# Projet P6 : Modélisation de la collision de particules sphériques pour la simulation numérique d'un sablier 3D

Dassou Oliver, Durand Antoine, Garinot Antoine,
Lapointe Maxime, Papachristou Charles, Solaz Coralie

19 juin 2017

Date de remise du rapport : 19/06/2017

Référence du projet : STPI/P6/2017 – 54

Intitulé du projet : Modélisation de la collision de particules sphériques pour la simulation numérique d'un sablier 3D

Type de projet : Bibliographique, Numérique

Objectifs du projet :

- Comprendre l'importance de la modélisation
- Réussir à modéliser la collision d'une particule sphérique avec une paroi
- Résoudre numériquement l'équation du mouvement d'un solide
- Prendre en main un tableur, écrire un code de calcul
- Organiser un travail de groupe

# Table des matières

# Notations, Acronymes

École des corps déformables : École D
École des corps indéformables : École I

Coefficient de Poisson : $\sigma$
Coefficient de frottement : $\lambda$
Coefficient normal de restitution : $e$

Module de Young : $E$ $(GPa)$
Pulsation : $\omega$ $(rad.s^{-1})$
Raideur du ressort : $k_n$ $(N.m^{-1})$
Recouvrement (aussi appelé overlap) : $\delta_n$ $(m)$
Temps de contact : $t_{contact,n}$ $(s)$

# Introduction

Ce projet a été réalisé dans le cadre de l'EC P6, qui consiste à proposer une initiation à la conduite d'un projet en groupe aux étudiants. Notre sujet traite de modélisation physique et de simulation numérique. La modélisation est la représentation mathématique d'un système réel dans un contexte et une problématique donnés. Pour modéliser, il faut se rattacher à un modèle afin de rassembler tous les paramètres du système. En effet, notre modèle ne peut pas décrire parfaitement la situation réelle, sauf dans le cas de système simple. Dans la physique newtonienne, il existe plusieurs modèles : l'école des corps déformables et l'école des corps indéformables.

Dans le cadre d'une simulation, il est nécessaire de disposer de résultats d'équations physiques obtenues à l'aide des connaissances mathématiques apprises depuis STPI1. La simulation est un outil qu'utilise un ingénieur pour étudier les conséquences d'une action sur un élément sans réaliser l'expérience sur l'élément réel. Elle nous permet d'avoir un résultat se rapprochant de la réalité, sans avoir à réaliser cette expérience. Notre projet s'inscrit dans un domaine où beaucoup de recherches ont été publiées. En effet, l'étude des collisions de particules est très utile dans les milieux granulaires.

Dans un premier temps, nous définirons le cadre de résolution, pour avoir une simulation cohérente. Puis, à l'aide de ce modèle, nous résoudrons les équations de collision nécessaires pour la simulation numérique que nous développerons dans la troisième partie de ce rapport.

# Chapitre 1

# Organisation et répartition du travail

Le travail réalisé au début de ce projet consistait principalement à faire des recherches bibliographiques sur le domaine de la collision de grains mais aussi sur les modèles de résolution des équations différentielles (Runge-Kutta, Euler, etc). Nous avons donc fait un travail de documentation pendant les premières séances afin de mieux comprendre où se situait notre projet dans les modèles déjà existants.

Nous nous sommes répartis le travail entre la partie théorique (résolution des équations différentielles et définition du modèle physique) et le codage des programmes. Étant un groupe de 6, nous avons fonctionné avec 3 binômes de 2 : Antoine G et Maxime ont travaillé sur la résolution des équations différentielles et le calcul théorique des différentes grandeurs physiques. Coralie et Olivier se sont chargés du code du programme informatique en langage python. Enfin, Charles et Antoine D ont défini le modèle physique et les différentes hypothèses nécessaires à la résolution. Lorsque les différentes parties étaient terminées, nous avons mis en commun nos travaux pour la rédaction du rapport.

Cette organisation nous a permis d'échanger tout au long du projet. En effet, pour réaliser le programme informatique, Coralie et Olivier avaient besoin des différentes hypothèses physiques du modèle mais aussi des calculs théoriques sur les grandeurs physiques.

# Chapitre 2

# Définition du cadre

## 2.1 Introduction à l'école des corps indéformables et déformables

Les précurseurs du modèle des corps déformables (école D) sont Cundall (1971), puis Cundall & Strack (1979) avec leur modèle des éléments distincts. Pour l'école D, des lois de répulsion (ressort) sont utilisées pour modéliser les contraintes mécaniques de non-interpénétrabilité de chaque paire de particules. En effet, lorsque deux particules sphériques sont en contact, ces dernières peuvent se chevaucher légèrement (phénomène d'overlap). Ainsi, les contacts ont une durée non nulle et les interactions entre les particules varient de façon continue au cours du chevauchement. Il faut donc intégrer une équation différentielle du second ordre pour calculer l'évolution du système et l'overlap.

Pour l'école des corps indéformables (école I), le choc est considéré comme instantané. De plus, le contact a lieu en un point, et non en une zone de recouvrement. Après un choc, chaque nouvelle vitesse des particules sphériques est calculée à l'aide de coefficients de friction et de restitution. La méthode de résolution développée par l'école I s'adapte bien à la simulation numérique car le calcul des nouvelles vitesses est simple. Le principal inconvénient réside dans la détection des collisions. En effet, pour ce modèle, il faut être capable de déterminer précisément le point de contact et le moment de l'impact. De plus, il est proscrit que les particules se chevauchent, contrairement à l'école D.

La probabilité que deux particules soient en contact à un instant donné est quasi nulle avec l'école I. Tandis que l'autre école permet dans un milieu suffisamment dense que des particules sphériques voisines soient en contact. L'école D est donc bien adaptée aux milieux denses quasi statiques, et la seconde aux milieux peu denses et dynamiques.

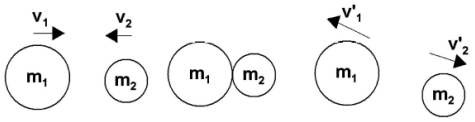
## 2.2 Choix du modèle

### 2.2.1 Définition du modèle des sphères dures

Le modèle des sphères dures (aussi appelées sphères rigides) émane de l'idéologie développée par l'école I. Ce modèle repose sur le principe que les particules ne subissent aucune déformation lors de leurs collisions. En effet, nous considérons qu'il y a une force de répulsion entre les sphères. La collision est instantanée et les particules ne se chevauchent pas. [1]

Ce type de modèle est proche de ceux utilisés en dynamique moléculaire et est utile pour définir des gaz rares, tous les gaz dilués ainsi que les liquides simples.

**Modèle des sphères dures**




**Pre-Contact** **Contact** **Post-Contact**

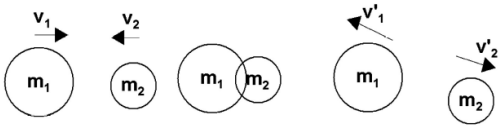
Temps de contact ≈ 0 et pas d'overlap

(m1, m2 : masse des particules)
(v1, v2 : vitesse des particules ; v1', v2' : vitesse des particules après le choc)

### 2.2.2 Définition du modèle des sphères molles

Dans le modèle des sphères molles, la particule recouvre partiellement la paroi pendant le choc. C'est pourquoi il est possible d'assimiler ce modèle à l'école D. Le choc n'est pas instantané, il n'y a pas un seul point de contact mais un recouvrement. Par conséquent, il faut résoudre l'équation pour plusieurs pas de temps à chaque contact entre deux particules ou entre une particule et un obstacle.

**Modèle des sphères molles**




**Pre-Contact** **Contact** **Post-Contact**

Temps de contact > 0 et overlap autorisé

(m1, m2 : masse des particules)
(v1, v2 : vitesse des particules ; v1', v2' : vitesse des particules après le choc)

1. https ://www.princeton.edu/cbe/people/faculty/sundaresan/group/publications/pdf/110.pdf

### 2.2.3 Comparaison des modèles

Chaque modèle peut être avantageux suivant le type de milieu. Dans le cas de l'école D, il est possible de rendre compte de contacts multiples simultanés. Pour l'école I, les vitesses sont recalculées après le choc entre les deux particules. Il est nécessaire de calculer les temps exacts des collisions, afin de pouvoir ordonner les chocs. Si nous voulions détecter les chocs de la même manière que pour l'école D, il y aurait un problème lorsque plusieurs particules rentreraient en contact, puisque que nous ne connaîtrons pas l'ordre dans lequel ont eu lieu les chocs. En ce sens, si nous choisissons de décrire le contact entre deux objets, nous pouvons en effet trouver une solution analytique simple. Cependant, dès lors que nous choisissons de décrire le contact de plus de deux objets, nous sommes obligés de passer par une discrétisation.

