 2011). *Let* $P(\lambda) = \lambda^4 + a\lambda^3 + b\lambda^2 + c\lambda + d$ *be the characteristic polynomial associated with* E_1 *and* E_2.

1. *The two ellipsoids* E_1 *and* E_2 *are separate if and only if* $\mathrm{Var}(1, a, b, c, d) = 2$ *and*
   - *a* $\mathbf{sr}_{22} > 0$, $\mathbf{sr}_{11} > 0$, $\mathbf{sr}_0 > 0$ *or*
   - *b* $\mathbf{sr}_{22} > 0$, $\mathbf{sr}_{11} > 0$, $\mathbf{sr}_{10} > 0$, $\mathbf{sr}_0 = 0$
2. *The two ellipsoids* E_1 *and* E_2 *touch each other externally if and only if*
   - *a* $\mathbf{sr}_{22} > 0$, $\mathbf{sr}_{11} > 0$, $\mathbf{sr}_{10} < 0$, $\mathbf{sr}_0 = 0$ *or*
   - *b* $\mathbf{sr}_{22} > 0$, $\mathbf{sr}_{20} < 0$, $\mathbf{sr}_{11} = 0$, $\mathbf{sr}_0 = 0$

In the other cases, the two ellipsoids E_1 *and* E_2 *overlap.*

Appendix C.1.3. Numerical schedule

The numerical process is summed up in Algorithm 7.

Algorithm 7 Overlap detection algorithm

```
Compute matrices (A_m)_{m∈⟦1,2⟧} in the reference frame associated with ellipsoid E_1, with Eq. (C.9)
Compute the coefficients (a, b, c, d) of the characteristic polynomial P defined in Eq. (C.2).
Set bool = .TRUE.
Compute Var(a, b, c, d)
if Var(a, b, c, d) = 2 then
    Compute sr_22,sr_20, sr_11, sr_10 and sr_0 with Eq. (C.4)
    if (sr_22 > 0, sr_11 > 0, sr_0 > 0) or ( sr_22 > 0, sr_11 > 0, sr_10 > 0, sr_0 = 0 ) then
        Set bool = .FALSE.
    end if
end if
return bool
```

Appendix C.2. Homogeneous coordinates

The equation $\frac{1}{2}\,{}^t\mathbf{x}\mathbf{A}_m\mathbf{x} + {}^t\mathbf{b}_m\mathbf{x} + \alpha_m \leq 0$ can be reformulated as:

$$X^\top \mathcal{A}_m X \leq 0 \tag{C.5}$$

where $X = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}$ and $\mathcal{A}_m$ is a 4×4 matrix.

Indeed, in its own reference frame, with the components of the position vector $\tilde{X} = \begin{pmatrix} \tilde{\mathbf{x}} \\ 1 \end{pmatrix}$, the equation of E_m can be written:

$$\tilde{X}^\top \underbrace{\begin{pmatrix} \left(\frac{1}{a_m}\right)^2 & 0 & 0 & 0 \\ 0 & \left(\frac{1}{b_m}\right)^2 & 0 & 0 \\ 0 & 0 & \left(\frac{1}{c_m}\right)^2 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}}_{\mathcal{S}_m} \tilde{X} \leq 0. \tag{C.6}$$

The relation between X and $\tilde{X}$ is given by the affine transformation (translation + rotation) characterizing the position and orientation of the ellipsoid E_m. In homogeneous coordinates, an affine transformation is represented by a 4×4 matrix $\mathcal{M}_m$ such that:

$$X = \mathcal{M}_m \tilde{X}. \tag{C.7}$$

The matrix $\mathcal{M}_m$ is related to the centre $\mathbf{r}_m$ and 3×3 rotation matrix $\mathbf{U}_m$ of E_m by [29]:

$$\mathcal{M}_m = \begin{pmatrix} \mathbf{U}_m & \mathbf{r}_m \\ \mathbf{0}^\top & 1 \end{pmatrix} \Longleftrightarrow \mathcal{M}_m^{-1} = \begin{pmatrix} \mathbf{U}_m^\top & -\mathbf{U}_m^\top \mathbf{r}_m \\ \mathbf{0}^\top & 1 \end{pmatrix}. \tag{C.8}$$