C'est pourquoi nous avons choisi le modèle des sphères molles. Dans le cas où il y a n particules, ce dernier nous permet de résoudre les équations à l'ordre n+1 beaucoup plus facilement qu'avec le modèle des sphères rigides. Le temps de contact et l'overlap permettent de donner des conditions de stabilité lorsqu'on modélise avec le modèle des sphères molles.

| | École des corps déformables | École des corps indéformables |
|---|---|---|
| Durée d'un choc | $> 0$ | $\approx 0$ |
| Influence du pas de temps sur les résultats | Non négligeable | Nulle pour une grande précision |
| Choix des paramètres physiques | Difficile | Assez facile |
| Densité du milieu | Dense | Peu dense |
| Résolution | Discrétiser le temps | Calculer les temps de collision |

## 2.3 Choix des paramètres physiques

L'inconvénient majeur de ce modèle, que nous retrouvons dans le modèle des corps indéformables, est la valeur à donner aux paramètres de collision. Ici, l'un des paramètres numériques à définir est la raideur $k_n$ décrit dans le modèle des sphères molles. Ce dernier peut prendre n'importe quelle valeur, mais peut être lié à des propriétés mécaniques des matériaux mis en jeu. Ce paramètre doit être déterminé expérimentalement. Cependant, il est possible de trouver une formule qui permet de retrouver sa valeur réelle. Prenons l'exemple de la raideur du ressort $k_n$.

$$k_n = \frac{ER}{3(1-\sigma)}$$

avec $E$ le module de Young du matériau, $\sigma$ le coefficient de Poisson du matériau et $R$ le rayon d'une particule.

**Calcul de différentes constantes de raideur (en N/m)**

| Matériau | Coefficient de Poisson | Module de Young (GPa) | Constante de raideur (N/m) |
|---|---|---|---|
| Verre | 0,25 | 69 | $3{,}1.10^{8}$ |
| Acier | 0,5 | 0,01 | $1{,}0.10^{9}$ |

Nous obtenons des valeurs de constante de raideur cohérentes grâce aux formules théoriques. Cependant, si nous voulons avoir un modèle exploitable, nous permettant de résoudre les équations numériquement, il est primordial d'adapter la valeur de la constante de raideur $k_n$. Nous devons donc adapter cette valeur, jusqu'à obtenir un critère de stabilité correct et un pas de temps modéré.

L'autre grandeur physique importante est l'overlap, c'est-à-dire le recouvrement autorisé entre la paroi et la sphère.

Dans notre modèle, nous avons la relation suivante pour le temps de contact $t_{contact,n}$ :

$$t_{contact,n} = \frac{\pi}{\omega}$$

avec $\omega$ la pulsation :

$$\omega = \sqrt{\frac{k_n}{m}}$$

Le temps de contact et le recouvrement ($\delta_n$) sont deux autres grandeurs physique qui interviennent lors de la collision.

## 2.4 Hypothèses nécessaires à la résolution

Comme décrit précédemment, l'un des inconvénients du modèle des sphères molles réside dans la nécessité de définir des paramètres de collision. Pour modéliser le contact entre 2 particules sphériques, nous avons choisi le modèle des sphères molles. Cependant, nous devons faire plusieurs hypothèses qui permettent d'utiliser un modèle le plus adapté possible à notre système. En effet, le modèle des sphères molles est la base de notre résolution, mais nous devons définir un cadre plus précis afin de réaliser la modélisation des particules.

- Nous avons choisi le modèle des sphères molles (c'est à dire qu'une sphère n'est définie que par un point central et par un rayon) par conséquent il ne peut pas y avoir déformation de la matière (mais il y a un chevauchement).

- Nous nous placerons uniquement dans la cadre d'une collision normale. En effet, la collision tangentielle étant plus difficile à implémenter, nous ne l'avons pas incluse dans notre projet. En ce sens, les grandeurs physiques du contact ($t_{contact,n}$, $\delta_n$, etc) seront calculées sur la composante normale du contact.

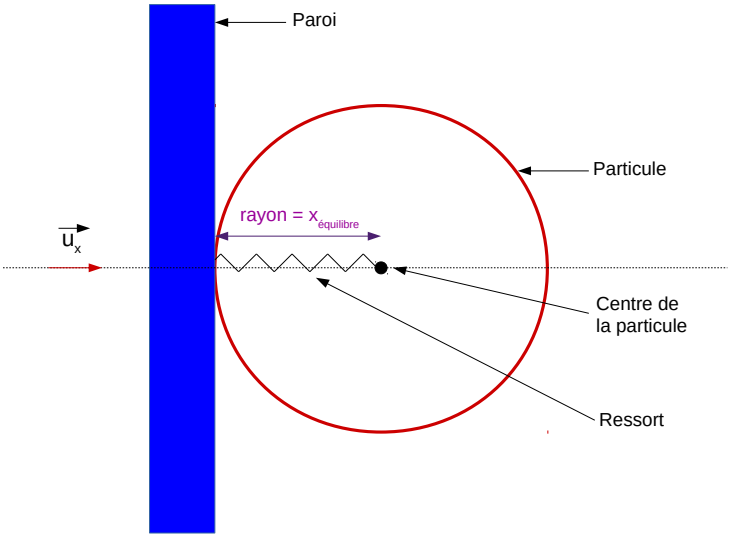
- Nous supposons que la particule est un corps homogène et isotrope, sinon la résolution des équations différentielles devient trop complexe.

# Chapitre 3

# Résolution physique du problème

Pour la réalisation de notre projet, nous avons choisi le modèle des sphères molles pour modéliser le choc des particules. Pour la résolution, nous représentons donc chacune des particules comme un point concentrant sa masse et un ressort ayant pour longueur son rayon. Nous nous intéresserons ici seulement au contact normal entre une particule et une paroi.

### Résolution système masse-ressort :



Ici $x$ représente la position du centre de gravité de la particule. $\dot{x}$ et $\ddot{x}$ représentent respectivement sa vitesse et son accélération. De plus, nous nous plaçons dans le référentiel terrestre que nous supposerons galiléen.
Bilan des forces exercées sur la particule lors de la collision :

- Ressort : $\overrightarrow{F_{ressort}} = -k \times (x - x_{équilibre}).\overrightarrow{u_x}$ avec $x_{équilibre}$ qui correspond au rayon de la particule.
- Forces dissipant l'énergie : $\overrightarrow{F_{dissipation}} = -\lambda \times \dot{x}.\overrightarrow{u_x}$ avec $\lambda$ qui correspond au coefficient de dissipation de l'énergie. En effet, lors d'une collision, il n'y a pas nécessairement conservation de l'énergie. Cette force explique pourquoi la particule ne repart pas à la même vitesse après collision.
- Nous négligeons le poids de la particule dans la mesure où une collision est un phénomène instantané.

## Application du principe fondamental de la dynamique (PFD) :

D'après la deuxième loi de Newton nous avons : $\overrightarrow{F}_{totale} = m \times \ddot{x}.\overrightarrow{u_x}$ avec $\overrightarrow{F_{totale}} = \overrightarrow{F_{ressort}} + \overrightarrow{F_{dissipation}}$. Ce qui nous donne donc :

$$m \times \ddot{x}.\overrightarrow{u_x} = -\lambda \times \dot{x}.\overrightarrow{u_x} - k \times (x - x_{équilibre}).\overrightarrow{u_x}$$

Nous projetons sur $\overrightarrow{u_x}$ :

$$\begin{aligned} m \times \ddot{x} &= -\lambda \times \dot{x} - k \times (x - x_{équilibre}) \\ 0 &= m \times \ddot{x} + \lambda \times \dot{x} + k \times (x – x_{équilibre}) \end{aligned}$$

Nous posons $\delta_n = (x – x_{équilibre})$
Donc $\dot{\delta_n} = (x – x_{équilibre})' = \dot{x}$ et $\ddot{\delta_n} = \ddot{x}$
Il faut donc résoudre l'équation différentielle :

$$\begin{aligned} m \times \ddot{\delta_n} + \lambda \times \dot{\delta_n} + k \times \delta_n &= 0 \\ \ddot{\delta_n} + \frac{\lambda}{m} \times \dot{\delta_n} + \frac{k}{m} \times \delta_n &= 0 \\ \ddot{\delta_n} + 2 \times a \times \dot{\delta_n} + \omega^{\mathbf{2}} \times \delta_n &= 0 \end{aligned}$$

avec $a = \frac{\lambda}{(2 \times m)}$ et $\omega = \sqrt{\frac{k}{m}}$
Equation caractéristique :

$$x^{\mathbf{2}} + \frac{\lambda}{m} \times x + \frac{k}{m} = 0$$

Nous avons donc :

$$\begin{aligned} \triangle &= (\frac{\lambda}{m})^2 – 4 \times \frac{k}{m} \\ &= \frac{(\lambda^2 – 4 \times k \times m)}{m^{\mathbf{2}}} \end{aligned}$$