Thus the matrix $\mathcal{A}_m$ has the expression:

$$\mathcal{A}_m = \left(\mathcal{M}_m^{-1}\right)^\top \mathcal{S}_m \mathcal{M}_m^{-1}. \tag{C.9}$$

Appendix D. Computation of the intersection point(s) between an ellipsoid and a segment

Let E_m be an ellipsoid represented by f_m and $(\mathbf{A}_m, \mathbf{b}_m, \alpha_m)$ (see Eq. A.1 and Eq. A.2). We wish to compute the intersection point of ∂E_m with the segment $[\mathbf{c}_1, \mathbf{c}_2]$. To proceed, the segment $[\mathbf{c}_1, \mathbf{c}_2]$ is parameterized by $t \in [0, 1]$:

$$\begin{cases} x(t) = x_1 + t(x_2 - x_1) \\ y(t) = y_1 + t(y_2 - y_1) \\ z(t) = z_1 + t(z_2 - z_1). \end{cases} \tag{D.1}$$

By injecting $x(t)$, $y(t)$ and $z(t)$ in the equation of ∂E $f = 0$, a second order polynomial equation in t is obtained:

$$At^2 + Bt + C = 0, \quad t \in [0, 1] \tag{D.2}$$

where

$$2A = (\mathbf{A}_m)_{11}(x_2 - x_1)^2 + (\mathbf{A}_m)_{22}(y_2 - y_1)^2 + (\mathbf{A}_m)_{33}(z_2 - z_1)^2 \tag{D.3}$$
$$+ 2(\mathbf{A}_m)_{12}(x_2 - x_1)(y_2 - y_1)$$
$$+ 2(\mathbf{A}_m)_{13}(x_2 - x_1)(z_2 - z_1)$$
$$+ 2(\mathbf{A}_m)_{23}(y_2 - y_1)(z_2 - z_1)$$
$$B = (\mathbf{A}_m)_{11} x_1(x_2 - x_1) + (\mathbf{A}_m)_{22} y_1(y_2 - y_1) + (\mathbf{A}_m)_{33} z_1(z_2 - z_1) \tag{D.4}$$
$$+ (\mathbf{A}_m)_{12}[x_1(y_2 - y_1) - y_1(x_2 - x_1)]$$
$$+ (\mathbf{A}_m)_{13}[x_1(z_2 - z_1) - z_1(x_2 - x_1)]$$
$$+ (\mathbf{A}_m)_{23}[y_1(z_2 - z_1) - z_1(y_2 - y_1)]$$
$$+ (\mathbf{b}_m)_1(x_2 - x_1) + (\mathbf{b}_m)_2(y_2 - y_1) + (\mathbf{b}_m)_3(z_2 - z_1)$$
$$C = \frac{1}{2}\left[(\mathbf{A}_m)_{11}(x_1)^2 + (\mathbf{A}_m)_{22}(y_1)^2 + (\mathbf{A}_m)_{33}(z_1)^2\right] \tag{D.5}$$
$$+ (\mathbf{A}_m)_{12} x_1 y_1 + (\mathbf{A}_m)_{13} x_1 z_1 + (\mathbf{A}_m)_{23} y_1 z_1$$
$$+ (\mathbf{b}_m)_1 x_1 + (\mathbf{b}_m)_2 y_1 + (\mathbf{b}_m)_3 z_1$$
$$+ \alpha_m. \tag{D.6}$$

Appendix E. Reformulating the stopping criterion on angles

The angle $\theta(\mathbf{a}, \mathbf{b})$ between two vectors $\mathbf{a}$ and $\mathbf{b}$ is calculated as follows:

$$\theta(\mathbf{a}, \mathbf{b}) = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||||\mathbf{b}||}\right), \quad 0 \leq \theta \leq \pi. \tag{E.1}$$

It is necessary to reformulate the condition expressed in Eq. (2.4) to avoid the computation of an arccos , square roots and a division (see Eq. (E.1)) at each iteration. Here is how to proceed:

$$\begin{aligned}
\theta(\mathbf{a}, \mathbf{b}) \leq \epsilon_\theta &\iff \cos\theta(\mathbf{a}, \mathbf{b}) \geq \cos\epsilon_\theta \\
&\iff \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||||\mathbf{b}||} \geq 1 - \frac{\epsilon_\theta^2}{2} \\
&\iff (\mathbf{a} \cdot \mathbf{b})^2 \geq \left(1 - \frac{\epsilon_\theta^2}{2}\right)^2 ||\mathbf{a}||^2 ||\mathbf{b}||^2.
\end{aligned} \tag{E.2}$$

- The first equivalency is a consequence of the convention introduced in Eq. (E.1): $\theta \in [0, \pi]$. The decreasing of cos on $[0, \pi]$ is used.
- The second equivalency is true if ϵ_θ is small.

Appendix F. How to scale ϵ_θ

In Appendix F.1, an inequality is derived that relates the stopping parameter ϵ_θ (2.4) with the committed error on the numerical solution. Then, this inequality is used in Appendix F.2 to calculate a value for ϵ_θ that enforces the distance error to be inferior to ϵ_d. Two different implementations are proposed. Lastly, the consistency of these implementations is tested on a large number of distance queries in Appendix F.3.

Appendix F.1. Relation between the stopping criterion on angles ϵ_θ and the error made by the algorithm

An ideal configuration is presented in Fig. F.25. The points $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ and $d^* = ||\mathbf{x}_2^* - \mathbf{x}_1^*||$ are the exact solutions of problem (1.3). The points $\mathbf{x}_1$, $\mathbf{x}_2$ and $d = ||\mathbf{x}_2 - \mathbf{x}_1||$ are the approximated solutions of the

Problem (1.3), provided by the algorithm. The angle between the normal of the surface ∂E_m and the approximated distance vector is noted θ_m. Both the angles $(\theta_m)_{m\in[\![1,2]\!]}$ are inferior to the stopping parameter ϵ_θ (meaning they satisfy the criterion expressed in Eq. (2.4)).

The goal of this section is to relate the errors $|d-d^*|$ and $(\|\mathbf{x}_m-\mathbf{x}_m^*\|)_{m\in[\![1,2]\!]}$ to the value of the stopping parameter ϵ_θ.

It is assumed (with lack of generality) that the approximated distance vector $\mathbf{x}_1-\mathbf{x}_2$ is colinear to the exact distance vector $\mathbf{x}_1^*-\mathbf{x}_2^*$: it is convenient for the calculation to come, but it is not necessarily the case in practice when the algorithm ends. Nonetheless, this simplification is not an issue as long as only a scaling law is looked for.

Thanks to the previous simplifications, the four points $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$ are contained in the a same plane Π. Then, in the vicinity of $\mathbf{x}_m^*$, the curve defined as the intersection $\partial E_m\cap\Pi$ is approximated as a curve with a constant curvature radius $R_m(\mathbf{x}_m^*,\Pi)$ for $m\in[\![1,2]\!]$ (see Fig. F.25).

By geometrical considerations, the quantities $(\|\mathbf{x}_m-\mathbf{x}_m^*\|)_{m\in[\![1,2]\!]}$ and $(S_m)_{m\in[\![1,2]\!]}$ (see Fig. F.25) can be expressed:

$$\begin{cases} ||\mathbf{x}_m-\mathbf{x}_m^*|| = \sqrt{2\left(1-\cos\theta_m\right)}R_m \simeq \theta_m R_m \\ S_m = \sin\dfrac{\theta_m}{2}\sqrt{2\left(1-\cos\theta_m\right)}R_m \simeq (\theta_m)^2\dfrac{R_m}{2}. \end{cases} \tag{F.1}$$

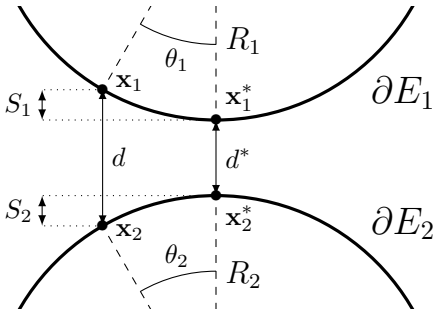


Figure F.25: A possible configuration when an algorithm stops

If ϵ_θ is the stopping parameter introduced in Eq. (2.4), the following scaling laws are true:

$$\begin{cases} ||\mathbf{x}_m-\mathbf{x}_m^*|| \simeq \epsilon_\theta R_m \\ d-d^* \simeq (\epsilon_\theta)^2\dfrac{R_1+R_2}{2}. \end{cases} \tag{F.2}$$