Nous pouvons aussi écrire :

$$\begin{aligned} \triangle &= 4 \times a^{\mathbf{2}} – 4 \times \omega^{\mathbf{2}} \\ &= 4 \times (a^{\mathbf{2}} – \omega^2) \end{aligned}$$

$m > 0$ donc $\triangle$ est du signe du numérateur. Nous nous plaçons dans le cas où $\triangle < 0$. En effet, si $\triangle \geqslant 0$, nous serions dans le cas d'un ressort amorti sans oscillations. Or lors d'un choc la particule repart avec une vitesse non nulle. Avec $\triangle < 0$, nous nous plaçons alors dans le cas d'un ressort avec oscillations. Il faut donc :

$$\begin{aligned} \triangle &< 0 \\ \lambda^{\mathbf{2}} – 4 \times k \times m &< 0 \\ \lambda^2 &< 4 \times k \times m \end{aligned}$$

où l, k et m sont des valeurs positives.
Les racines de l'équation caractéristiques sont :
$racine_1 = r + i \times s$
$racine_2 = r – i \times s$
avec ici :

$$r = -a = \frac{-\lambda}{(2 \times m)}$$

$$s = \sqrt{\frac{-\triangle}{2}} = \sqrt{\omega^2 - a^{\mathbf{2}}}$$

Finalement, nous obtenons l'équation de l'overlap en fonction du temps :

$$\delta_n(t) = exp(r \times t) \times K \times sin(s \times t + \varphi)$$

avec K et $\varphi$ qui dépendent des conditions initiales. Nous considérons l'origine du temps lorsque la particule entre en contact avec la paroi ce qui donne un overlap nul et une vitesse initiale égale à la vitesse de la particule lorqu'elle arrive en contact avec la paroi :

$$\begin{aligned} \delta_n(0) &= K \times exp(r \times 0) \times sin(s \times 0 + \varphi) \\ &= K \times sin(\varphi) \\ &= 0 \end{aligned}$$

D'où $\varphi = 0$. En effet $K$ ne peut pas être nul sinon nous aurions $\delta_n(t) = 0$ quelque soit $t$. Ensuite :

$$\begin{aligned} \dot{\delta_n}(0) &= K \times exp(r \times 0) \times (r \times sin(s \times 0) + s \times cos(s \times 0)) \\ &= K \times s \\ &= v_0 \end{aligned}$$

D'où $K = \frac{v_0}{s}$.

Nous allons maintenant calculer le temps de contact normal en fonction des paramètres. $t_{contact,n}$ est donné lorsque l'overlap devient à nouveau nul, c'est-à-dire quand le sinus a effectué une demi-période. Ce qui nous donne :

$$\begin{aligned} s \times t_{contact,n} &= \pi \\ t_{contact,n} &= \frac{\pi}{s} \end{aligned}$$

Nous pouvons donc maintenant déterminer le coefficient normal de restitution qui est donné par la vitesse de la molécule après le choc sur la vitesse initiale :

$$\begin{aligned} e &= \left| \frac{\dot{\delta}(t_{contact,n})}{\dot{\delta}(0)} \right| \\ &= \frac{K \times exp(r \times t_{contact,n}) \times s}{K \times s} \\ &= exp(r \times t_{contact,n}) \\ &= exp(\pi \times \frac{r}{s}) \end{aligned}$$

Il est donc possible de calculer le coefficient de frottement normal en fonction des autres paramètres :

$$\begin{aligned} e &= exp(\pi \times \frac{r}{s}) \\ ln(e) &= \pi \times \frac{r}{s} \\ \pi \times r &= \sqrt{\omega^2 - a^2} \times ln(e) \\ \pi^2 \times r^2 &= \left( \omega^2 - \frac{\lambda^2}{(4 * m^2)} \right) \times ln^2(e) \\ \omega^2 \times ln^2(e) &= (\pi^2 + ln^2(e)) \times \frac{\lambda^2}{(4 \times m^2)} \\ \lambda &= \frac{-2 \times \sqrt{m \times k} \times ln(e)}{\sqrt{\pi^2 + ln^2(e)}} \end{aligned}$$

Ensuite pour estimer le temps de l'overlap maximum $t_{\delta_{max}}$, il faut trouver le maximum de $x(t) = K \times exp(r \times t) \times sin(st)$ avec $t \in \left[0, \frac{\pi}{s}\right]$. Cet intervalle correspond à la durée de l'overlap $\delta$. Pour cela , nous allons chercher à savoir quand sa dérivée s'annule, c'est-à-dire quand :

$$\dot{x}(t) = K \times exp(r \times t_{\delta_{max}}) \times (r \times sin(s \times t_{\delta_{max}}) + s \times cos(s \times t_{\delta_{max}})) = 0$$

Nous trouvons que cette dérivée s'annule quand :

$$\begin{aligned} r \times sin(s \times t_{\delta_{max}}) + s \times cos(s \times t_{\delta_{max}}) &= 0 \\ r \times sin(s \times t_{\delta_{max}}) &= -s \times cos(s \times t_{\delta_{max}}) \\ \frac{sin(s \times t_{\delta_{max}})}{cos(s \times t_{\delta_{max}})} &= \frac{-s}{r} \\ tan(s \times t_{\delta_{max}}) &= \frac{-s}{r} \\ t_{\delta_{max}} &= \frac{arctan\left(\frac{-s}{r}\right)}{s} \\ t_{\delta_{max}} &= \frac{-arctan\left(\frac{s}{r}\right)}{s} \end{aligned}$$

Pour trouver l'overlap maximum, il suffit d'évaluer l'équation de l'overlap en $t_{\delta_{max}}$.

Dans le cas d'une collision entre deux particules, nous appliquons le PFD sur la particule 1 : $m_1 \times a_1.\overrightarrow{u} = F_{1,totale}.\overrightarrow{u} \Longrightarrow a_1.\overrightarrow{u} = \frac{1}{m_1} \times F_{1,totale}.\overrightarrow{u}$ avec $\overrightarrow{u}$ le vecteur porté par le centre des deux particules. De même sur la particule 2 : $m_2 \times a_2.\overrightarrow{u} = F_{2,totale}.\overrightarrow{u} \Longrightarrow a_2.\overrightarrow{u} = \frac{1}{m_2} \times F_{2,totale}.\overrightarrow{u}$.

Nous appliquons alors la 3ème loi de Newton : $F_{1,totale} = -F_{2,totale} = F_{interaction}$

$$(a_1 - a_2).\overrightarrow{u} = \frac{1}{m_1} \times F_{1,totale}.\overrightarrow{u} - \frac{1}{m_2} \times F_{2,totale}.\overrightarrow{u}$$

Nous projetons sur $\overrightarrow{u}$ et nous remarquons que $(a_1 - a_2)$ équivaut à l'accélération de l'overlap des particules :

$$\begin{aligned} \ddot{\delta} &= \left(\frac{1}{m_1} + \frac{1}{m_2}\right) \times F_{interaction} \\ m_{eff} \times \ddot{\delta} &= F_{interaction} \end{aligned}$$

En posant $\frac{1}{m_{eff}} = \frac{1}{m_1} + \frac{1}{m_2}$.

Ensuite, nous pouvons choisir d'écrire $F_{interaction}$ sous la forme $-\lambda \times \dot{\delta} - k \times \delta$ avec de nouveau, $\lambda$ comme coefficient de dissipation de l'énergie lors de la collision et $k$ comme coefficient de raideur résultant des ressorts des deux particules, dans le cadre du modèle des sphères molles. Ainsi nous retrouvons la même équation qu'avec une collision paroi-particule, seulement la masse est remplacée par la masse réduite des deux particules.

Nous pouvons remarquer que si nous appliquons ce modèle pour une collision paroi-particule, il faudrait remplacer une des masses par la masse de la paroi ce qui revient à lui mettre une masse infinie. Ainsi, $m_{eff} = m_{particule}$ et nous retombons sur l'équation que nous avions au début.