If the maximum curvature radius of the surface ∂E_m at the point $\mathbf{x}_m^*$ is noted $R_{\max,m}(\mathbf{x}_m^*)$, Eq. (F.2) implies the inequalities:

$$|d-d^*| \leq (\epsilon_\theta)^2\,\frac{R_{\max,1}(\mathbf{x}_1^*)+R_{\max,2}(\mathbf{x}_2^*)}{2} \tag{F.3}$$

$$\leq (\epsilon_\theta)^2\,\frac{1}{2}\left(\max_{\partial E_1}(R_{\max,1})+\max_{\partial E_2}(R_{\max,2})\right). \tag{F.4}$$

The last inequalities (F.3) (local) and (F.4) (global) can be used to scale the stopping parameter ϵ_θ.

Appendix F.2. Possible versions of implementation

Both the Moving Ball Algorithm 2.1 and the Newton-Coulomb Method 2.3 provide a sequence of points $\left(\mathbf{x}_1^k,\mathbf{x}_2^k\right)_k$, which stops at iteration n when the criterion (2.4) involving the parameter ϵ_θ is met.

Since the maximal error on the distance is limited by the parameter ϵ_d (see Eq. (2.8)), the inequalities (F.3) and (F.4) provide two possible versions for the algorithms in question.

Appendix F.2.1. Version 1: constant-value ϵ_θ

If $\epsilon_d = (\epsilon_\theta)^2 \frac{1}{2}\left(\max_{\partial E_1}(R_{\max,1}) + \max_{\partial E_2}(R_{\max,2})\right)$, then the inequality $\left|\|\mathbf{x}_1^n - \mathbf{x}_2^n\| - d^*\right| \leq \epsilon_d$ will be met when the algorithm stops at iteration n, according to the global inequality (F.4).

Therefore, the value of ϵ_θ can be set to the global constant:

$$\epsilon_\theta = \sqrt{\frac{2\epsilon_d}{\max_{\partial E_1}(R_{\max,1}) + \max_{\partial E_2}(R_{\max,2})}}. \tag{F.5}$$

Note that the global maxima $\left(\max_{\partial E_m}(R_{\max,m})\right)_{m\in[\![1,2]\!]}$ are easily computed in the case of axisymmetric ellipsoids (see Appendix Appendix B).

Appendix F.2.2. Version 2: evolving ϵ_θ

A less constraining version can be established from the local inequality (F.3), by locally compute the value of ϵ_θ at each iteration k from the local curvature radii of the surfaces $(\partial E_m)_{m\in[\![1,2]\!]}$ at the points $\left(\mathbf{x}_m^k\right)_{m\in[\![1,2]\!]}$:

$$\epsilon_\theta^k = \sqrt{\frac{2\epsilon_d}{R_{\max,1}\left(\mathbf{x}_1^k\right) + R_{\max,2}\left(\mathbf{x}_2^k\right)}}. \tag{F.6}$$

With this version, less iterations will be necessary to satisfy the desired distance accuracy ϵ_d, but the local curvature radii $\left(R_{\max,m}\left(\mathbf{x}_m^k\right)\right)_{m\in[\![1,2]\!]}$ must be computed at each iteration k.

Appendix F.3. Consistency check

It is aimed in this part to verify that the previously described convergence criterion enforces the distance error to verify the inequality (2.8). This verification can be conducted with any method that employs this convergence criterion; the Moving Ball algorithm is one of them, and will be employed in this part. Among the two possible versions of the convergence criterion, the adaptive version Appendix F.2.2 is tested because it is less restrictive than its constant version Appendix F.2.1.

The randomly generated arrays of spheroids that are described in 4.1.1 are used to test the convergence criterion on a large number of distance queries. Of course, for all these distance queries no analytical solution is available. The reference distance is thus taken as the distance d^{GJK} obtained with the GJK algorithm. Indeed the GJK algorithm does not rely on the studied convergence criterion to impose the distance error to be inferior to ϵ_d, so that the comparison is legitimate. 2.2 set with a low value of the ϵ_d parameter:

$$\epsilon_d^{GJK} = 10^{-8} R_{eq} \tag{F.7}$$

while the Moving Balls algorithm is set with a parameter ϵ_d such that

$$\epsilon_d \gg \epsilon_d^{GJK}. \tag{F.8}$$

For each distance query, the MB algorithm provides a distance d^{MB} that is compared to the solution d^{GJK} of the GJK algorithm. It enables to estimate the error committed by the Moving Balls algorithms because:

$$\begin{aligned}\left|d^{MB} - d^{GJK}\right| &= \left|\underbrace{d^{MB} - d^*}_{\mathcal{O}(\epsilon_d)} - \underbrace{\left(d^{GJK} - d^*\right)}_{\mathcal{O}\left(\epsilon_d^{GJK}\right)}\right| \\ &\simeq \left|d^{MB} - d^*\right|. \end{aligned} \tag{F.9}$$

In Fig. F.26, the maximum deviation of $d^{MB}(\epsilon_d)$ to the reference distance d^{GJK} is plotted against ϵ_d. This maximum is computed on an important number of distance queries (more than two millions). It can be seen that the result coincides with the line $y = \epsilon_d$, demonstrating that the convergence criterion exposed in Appendix F and implemented in the Moving Ball algorithm is suitable to control the distance accuracy.

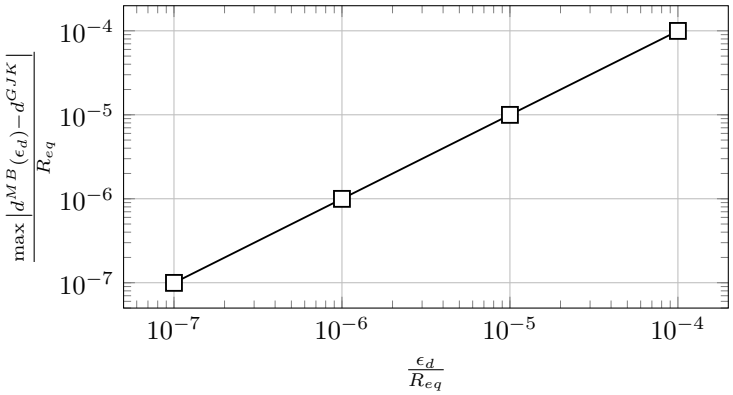


Figure F.26: Maximum error committed by the MB algorithm set with ϵ_d, by comparison with the reference solution d^{GJK}. The maximum error is computed for all distance queries occurring in the arrays described in 4.1.1 and corresponding to $l = 40$ and $\alpha = 0.25$ (for all values of aspect ratio A_r); it involves 2.081.738 distance queries.

Appendix G. Convex analysis notions

Definition 2 (Minkowski sum). *Let E_1 and E_2 be two convex set. The Minkowski sum $E_1 - E_2$ is defined as*

$$E_1 - E_2 = \{\mathbf{p} - \mathbf{q} \mid \mathbf{p} \in E_1, \mathbf{q} \in E_2\}.$$

Theorem 5 (Hilbert Projection Theorem for $\mathbb{R}^3$). *Let C be a closed convex set of $\mathbb{R}^3$, and $\mathbf{x_0}$ a point in $\mathbb{R}^3 \setminus C$.*
There exists a unique $\mathbf{z}_C \in C$ such that:

$$\|\mathbf{z}_C - \mathbf{x_0}\| = \min_{\mathbf{z} \in C} \|\mathbf{z} - \mathbf{x_0}\|.$$

Definition 3 (Support mapping). *Let C be a convex set of $\mathbb{R}^3$. The support mapping of C is an application $\mathbf{s}_C : \mathbb{R}^3 \to C$, such that $\forall \mathbf{d} \in \mathbb{R}^3$, $\mathbf{s}_C(\mathbf{d})$ is the unique vector verifying:*

$$\mathbf{s}_C(\mathbf{d}) \cdot \mathbf{d} = \max\{\mathbf{x} \cdot \mathbf{d} \mid \mathbf{x} \in C\}.$$

Geometrically, $\mathbf{s}_C(\mathbf{d})$ is the most extreme point of C in the direction defined by $\mathbf{d}$.