# Chapitre 4

# Méthodes numériques

## 4.1 Qu'est-ce qu'une méthode numérique ? Pourquoi les avons-nous utilisées ?

Les méthodes numériques permettent d'approcher les solutions d'une équation différentielle. Elles consistent à discrétiser le temps en fixant un pas de temps $\triangle t$ puis de calculer les solutions de l'équation de « proche en proche » : $f\left(x_{t+\triangle t}\right)$, $f\left(x_{t+2\triangle t}\right)$, $f\left(x_{t+3\triangle t}\right)$, $f\left(x_{t+4\triangle t}\right)$... Nous avons utilisé des méthodes numériques car cela fait partie des objectifs du projet et nous montre un nouveau moyen de résoudre un problème physique décrit par des équations différentielles. De plus, certains problèmes ne possèdent pas de solution analytique, les méthodes numériques sont alors le seul moyen d'approcher les solutions. Nous avons vu dans la partie précédente que nous avons une solution analytique. Elle nous sert de référence donc nous allons l'utiliser pour comparer la précision des méthodes utilisées lors de ce projet.

Les critères d'une bonne méthode numérique sont :

- L'erreur locale de troncature ($\epsilon$) : C'est l'écart entre la solution de la méthode numérique employée et la solution exacte de la méthode analytique
- La stabilité : Une méthode est stable si la propagation d'erreur est minimale
- La convergence : Il s'agit d'une propriété théorique assurant que l'erreur locale de troncature tend vers 0 lorsque le pas de temps tend vers 0
- L'ordre : Une méthode est dite d'ordre p si $\epsilon = (\triangle t)^{p+1}$.

## 4.2 Rappel des équations différentielles

Comme vu au chapitre 4, à l'aide du principe fondamental de la dynamique on obtient le système d'équations différentielles suivant :

$$\frac{d\dot{\delta}_n}{dt} = \frac{\sum F_{(\delta_n, \dot{\delta}_n)}}{m}$$

$$\frac{d\delta_n}{dt} = \dot{\delta}_n$$

## 4.3 Les méthodes choisies

Nous avons sélectionné et codé 4 méthodes :

— la méthode d'Euler : elle tient son nom de son inventeur Leonhard Euler (1707-1783)

— deux méthodes de Runge-Kutta : elles tirent leur nom des deux mathématiciens Carl Runge et Martin Wilhelm Kutta qui les inventèrent en 1901.

— la méthode de LeapFrog / Verlet : cette méthode a été mise au point à la fin du 18ème siècle.

Afin de programmer chacune de ces méthodes, il est nécessaire de connaître les conditions initiales avant l'impact :

— l'overlap initial (que l'on a fixé à 0) : $\delta\,(t=0) = \delta_0 = 0$

— la vitesse initiale : $\dot{\delta}\,(t=0) = \dot{\delta}_0$

— le pas de temps : $\triangle t$

Dans notre modélisation, pour toutes les méthodes, le calcul le plus coûteux est celui de la somme des forces. Pour modéliser l'évolution du recouvrement, nous prendrons donc en compte le nombre de fois où l'on répète ce calcul comme critère important de sélection d'une méthode.

### 4.3.1 Euler

**Intérêt et particularité de la méthode**

Nous avons d'abord commencé à nous intéresser à la méthode d'Euler parce que c'est la plus simple des méthodes. De plus, elle ne nécessite de calculer qu'une seule fois la somme des forces pour placer un point. Elle est donc très économique.

**Algorithme**

$$\dot{\delta}_{n+1} = \dot{\delta}_n + \triangle t \frac{\sum F_{\left(\delta_n, \dot{\delta}_n\right)}}{m}$$

$$\delta_{n+1} = \delta_n + \triangle t \dot{\delta}_{n+1}$$

### 4.3.2 Runge-Kutta 2

**Intérêt et particularité de la méthode**

Ensuite, la méthode qui est très souvent utilisée pour modéliser une collision est celle de Runge-Kutta 2 (RK2). Elle permet de gagner en précision mais le calcul des forces s'effectue deux fois. Elle est donc deux fois plus coûteuse que la méthode d'Euler.

**Algorithme**

Première étape :

$$\dot{\delta}_{n+\frac{1}{2}} = \dot{\delta}_n + \frac{\triangle t}{2}\frac{\sum F_{(\delta_n, \dot{\delta}_n)}}{m}$$

$$\delta_{n+\frac{1}{2}} = \delta_n + \frac{\triangle t}{2}\dot{\delta}_n$$

Deuxième étape :

$$\dot{\delta}_{n+1} = \dot{\delta}_n + \triangle t\frac{\sum F_{\left(\delta_{n+\frac{1}{2}}, \dot{\delta}_{n+\frac{1}{2}}\right)}}{m}$$

$$\delta_{n+1} = \delta_n + \triangle t\dot{\delta}_{n+1}$$

### 4.3.3 Runge-Kutta 3

**Intérêt et particularité de la méthode**

Nous avons voulu tester la méthode de Runge-Kutta 3 (RK3) pour comparer avec Runge-Kutta 2. Sachant qu'elle est trois fois plus coûteuse que la méthode d'Euler à cause du calcul des forces qui s'effectue trois fois, nous voulions déterminer si ce coût était compensé par un gain important de précision. C'est aussi pourquoi nous nous sommes arrêtés à RK3 alors qu'on peut construire des méthodes de Runge-Kutta d'ordre supérieur.

**Algorithme**

Première étape :

$$\dot{\delta}_{n+\frac{1}{3}} = \dot{\delta}_n + \frac{\triangle t}{3}\frac{\sum F_{(\delta_n, \dot{\delta}_n)}}{m}$$

$$\delta_{n+\frac{1}{3}} = \delta_n + \frac{\triangle t}{3}\dot{\delta}_n$$

Deuxième étape :

$$\dot{\delta}_{n+\frac{1}{2}} = \dot{\delta}_n + \frac{\triangle t}{2}\frac{\sum F_{\left(\delta_{n+\frac{1}{3}}, \dot{\delta}_{n+\frac{1}{3}}\right)}}{m}$$

$$\delta_{n+\frac{1}{2}} = \delta_n + \frac{\triangle t}{2}\dot{\delta}_{n+\frac{1}{3}}$$

Troisième étape :

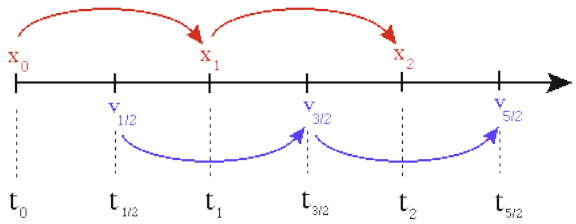
$$\dot{\delta}_{n+1} = \dot{\delta}_n + \triangle t\frac{\sum F_{\left(\delta_{n+\frac{1}{2}}, \dot{\delta}_{n+\frac{1}{2}}\right)}}{m}$$

$$\delta_{n+1} = \delta_n + \triangle t\dot{\delta}_{n+\frac{1}{2}}$$

### 4.3.4 LeapFrog / Verlet

**Intérêt et particularité de la méthode**

La méthode de LeapFrog / Verlet est d'ordre 2 comme RK2 mais ne nécessite qu'un calcul de la somme des forces. Elle est basée sur un système de saut. En effet, l'overlap et la vitesse ne sont pas calculés aux mêmes instants. Ils ont un demi pas de temps d'écart comme on peut le voir sur le graphique ci-dessous.




Cependant, la méthode originale a dû nécessiter une adaptation dans notre cas. En effet, cette méthode fut à l'origine utilisée dans le domaine de la Dynamique Moléculaire où les forces ne dépendent pas de la vitesse or notre équation de la somme des forces contient la vitesse. Nous calculons la somme des forces à un instant $n+1$ mais nous n'avons la vitesse qu'à l'instant $n+\frac{1}{2}$ .

**Algorithme**

Tout d'abord, nous initialisons la vitesse. En effet, les conditions initiales ne nous donnent que la vitesse à l'instant 0 ($\dot{\delta}_0$) alors que nous avons besoin de $\dot{\delta}_{\frac{1}{2}}$. Nous trouvons $\dot{\delta}_{\frac{1}{2}}$ grâce à l'équation suivante :

$$\dot{\delta}_{\frac{1}{2}} = \dot{\delta}_0 + \frac{\triangle t}{2}\frac{\sum F_{(\delta_0,\dot{\delta}_0)}}{m}$$

Partie générale :

$$\delta_{n+1} = \delta_n + \triangle t \dot{\delta}_{n+\frac{1}{2}}$$

$$\dot{\delta}_{n+\frac{3}{2}} = \dot{\delta}_{n+\frac{1}{2}} + \triangle t \frac{\sum F_{\left(\delta_{n+1},\dot{\delta}_{n+\frac{1}{2}}\right)}}{m}$$

## 4.4 Notre programme Python

Nous avons créé un programme en langage Python qui nous permet de résoudre notre système d'équations différentielles et tracer les courbes en utilisant une ou plusieurs des 4 méthodes. Notre code peut être consulté en intégralité dans la partie annexe.