Definition 4 (Affine and convex hull). *Let $X = \{\mathbf{x}_0, ..., \mathbf{x}_n\}$ be a finite subset of $\mathbb{R}^3$. One can define the affine hull and the convex hull of X, respectively noted $\mathrm{aff}(X)$ and $\mathrm{conv}(X)$, as:*

$$\mathrm{aff}(X) = \left\{ \sum_{i=0}^{n} \lambda_i \mathbf{x}_i \mid \sum_{i=0}^{n} \lambda_i = 1 \right\} \tag{G.1}$$

$$\mathrm{conv}(X) = \left\{ \sum_{i=0}^{n} \lambda_i \mathbf{x}_i \mid \sum_{i=0}^{n} \lambda_i = 1, \lambda_i \geq 0 \right\}. \tag{G.2}$$

Definition 5 (Notion of affinely independent set of points). *A finite set of points $X \subset \mathbb{R}^3$ is said to be affinely independent if $\forall \mathbf{x} \in X$, $\mathbf{x}$ is not an affine combination of elements of $X \setminus \mathbf{x}$.*

In $\mathbb{R}^3$, an affinely independent set of points X has at most 4 elements. Following the value of $\mathrm{Card}(X)$, the affine hull of X, $\mathrm{aff}(X)$, has a different nature:

- if $\text{Card}(X) = 2$, $\text{aff}(X)$ is the line passing by the two elements of X.
- if $\text{Card}(X) = 3$, $\text{aff}(X)$ is the plane defined by the 3 elements of X.
- if $\text{Card}(X) = 4$, $\text{aff}(X) = \mathbb{R}^3$.

Definition 6 (Simplex). *A simplex is the convex hull of an affinely independent set of points*

If S is the simplex shuch that $S = \text{conv}(X)$, X being an affinely independent set of points, then the nature of S depends on $\text{Card}(X)$:

- if $\text{Card}(X) = 1$, $S = \text{conv}(X)$ is the only element of X.
- if $\text{Card}(X) = 2$, $S = \text{conv}(X)$ is the segment which the extremities are the two elements of X.
- if $\text{Card}(X) = 3$, $S = \text{conv}(X)$ is the triangle which the vertices are the 3 elements of X.
- if $\text{Card}(X) = 4$, $S = \text{conv}(X)$ is the tetrahedron the which vertices are the four elements of X.

References

[1] G. van den Bergen, Proximity queries and pentration depth computation on 3d game objects, 2001.
URL https://graphics.stanford.edu/courses/cs468-01-fall/Papers/van-den-bergen.pdf

[2] S. Cameron, A study of the clash detection problem in robotics, in: Proceedings. 1985 IEEE International Conference on Robotics and Automation, Vol. 2, Institute of Electrical and Electronics Engineers, St. Louis, MO, USA, 1985, pp. 488–493. doi:10.1109/ROBOT.1985.1087245.
URL http://ieeexplore.ieee.org/document/1087245/

[3] M. N. Ardekani, P. Costa, W. P. Breugem, L. Brandt, Numerical study of the sedimentation of spheroidal particles, International Journal of Multiphase Flow 87 (2016) 16–34. doi:10.1016/j.ijmultiphaseflow.2016.08.005.
URL https://linkinghub.elsevier.com/retrieve/pii/S0301932216300556

[4] R. Jain, S. Tschisgale, J. Fröhlich, A collision model for DNS with ellipsoidal particles in viscous fluid, International Journal of Multiphase Flow 120 (2019) 103087. doi:10.1016/j.ijmultiphaseflow.2019.103087.
URL https://linkinghub.elsevier.com/retrieve/pii/S0301932219303453

[5] B. Lambert, L. Weynans, M. Bergmann, Methodology for numerical simulations of ellipsoidal particle-laden flows, International Journal for Numerical Methods in Fluids (2020) fld.4809doi:10.1002/fld.4809.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.4809

[6] S. Vincent, J. C. Brändle de Motta, A. Sarthou, J.-L. Estivalezes, O. Simonin, E. Climent, A Lagrangian VOF tensorial penalty method for the DNS of resolved particle-laden flows, Journal of Computational Physics 256 (2014) 582–614. doi:10.1016/j.jcp.2013.08.023.
URL https://linkinghub.elsevier.com/retrieve/pii/S0021999113005615

[7] J. C. Brändle de Motta, W.-P. Breugem, B. Gazanion, J.-L. Estivalezes, S. Vincent, E. Climent, Numerical modelling of finite-size particle collisions in a viscous fluid, Physics of Fluids 25 (8) (2013) 083302. doi:10.1063/1.4817382.
URL http://aip.scitation.org/doi/10.1063/1.4817382

[8] M. Chadil, S. Vincent, J.-L. Estivalèzes, Accurate estimate of drag forces using particle-resolved direct numerical simulations, Acta Mechanica 230 (2) (2019) 569–595. doi:10.1007/s00707-018-2305-1.
URL http://link.springer.com/10.1007/s00707-018-2305-1

[9] E. I. Thiam, E. Masi, E. Climent, O. Simonin, S. Vincent, Particle-resolved numerical simulations of the gas–solid heat transfer in arrays of random motionless particles, Acta Mechanica 230 (2) (2019) 541–567. doi:10.1007/s00707-018-2346-5.
URL http://link.springer.com/10.1007/s00707-018-2346-5

[10] A. Ozel, J. Brändle de Motta, M. Abbas, P. Fede, O. Masbernat, S. Vincent, J.-L. Estivalezes, O. Simonin, Particle resolved direct numerical simulation of a liquid–solid fluidized bed: Comparison with experimental data, International Journal of Multiphase Flow 89 (2017) 228–240. doi:10.1016/j.ijmultiphaseflow.2016.10.013.
URL https://linkinghub.elsevier.com/retrieve/pii/S0301932216302221

[11] J. Brändle de Motta, P. Costa, J. Derksen, C. Peng, L.-P. Wang, W.-P. Breugem, J. Estivalezes, S. Vincent, E. Climent, P. Fede, P. Barbaresco, N. Renon, Assessment of numerical methods for fully resolved simulations of particle-laden turbulent flows, Computers & Fluids 179 (2019) 1–14. doi:10.1016/j.compfluid.2018.10.016.
URL https://linkinghub.elsevier.com/retrieve/pii/S0045793018307709

[12] A. Wachs, Particle-scale computational approaches to model dry and saturated granular flows of non-Brownian, non-cohesive, and non-spherical rigid bodies, Acta Mechanica 230 (6) (2019) 1919–1980. doi:10.1007/s00707-019-02389-9.
URL http://link.springer.com/10.1007/s00707-019-02389-9

[13] A. Lin, S.-P. Han, On the Distance between Two Ellipsoids, SIAM J. Optim. 13 (2002) 298–308. doi:https://doi.org/10.1137/S1052623401396510.
URL https://epubs.siam.org/doi/10.1137/S1052623401396510

[14] E. Gilbert, D. Johnson, S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, IEEE Journal on Robotics and Automation 4 (2) (1988) 193–203. doi:10.1109/56.2083.
URL http://ieeexplore.ieee.org/document/2083/

[15] A. Wachs, L. Girolami, G. Vinay, G. Ferrer, Grains3D, a flexible DEM approach for particles of arbitrary convex shape — Part I: Numerical model and validations, Powder Technology 224 (2012) 374–389. doi:10.1016/j.powtec.2012.03.023.
URL https://linkinghub.elsevier.com/retrieve/pii/S003259101200191X

[16] L. Seelen, J. Padding, J. Kuipers, A granular Discrete Element Method for arbitrary convex particle shapes: Method and packing generation, Chemical Engineering Science 189 (2018) 84–101. doi:10.1016/j.ces.2018.05.034.
URL https://linkinghub.elsevier.com/retrieve/pii/S0009250918303312