**Comment l'utiliser ?**

1. L'utilisateur est d'abord confronté à 2 choix :
   — soit il rentre les conditions initiales lui-même, c'est-à-dire :
      (a) la position initiale de la particule
      (b) sa vitesse initiale
      (c) le rayon de la particule
      (d) de quoi elle est faite (verre ou acier)
      (e) le coefficient de restitution
      (f) le pourcentage d'overlap maximum autorisé durant le contact
   — soit il utilise les valeurs par défaut déjà entrées dans le programme.
2. Le programme affiche la masse de la particule étudiée.
3. L'utilisateur rentre la ou les méthodes qu'il veut utiliser.
4. Le programme trace la ou les courbe(s) de l'overlap en fonction du temps avec les méthodes utilisées.

**Comment fonctionne-t-il ?**

Notre programme fonctionne avec 3 unités que nous avons créées :

— « partie_utilisateur.py » : elle contient toutes les fonctions permettant de demander à l'utilisateur les conditions initiales et les méthodes qu'il veut utiliser.
— « partie_graphique.py » : elle contient toutes les fonctions qui permettent de tracer les courbes.
— « methode_resolution.py » : elle contient toutes les fonctions qui permettent de calculer la position et la vitesse à un instant $n + 1$ pour chaque méthode à l'aide des conditions initiales ainsi que de la position et de la vitesse à l'instant $n$.

Quant au programme principal, celui demande d'abord les conditions initiales. Le programme calcule automatiquement, à l'aide des formules trouvées au chapitre 4 notamment :

— la masse $m$
— le coefficient de raideur $k$
— l'overlap $\delta$
— le temps de contact $t_c$
— le pas de temps $\triangle t$ (qui doit être inférieur au temps de contact)

Le temps est aussi initialisé au moment où la particule entre en contact avec la paroi :

$$t = \frac{x_0 - rayon}{v_0}$$

Le programme demande ensuite à l'utilisateur les méthodes qu'il veut utiliser. Enfin, le programme stocke les résultats dans un tableau dont la première colonne contient les valeurs de t et dont les 4 autres colonnes contiennent la position de la particule à un instant t qui est calculée grâce aux 4 méthodes numériques. Pour terminer, la programme utilise les fonctions de la partie graphique pour tracer le graphique à l'aide du tableau de valeurs.

## 4.5 Ordre et précision des méthodes

Maintenant que nous avons résolu notre équation différentielle, il faut à présent vérifier l'ordre des méthodes utilisées. Retrouver l'ordre d'une méthode revient à vérifier que :

$$\epsilon \sim \triangle t^n$$

avec $\epsilon = max\left(|\delta_{sol.analytique}\left(t\right) - \delta_{sol.méthode}\left(t\right)|\right)$, $\triangle t$ le pas de temps et $n$ l'ordre de la méthode. Pour l'équation de la solution analytique, nous avons repris celle étudiée au chapitre 4. Pour cela, il faut tracer la droite dont l'équation est $log\left(\epsilon\right) = f\left(log\left(\triangle t\right)\right)$. Théoriquement, la fonction $f$ doit être linéaire pour un intervalle donné et le coefficient directeur correspond à l'ordre $n$ de la méthode

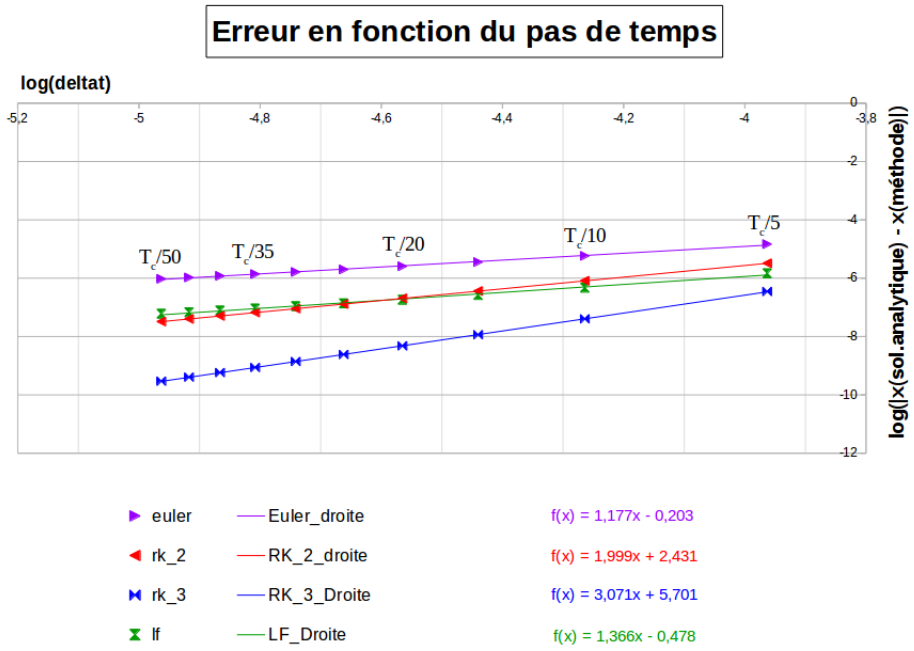
$$\begin{aligned} \epsilon &= \triangle t^n \\ log\left(\epsilon\right) &= log(\triangle t^n) \\ log\left(\epsilon\right) &= nlog\left(\triangle t\right) \end{aligned}$$

Comment avons - nous vérifié ces méthodes ? Tout d'abord, nous avons codé nos 4 méthodes (Euler, Runge-Kutta d'ordres 2 et 3, LeapFrog) sur LibreOffice Calc ainsi que la solution analytique de la position. Nous avons ensuite tracé les 4 courbes pour déterminer l'erreur et voir quelle méthode était la plus précise. Nous avons ensuite calculé les écarts (en valeur absolue) et trouver le max de ces écarts pour des pas de temps différents :

$$\triangle t = \frac{t_{contact}}{k}$$

avec $k \in [\![15; 50]\!]$ et $t_{contact}$ le temps de contact. En effet, certains auteurs préconisent un choix de k allant de 15 à 50. Cependant, dans l'objectif de réduire au maximum les temps de calcul, nous avons également réalisé des tests pour deux valeurs inférieures à 20 : nous avons rajouté les valeurs $k = 5$ et $k = 10$ . Nous calculons le log du pas de temps et de l'erreur sur la position. Puis, nous traçons la courbe.

Voici les résultats obtenus avec les paramètres suivants :
— masse = 0,003 kg
— position initiale = 1 m
— vitesse initiale = 0,1 m/s
— rayon de la particule = 0,00001 m
— coefficient de restitution = 0,9



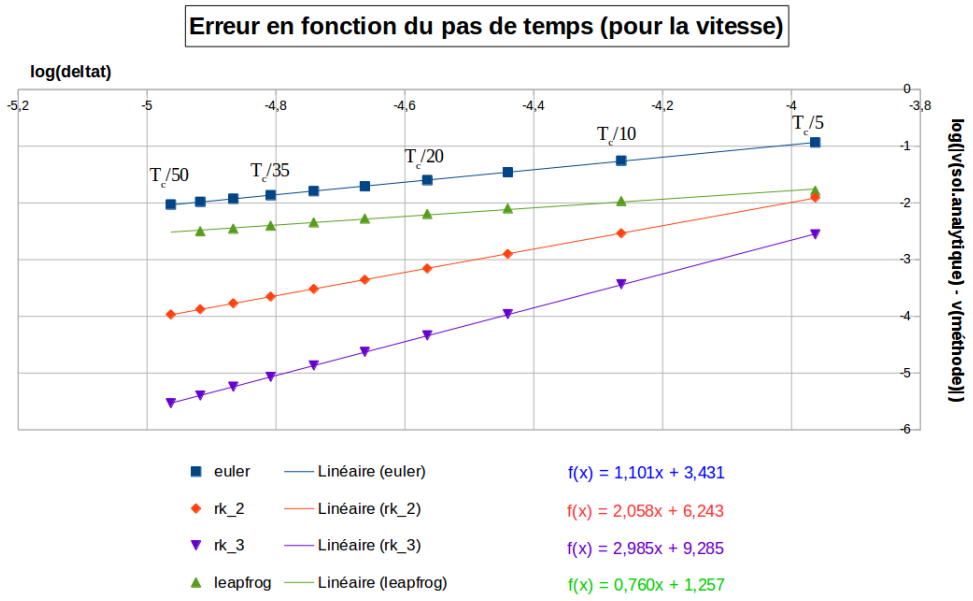

| Méthode | Erreur (en %) | Ordre calculé | Ordre théorique |
|---|---|---|---|
| Euler | 0.0003 | 1.177 | 1 |
| RK2 | 0.00005 | 1.999 | 2 |
| RK3 | 0.000004 | 3.071 | 3 |
| LeapFrog | 0.00003 | 1.366 | 2 |

TABLE 4.1 – Résultats obtenus

L'ordre calculé pour LeapFrog n'est pas égal à l'ordre recherché (qui vaut 2) car, comme expliqué précédemment, nous avons adapté la méthode pour qu'elle fonctionne dans notre cas. Cependant, lorsqu'on fixe un coefficient de restitution égal à 1 ce qui est le cas lorsqu'il n'y a pas de dissipation alors la force ne dépend plus de la vitesse. On retrouve alors l'ordre 2 théorique.

Euler est la moins précise des méthodes tandis que RK3 est la plus précise. Nous remarquons que lorsque le temps de contact est divisé par 20 ou moins, LeapFrog nous donne de meilleurs résultats que RK2. Sinon RK2 est plus précis que LeapFrog.

De même, nous avons essayé de retrouver l'ordre avec la vitesse. Voici les résultats obtenus avec les mêmes paramètres que pour la position :




| Méthode | Erreur (en %) | Ordre calculé | Ordre théorique |
|---|---|---|---|
| Euler | 3 | 1.101 | 1 |
| RK2 | 0.19 | 2.058 | 2 |
| RK3 | 0.03 | 2.985 | 3 |
| LeapFrog | 1 | 0.760 | 2 |

TABLE 4.2 – Résultats obtenus

Comme prévu, l'ordre pour les méthodes Euler, RK2 et RK3 correspondent à ceux recherchés et l'ordre pour LeapFrog n'est pas celui recherché. Cela nous pose problème car il est très important pour nous de vérifier les résultats obtenus pour la vitesse car l'énergie cinétique en dépend. Or il est primordial en physique de vérifier si les énergies se conservent ou se dissipent.

## 4.6 Préconisations

| Méthode | Coût | Ordre | Précision |
|---|---|---|---|
| Euler | +++ | + | + |
| RK2 | ++ | ++ | ++ |
| RK3 | + | +++ | +++ |
| LeapFrog | +++ | + | ++ |

TABLE 4.3 – Récapitulatif des coûts, ordres et précisions des méthodes

Nous recherchons la précision à moindre coût. Euler est la moins précise des quatre, on la met de côté. RK3 nous offre une grande précision mais elle nous coûte beaucoup trop cher en terme de calculs. On la met de côté aussi. Il nous reste RK2 et LeapFrog. Tout dépend ici de la division du temps de contact. Nous avons vu précédemment que si le temps de contact était divisé par 15 ou moins, LeapFrog est plus précis que RK2 sinon RK2 est plus précis que LeapFrog.

Si l'utilisateur veut diviser son temps de contact plus de 15 fois, il devrait utiliser la méthode RK2 pour avoir des résultats précis à moindre coût. Sinon, s'il veut diviser son temps de contact 15 fois ou moins, il devrait utiliser LeapFrog pour les mêmes raisons.

# Conclusions et perspectives

Nous avions choisi ce projet car nous étions tous intéressés par le domaine de la modélisation et aussi par l'aspect bibliographique de ce projet. Il nous a permis de développer diverses compétences, notamment dans l'organisation et la répartition du travail au sein d'un groupe de 6 personnes. Nous avons décidé de travailler en sous-groupes, cette organisation a permis à chaque binôme de se concentrer sur les tâches du projet qui l'intéressaient davantage et donc d'approfondir ses connaissances dans des domaines précis.

De plus, nous avons appris à synthétiser et analyser des informations parfois complexes (par exemple sur des thèses précises) afin de les mettre à profit pour notre projet. C'est une compétence essentielle pour notre avenir professionnel. En effet, un ingénieur doit avoir un esprit de synthèse lorsqu'il est confronté à des publications scientifiques. Concernant la modélisation, nous avons pu voir qu'il est important de définir précisément les hypothèses nécessaires à la résolution de nos équations.

L'intérêt d'un tel projet est qu'il dispose encore de nombreuses pistes d'évolution. Deux optimisations intéressantes seraient de modéliser le contact tangentiel entre deux particules ou même de développer le contact entre deux particules sphériques.

# Bibliographie


[1] https ://www.princeton.edu/cbe/people/faculty/sundaresan/group/publications/pdf/110.pdf

[2] http ://www.fresnel.fr/perso/stout/Anal_numer/Cours4.pdf

[3] http ://femto-physique.fr/physique_statistique/pdf/diffusion_moleculaire.pdf

[4] http ://oatao.univ-toulouse.fr/12239/1/Bernard.pdf

[5] http ://www.f-legrand.fr/scidoc/docmml/sciphys/dynmol/spheresmolles2d/spheresmolles2d.html

[6] http ://phyexpdoc.script.univ-paris-diderot.fr/projets_/sites_01_02_2/rebond/Theorie.htm

[7] http ://www.lmm.jussieu.fr/~lagree/COURS/MECAVENIR/cours1_tcin.pdf

[8] http ://www-evasion.imag.fr/Membres/Francois.Faure/ascollisions/rapport_final_ascollis.pdf

[9] http ://young.physics.ucsc.edu/115/leapfrog.pdf

[10] http ://matplotlib.org/examples/index.html

[11] http ://math.univ-lille1.fr/~debievre/M45website/chapter_equadiff_toprint.pdf

[12] http ://taupe.free.fr/Tz/equaDiff/2_ResolutionEDO.pdf

[13] http ://www.ulb.ac.be/di/map/gbonte/calcul/math31_7_1_dia.pdf

[14] http ://ethesis.inp-toulouse.fr/archive/00002842/01/Bernard.pdf

[15] http ://calcul.math.cnrs.fr/Documents/Ecoles/2008/m2/cours-ODE.pdf

[16] http ://www.les-math ematiques.net/phorum/read.php ?10,295995,295995

[17] http ://math.univ-lyon1.fr/~caldero/Cochoy-TIPE.pdf

[18] https ://www.math.auckland.ac.nz/~butcher/CONFERENCES/JAPAN/KYUSHU/kyushu-slides.pdf