[17] S. Zhao, J. Zhao, A poly-superellipsoid-based approach on particle morphology for DEM modeling of granular media, International Journal for Numerical and Analytical Methods in Geomechanics 43 (13) (2019) 2147–2169. doi:10.1002/nag.2951.
URL https://onlinelibrary.wiley.com/doi/10.1002/nag.2951

[18] M. E. Abbasov, New optimization algorithm for finding distance between two convex sets, in: 2015 International Conference "Stability and Control Processes" in Memory of V.I. Zubov (SCP), IEEE, Saint Petersburg, Russia, 2015, pp. 293–294. doi:10.1109/SCP.2015.7342128.
URL http://ieeexplore.ieee.org/document/7342128/

[19] M. E. Abbasov, Charged ball method for solving some computational geometry problems, Vestnik St. Petersburg University, Mathematics 50 (3) (2017) 209–216. doi:10.3103/S1063454117030025.
URL http://link.springer.com/10.3103/S1063454117030025

[20] M. E. Abbasov, F. Aliev, Modifications of the Charged Balls Method, Open Computer Science 10 (1) (2020) 30–32. doi:10.1515/comp-2020-0008.
URL http://www.degruyter.com/view/j/comp.2020.10.issue-1/comp-2020-0008/comp-2020-0008.xml

[21] G. S. Tamasyan, A. A. Chumakov, Finding the distance between ellipsoids, Journal of Applied and Industrial Mathematics 8 (3) (2014) 400–410. doi:10.1134/S1990478914030132.
URL http://link.springer.com/10.1134/S1990478914030132

[22] A. Y. Uteshev, M. V. Yashina, Computation of the distance from an ellipsoid to a linear surface and a quadric in $\mathbb{R}^n$, Doklady Mathematics 77 (2) (2008) 269–272. doi:10.1134/S1064562408020270.
URL http://link.springer.com/10.1134/S1064562408020270

[23] R. Jain, S. Tschisgale, J. Fröhlich, Impact of shape: DNS of sediment transport with non-spherical particles, Journal of Fluid Mechanics 916 (2021) A38. doi:10.1017/jfm.2021.214.
URL https://www.cambridge.org/core/product/identifier/S0022112021002147/type/journal_article

[24] R. Jain, S. Tschisgale, J. Fröhlich, A collision model for DNS with ellipsoidal particles in viscous fluid, International Journal of Multiphase Flow (2022) 104009doi:10.1016/j.ijmultiphaseflow.2022.104009.
URL https://linkinghub.elsevier.com/retrieve/pii/S0301932222000313

[25] G. van den Bergen, A Fast and Robust GJK Implementation for Collision Detection of Convex Objects, Journal of Graphics Tools 4 (2) (1999) 7–25. doi:10.1080/10867651.1999.10487502.
URL http://www.tandfonline.com/doi/abs/10.1080/10867651.1999.10487502

[26] X. Jia, Y.-K. Choi, B. Mourrain, W. Wang, An algebraic approach to continuous collision detection for ellipsoids, Computer Aided Geometric Design 28 (3) (2011) 164–176. doi:10.1016/j.cagd.2011.01.004.
URL https://linkinghub.elsevier.com/retrieve/pii/S016783961100015X

[27] D. Poelaert, J. Schniewind, F. Janssens, Surface Area and Curvature of the general Ellipsoid, arXiv:1104.5145 [math]ArXiv: 1104.5145 (Apr. 2011).
URL http://arxiv.org/abs/1104.5145

[28] W. Wang, J. Wang, M.-S. Kim, An algebraic condition for the separation of two ellipsoids, Computer Aided Geometric Design 18 (6) (2001) 531–539. doi:10.1016/S0167-8396(01)00049-8.
URL https://linkinghub.elsevier.com/retrieve/pii/S0167839601000498

[29] E. Ghossein, M. Lévesque, Random generation of periodic hard ellipsoids based on molecular dynamics: A computationally-efficient algorithm, Journal of Computational Physics 253 (2013) 471–490. doi:10.1016/j.jcp.2013.07.004.
URL https://linkinghub.elsevier.com/retrieve/pii/S0021999113004750