# Annexe

## Programme Python

### Programme principal

```
import numpy as np
from math import *
from partie_utilisateur import *
from methode_resolution import *
from partie_graphique import *

# Choix de valeurs par defaut ou saisies par l'utilisateur :

choixVI = demanderChoixVI ()
print ('\n')
print ('Vous etes en 1 dimension')
print ('\n')

if choixVI == 1 :
  print ('Toutes les valeurs doivent etre donnees dans ')
  print ('les unites du systeme international')
  print ('\n')
  # Permet de taper les conditions initiales
  x0 = demanderPositionInitiale()
  v0 = demanderVitesseInitiale()
  rayon = demanderRayon()
  mv = demanderMateriau()
  coeffRest = demanderCoefficientRestitution()
  pourcentageOverlap = demanderPourcentageOverlap()
  masse = 4/3*pi*rayon**3*mv
  print ('\n')
else :
  print ('Toutes les valeurs sont dans les unites ')
  print ('du systeme international')
  print ('\n')
```

```
  x0 = 1
  v0 = 0.1
  rayon = 0.005
  masse = 4/3*pi*rayon**3*8000
  coeffRest = 0.9
  pourcentageOverlap = 0.01
  print ('Valeurs par defaut : ')
  print ('x0 = 1')
  print ('v0 = 0.1')
  print ('rayon = 0.005')
  print ('coeffRest = 0.9')
  print ('pourcentageOverlap = 0.01')
  print ('\n')

# Initialise les variables communes a toutes les methodes
t = (x0-rayon)/v0
xN = 0
k = 0
overlap = rayon
while overlap/rayon > pourcentageOverlap :
  k += 1000
  w2 = k/masse
  f = (-2*sqrt(masse*k)*log(coeffRest))/ sqrt (pi**2 + (log(coeffRest))**2)
  a = f/(2*masse)
  s = sqrt(w2 - a*a)
  r = -a
  t_overlap = atan(-s/r)/s
  overlap = v0*exp(r*t_overlap)*sin(s*t_overlap)/s
print ('Votre particule fait ',masse,' kg.')
t_contact = t + pi/s
deltat = pi/(50*s)

# Cree et initialise avec des zeros un tableau de resultats
# (50 000 000 lignes et 5 colonnes)
# Les valeurs sont de type 'float'
tabResultats = np.zeros ((50000000, 5), dtype = 'f')

# On remplit la premiere ligne du tableau
tabResultats [0][0] = t
tabResultats [0][1] = 0
tabResultats [0][2] = 0
tabResultats [0][3] = 0
tabResultats [0][4] = 0

# Compte le nombre de methodes utilisees
compteur = 1
```

```
# Cree le tableau recapitulatif des methodes utilisees
tabMethodes = []

# Demande pour la premiere fois la methode souhaitee
methode = demanderMethodeResolution()

# Enregistre la premiere methode
tabMethodes.append(methode)

# Initialise le numero de valeur de ligne
i = 0

# Initialise les variables en fonction de la methode utilisee
if (methode == 'RK2') or (methode == 'RK3') or (methode == 'Euler') :
  vN = v0
  xN1 = 0
  vN1 = 0
elif (methode == 'LF'):
  sommedesforcesN = (-k*xN-f*v0)/masse
  vN1_2 = v0 + 1/2 * deltat * sommedesforcesN
  xN1 = 0

while True :
  # Application de la methode choisie
  if (methode == 'RK2') :
    xN,vN = RK2 (deltat,masse,k,f,t,xN,vN,xN1,vN1)
  elif (methode == 'RK3') :
    xN,vN = RK3 (deltat,masse,k,f,t,xN,vN,xN1,vN1)
  elif (methode == 'LF') :
    xN,vN1_2 = LeapFrog (deltat,masse,k,f,t,xN,vN1_2)
  elif (methode == 'Euler') :
    xN,vN = Euler (deltat, masse, k, f, t, xN, vN)

  # Passe a l'instant suivant
  t += deltat
  # Arrondi t a 10 decimales
  t = round(t,10)

  i += 1
  tabResultats [i][0] = t
  tabResultats [i][compteur] = xN

  if t > t_contact + deltat : break

autreMethodeOuPas = 'oui'
```

```
nouvelleMethode = methode

while (autreMethodeOuPas == 'oui') and (compteur < 4) :
  print ('\n')
  autreMethodeOuPas = demanderNouvelleMethodeResolution()

  if (autreMethodeOuPas == 'oui') :
    compteur += 1
    print ('\n')
    nouvelleMethode = demanderMethodeResolution()

    while tabMethodes.count(nouvelleMethode) == 1 :
      print ('\n')
      print ('Vous avez redemande une methode deja utilisee')
      print ('Recommencez')
      nouvelleMethode = demanderMethodeResolution()
    tabMethodes.append(nouvelleMethode)

    # Reinitialise les variables communes a toutes les methodes
    t = (x0-rayon)/v0
    xN = 0

    # Reinitialise les variables en fonction de la methode utilisee
    if ((nouvelleMethode == 'RK2') or (nouvelleMethode == 'RK3')
    or (nouvelleMethode == 'Euler')) :
      vN = v0
      xN1 = 0
      vN1 = 0
    elif (nouvelleMethode == 'LF'):
      sommedesforcesN = (-k*xN-f*v0)/masse
      vN1_2 = v0 + 1/2 * deltat * sommedesforcesN
      xN1 = 0

    i = 0

    while True :
      if (nouvelleMethode == 'RK2') :
        xN,vN = RK2 (deltat,masse,k,f,t,xN,vN,xN1,vN1)
      elif (nouvelleMethode == 'RK3') :
        xN,vN = RK3 (deltat,masse,k,f,t,xN,vN,xN1,vN1)
      elif (nouvelleMethode == 'LF') :
        xN,vN1_2 = LeapFrog (deltat,masse,k,f,t,xN,vN1_2)
      elif (methode == 'Euler') :
        xN,vN = Euler (deltat, masse, k, f, t, xN, vN)

      # Passe a l'instant suivant
```

```
        t += deltat
        # Arrondi t a 10 decimales
        t = round(t,10)

        i += 1
        tabResultats [i][compteur] = xN

        if t > t_contact + deltat : break


afficherGraphique (deltat,t,compteur,tabMethodes,tabResultats,i)
```

### Unité « methode_resolution.py »

```
def RK2 (deltat,masse,k,f,t,xN,vN,xN1,vN1) :

  # Initialise les variables propres à la méthode de résolution
  sommeForcesN = 0
  sommeForcesN1_2 = 0
  xN1_2 = 0
  vN1_2 = 0

  # Calcul de la somme des forces à l'instant N
  sommeForcesN = ((-k)*xN-(f*vN))/masse

  # Calcul de la vitesse à l'instant N+ 1/2
  vN1_2 = vN + (deltat/2)*sommeForcesN
  # Calcul de l'overlap à l'instant N+ 1/2
  xN1_2 = xN + (deltat/2)*vN
  # Calcul de la somme des forces à l'instant N +1/2
  sommeForcesN1_2 = (-k*xN1_2-f*vN1_2)/masse

  # Calcul de la vitesse à l'instant N+1
  vN1 = vN + deltat*sommeForcesN1_2
  # Calcul de l'overlap à l'instant N+1
  xN1 = xN + deltat*vN1

  xN = xN1
  vN = vN1

  return [xN,vN]

def RK3 (deltat,masse,k,f,t,xN,vN,xN1,vN1) :
  # Initialise les variables propres à la méthode de résolution
  sommeForcesN = 0
  sommeForcesN1_3 = 0
```

```
    sommeForcesN1_2 = 0
    xN1_3 = 0
    xN1_2 = 0
    vN1_3 = 0
    vN1_2 = 0

    # Calcul de la somme des forces à l'instant N
    sommeForcesN = ((-k)*xN-(f*vN))/masse

    # Calcul de la vitesse à l'instant N+ 1/3
    vN1_3 = vN + (deltat/3)*sommeForcesN
    # Calcul de l'overlap à l'instant N+ 1/3
    xN1_3 = xN + (deltat/3)*vN
    # Calcul de la somme des forces à l'instant N +1/3
    sommeForcesN1_3 = (-k*xN1_3-f*vN1_3)/masse

    # Calcul de la vitesse à l'instant N+ 1/2
    vN1_2 = vN + (deltat/2)*sommeForcesN1_3
    # Calcul de l'overlap à l'instant N+ 1/2
    xN1_2 = xN + (deltat/2)*vN1_3
    # Calcul de la somme des forces à l'instant N +1/2
    sommeForcesN1_2 = (-k*xN1_2-f*vN1_2)/masse

    # Calcul de la vitesse à l'instant N+1
    vN1 = vN + deltat*sommeForcesN1_2
    # Calcul de l'overlap à l'instant N+1
    xN1 = xN + deltat*vN1_2

    xN = xN1
    vN = vN1

    return [xN,vN]

def LeapFrog (deltat,masse,k,f,t,xN,vN1_2) :
    # Initialise les variables propres à la méthode de résolution
    sommeForcesN1 = 0
    vN3_2 = 0

    # Calcul de l'overlap à l'instant N+1
    xN1 = xN + deltat*vN1_2

    # Calcul de la somme des forces à l'instant N+1
    sommeForcesN1 = ((-k)*xN1-(f*vN1_2))/masse

    # Calcul de la vitesse à l'instant N+3/2
    vN3_2 = vN1_2 + deltat*sommeForcesN1
```

```
  xN = xN1
  vN1_2 = vN3_2

  return [xN,vN1_2]

def Euler (deltat, masse, k, f, t, xN, vN):

  # Calcul de la somme des forces à l'instant N
  sommeForcesN = ((-k)*xN-(f*vN))/masse

  # Calcul de la vitesse à l'instant N+1
  vN1 = vN + deltat*sommeForcesN

  # Calcul de l'overlap à l'instant N+1
  xN1 = xN + deltat*vN1

  xN = xN1
  vN = vN1

  return [xN,vN]
```

### Unité « partie_graphique.py »

```
# création du graphique
import matplotlib.pyplot as plt

def afficherGraphique (deltat,t,compteur,tabMethodes,tabResultats,i) :

  if compteur == 1 :
    # Partie "chargement des données pour le graphique"

    # Crée 2 listes : une pour les valeurs prises par le temps
    # et une autre pour les valeurs prises par la position
    temps = []
    positionMethode1 = []

    # Remplit les 2 listes grâce au tableau des résultats
    for j in range (i):
      temps.append(tabResultats [j][0])
      positionMethode1.append(tabResultats [j][1])

    # Détermine quelle méthode est utilisée
    legende1 = tabMethodes[0]

    # Partie "création du graphique"
```

```
    # Donne un titre
    plt.title ('Evolution de l''overlap en fonction du temps')

    # Donne les données en x, les données en y, la couleur du trait
    # et le type de trait, la légende du tracé
    plt.plot (temps, positionMethode1, 'r--', label = legende1)

    # Donne la légende de l'axe x
    plt.xlabel ('Temps (en s)')

    # Donne la légende de l'axe y
    plt.ylabel ('Position (en m)')

    # Affiche la légende
    plt.legend (loc = 'best')

    # Affiche le graphe
    plt.show ()

  elif compteur == 2 :
    # Partie "chargement des données pour le graphique"

    # Crée 3 listes : une pour les valeurs prises par le temps
    # et deux autres pour les valeurs prises par la position
    temps = []
    positionMethode1 = []
    positionMethode2 = []

    # Remplit les 3 listes grâce au tableau des résultats
    for j in range (i):
      temps.append(tabResultats [j][0])
      positionMethode1.append(tabResultats [j][1])
      positionMethode2.append(tabResultats [j][2])

    # Détermine quelles méthodes sont utilisées
    legende1 = tabMethodes[0]
    legende2 = tabMethodes[1]

    # Partie "création du graphique"

    # Donne un titre
    plt.title ('Evolution de l''overlap en fonction du temps')

    # Donne les données en x, les données en y, la couleur du trait
    # et le type de trait, la légende du tracé
    plt.plot (temps, positionMethode1, 'r--', label = legende1)
```

```
    plt.plot (temps, positionMethode2, 'b-', label = legende2)

    # Donne la légende de l'axe x
    plt.xlabel ('Temps (en s)')

    # Donne la légende de l'axe y
    plt.ylabel ('Position (en m)')

    # Affiche la légende
    plt.legend (loc = 'best')

    # Affiche le graphe
    plt.show ()

  elif compteur == 3 :
    # Partie "chargement des données pour le graphique"

    # Crée 4 listes : une pour les valeurs prises par le temps
    # et les autres pour les valeurs prises par la position
    temps = []
    positionMethode1 = []
    positionMethode2 = []
    positionMethode3 = []

    # Remplit les 4 listes grâce au tableau des résultats
    for j in range (i):
      temps.append(tabResultats [j][0])
      positionMethode1.append(tabResultats [j][1])
      positionMethode2.append(tabResultats [j][2])
      positionMethode3.append(tabResultats [j][3])

    # Détermine quelles méthodes sont utilisées
    legende1 = tabMethodes[0]
    legende2 = tabMethodes[1]
    legende3 = tabMethodes[2]

    # Partie "création du graphique"

    # Donne un titre
    plt.title ('Evolution de l''overlap en fonction du temps')

    # Donne les données en x, les données en y, la couleur du trait
    # et le type de trait, la légende du tracé
    plt.plot (temps, positionMethode1, 'r--', label = legende1)
    plt.plot (temps, positionMethode2, 'b-', label = legende2)
    plt.plot (temps, positionMethode3, 'g-', label = legende3)
```

```
    # Donne la légende de l'axe x
    plt.xlabel ('Temps (en s)')

    # Donne la légende de l'axe y
    plt.ylabel ('Position (en m)')

    # Affiche la légende
    plt.legend (loc = 'best')

    # Affiche le graphe
    plt.show ()

  elif compteur == 4 :
    # Partie "chargement des données pour le graphique"

    # Crée 5 listes : une pour les valeurs prises par le temps
    # et les autres pour les valeurs prises par la position
    temps = []
    positionMethode1 = []
    positionMethode2 = []
    positionMethode3 = []
    positionMethode4 = []

    # Remplit les 5 listes grâce au tableau des résultats
    for j in range (i):
      temps.append(tabResultats [j][0])
      positionMethode1.append(tabResultats [j][1])
      positionMethode2.append(tabResultats [j][2])
      positionMethode3.append(tabResultats [j][3])
      positionMethode4.append(tabResultats [j][4])

    # Détermine quelles méthodes sont utilisées
    legende1 = tabMethodes[0]
    legende2 = tabMethodes[1]
    legende3 = tabMethodes[2]
    legende4 = tabMethodes[3]

    # Partie "création du graphique"

    # Donne un titre
    plt.title ('Evolution de l''overlap en fonction du temps')

    # Donne les données en x, les données en y, la couleur du trait
    # et le type de trait, la légende du tracé
    plt.plot (temps, positionMethode1, 'r--', label = legende1)
```

```
    plt.plot (temps, positionMethode2, 'b-', label = legende2)
    plt.plot (temps, positionMethode3, 'g-', label = legende3)
    plt.plot (temps, positionMethode4, 'c-', label = legende4)

    # Donne la légende de l'axe x
    plt.xlabel ('Temps (en s)')

    # Donne la légende de l'axe y
    plt.ylabel ('Position (en m)')

    # Affiche la légende
    plt.legend (loc = 'best')

    # Affiche le graphe
    plt.show ()
```

### Unité « partie_utilisateur.py »

```
def demanderChoixVI () :
  print ('Voulez-vous choisir vos propres valeurs initiales ')
  print ('ou voulez-vous des valeurs par défaut ?')
  print ('Tapez 1 pour choisir vos valeurs initiales')
  print ('Tapez 2 pour les valeurs par défaut')
  choixVI = int(input ('choix : '))
  return choixVI

def demanderPositionInitiale() :
  print ('Entrez la position initiale x0')
  x0 = float(input ('x0 = '))
  return x0

def demanderVitesseInitiale() :
  print ('Entrez la vitesse initiale')
  v0 = float(input ('v0 = '))
  return v0

def demanderRayon() :
  print ('Entrez le rayon de la particule')
  rayon = float(input ('rayon = '))
  return rayon

def demanderMateriau() :
  print ('Choisissez votre matériau parmi :')
  print ('Verre')
  print ('Acier')
  materiau = input ('materiau = ')
```

```
  while (materiau != 'Verre') and (materiau != 'Acier') :
    print ('\n')
    print ('Vous n''avez pas donné une réponse possible')
    print ('Choisissez votre matériau parmi :')
    print ('Verre')
    print ('Acier')
    materiau = input ('materiau = ')
  if materiau == 'Verre' :
    mv = 2500
  if materiau == 'Acier' :
    mv = 8000
  return mv

def demanderCoefficientRestitution() :
  print ('Entrez le coefficient de restitution attendu ')
  print ('(il doit être compris entre 0 et 1)')
  coeffRest = float(input ('Coefficient de restitution = '))
  return coeffRest

def demanderPourcentageOverlap() :
  print ('Entrez le pourcentage du recouvrement de la particule')
  pourcentageOverlap = float(input ('Pourcentage de recouvrement = '))
  return pourcentageOverlap

def demanderMethodeResolution() :
  print ('Choisissez votre méthode de résolution parmi :')
  print ('- Runge Kutta 2 (tapez RK2)')
  print ('- Runge Kutta 3 (tapez RK3)')
  print ('- LeapFrog (tapez LF)')
  print ('- Euler (tapez Euler)')
  methode = input ('methode = ')
  while (methode != 'RK2') and (methode != 'RK3')
  and (methode != 'LF') and (methode != 'Euler'):
    print ('\n')
    print ('Vous n''avez pas donné une réponse possible')
    print ('Choisissez votre méthode de résolution parmi :')
    print ('- Runge Kutta 2 (tapez RK2)')
    print ('- Runge Kutta 3 (tapez RK3)')
    print ('- LeapFrog (tapez LF)')
    print ('- Euler (tapez Euler)')
    methode = input ('methode = ')
  return methode

def demanderNouvelleMethodeResolution() :
  print ('Voulez-vous tester une autre méthode de résolution ?')
  print ('Répondez par oui ou non')
```

```
autreMethodeOuPas = input ('Votre réponse : ')
while (autreMethodeOuPas != 'oui') and (autreMethodeOuPas != 'non') :
  print ('\n')
  print ('Vous n''avez pas donné une réponse possible')
  print ('Voulez-vous tester une autre méthode de résolution ?')
  print ('Répondez par oui ou non')
  autreMethodeOuPas = input ('Votre réponse : ')
return autreMethodeOuPas
```