Émergence de structures gravitationnelles dans un univers bidimensionnel simplifié

Emile Kronenberg

2026

1 Introduction

L'objectif de ce projet est d'étudier, par simulation numérique, la formation spontanée de structures gravitationnelles à partir d'un état initial quasi homogène. Il s'agit d'un modèle jouet inspiré de la cosmologie, reposant sur la mécanique newtonienne.

Ce projet est fait par un étudiant en prépa, et ne se veut en aucun cas parfaitement rigoureux. Vous y trouverez peut-être des erreurs, ou des mauvais jugements/interprétations. Ce projet est uniquement à but d'amusement.

1.1 Univers Modélisé

Nous allons en effet modéliser un ensemble de N nuages - que l'on nommera « grumeau » par la suite - de noyaux d'hydrogènes.

Nous essaierons de suivre le modèle ΛCDM (Lambda Cold Dark Matter). C'est un modèle cosmologique simple ayant pour but de représenter l'univers après le Big Bang.

Ce modèle se fonde sur trois hypothèses :

- Le principe cosmologique, nous disant que l'Univers est homogène et isotrope.
- Le principe d'universalité, en vertu duquel la gravitation est décrite par la relativité générale (on ne cherche pas à rentrer dans les détails).
- Le contenu de l'Univers

1.2 Objectif du projet

L'objectif de ce projet et de cette modélisation est de mieux comprendre la fragmentation et l'effondrement d'un nuage d'hydrogène, comme il en existe beaucoup dans une galaxie spirale.
L'objectif de ce rapport n'est pas de faire un rapport propre, mais de suivre chronologiquement le projet.

1.3 Sujet initial du Projet

Cette modélisation aura pour but d'étudier, par simulation numérique, la formation spontanée de structures gravitationnelles à partir d'un état initial quasi homogène. Il s'agit d'un modèle jouet inspiré de la cosmologie, reposant exclusivement sur la mécanique newtonienne. On considère un système de N particules ponctuelles de masse identique m , évoluant dans un domaine carré bidimensionnel. Les positions initiales sont distribuées de manière presque uniforme, avec de petites fluctuations aléatoires contrôlées. Les vitesses initiales peuvent être nulles ou faiblement perturbées. Les particules interagissent via la gravitation newtonienne. La force exercée sur la particule i par la particule j est donnée par :

$$\vec{F}_{ij} = -\frac{Gm^2\vec{r}_{ij}}{\left(|\vec{r}_{ij}|^2 + \varepsilon^2\right)^{3/2}} \tag{1}$$

où ϵ est un paramètre d'adoucissement numérique destiné à éviter les divergences lorsque deux particules se rapprochent excessivement. La force totale sur chaque particule est obtenue par superposition.

2 Modélisation de N nuages d'hydrogène.

2.1 Introduction

On modélisera d'abord N immenses nuages d'hydrogène dans un univers bidimensionnel et on y appliquera la mécanique de Newton. Chaque nuage sera modélisé singulièrement, et on fera l'hypothèse que chaque grumeau est indivisible, neutre, homogène et isotrope.

Les objectifs de cette partie sont de :

- Créer un premier modèle pour regarder et analyser les interactions entre les différents nuages d'hydrogène.
- Coder un premier programme en python qui valide notre modèle newtonien.
- Une fois le code exécuté et validé, on tentera de le rendre moins complexe, pour faciliter les calculs et donc améliorer la vitesse de calcul.

On commencera par coder cela car la mécanique newtonienne est simple à coder en python, et on se basera par la suite sur ce code pour implémenter des phénomènes physiques plus complexes, dans le cas de l'étude de l'effondrement et de la fragmentation.

2.2 Méthode de Verlet cf. WikiPedia

L'intégration de Verlet est un schéma d'intégration qui permet de calculer la trajectoire de particules en simulation de dynamique moléculaire.

Cette méthode a de bon qu'elle est réversible et beaucoup plus simple à mettre en place que la méthode d'Euler, par exemple.

Source : Wikipedia — Algorithme de Verlet

L'algorithme de Verlet réduit le taux d'erreur introduit par l'intégration en calculant la position au pas de temps suivant à partir des positions courante et précédente, sans faire appel à la vitesse, en utilisant deux développements de Taylor de la position $\vec{x}(t)$ à deux instants distincts.

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{\vec{a}(t)\Delta t^2}{2} + \frac{\vec{b}(t)\Delta t^3}{6} + O(\Delta t^4) \tag{2}$$

$$\vec{x}(t-\Delta t) = \vec{x}(t) - \vec{v}(t)\Delta t + \frac{\vec{a}(t)\Delta t^2}{2} - \frac{\vec{b}(t)\Delta t^3}{6} + O(\Delta t^4) \tag{3}$$

avec :

- $\vec{x}$ la position ;
- $\vec{v}$ la vitesse ;
- $\vec{a}$ l'accélération ;
- $\vec{b}$ l'à-coup (unité S.I., dérivée troisième de la position par rapport au temps t).

En additionnant ces deux équations on obtient :

$$\vec{x}(t+\Delta t) = 2\vec{x}(t) - \vec{x}(t-\Delta t) + \vec{a}(t)\Delta t^2 + O(\Delta t^4) \tag{4}$$

Cette opération offre l'avantage d'avoir les dérivées de premier et troisième ordre qui s'annulent, rendant l'intégration de Verlet d'un ordre plus précis qu'un simple développement de Taylor.

Dans un premier temps, on utilisera l'équation (2) pour calculer successivement nos positions pour chacun de nos nuages.

2.3 Calculs de Force et complexité

On doit donc maintenant implémenter la mécanique newtonienne. On calcule donc la force gravitationnelle exercée sur chaque nuage pour ensuite réaliser un PFD, et intégrer notre PFD pour obtenir la position $u(t+\Delta t)$.

Le premier programme, (pour calculer les forces) que nous avons réalisé est le suivant :

```
x_min, x_max = -10, 10
y_min, y_max = -10, 10
nx = int(np.floor(np.sqrt(N)))
ny = int(np.ceil(N / nx))
x = np.linspace(x_min, x_max, nx)
y = np.linspace(y_min, y_max, ny)
X, Y = np.meshgrid(x, y)

points_normaux = np.column_stack((X.ravel(), Y.ravel()))
points_normaux = points_normaux[:N]
tableau_d = d * ((rng.random((points_normaux.shape[0], points_normaux.shape[1])) - 1) * 2)

points = points_normaux + tableau_d
pnumb = points.shape[0]

L   = []
pos = np.array(points)
vel = np.zeros((pnumb, 2))
t   = 0

while t < duree:
    i    = pos[:, np.newaxis, :]
    j    = pos[np.newaxis, :, :]
```

```
    r_ij = j - i
    n_r_ij  = np.linalg.norm(r_ij, axis=2)
    n_r_ij += np.eye(N) * 1e-12
    F   = -G * m**2 * r_ij / ((n_r_ij**2 + epsilon**2)**(3/2))[..., np.newaxis]

    acc = -F.sum(axis=1) / m
    pos, vel = verlet.velocity_verlet(pos, vel, acc, dt)
    L.append(pos.copy())

    t += dt
```

Le programme deviendra très lent quand on commencera à calculer beaucoup de points. Ici l'objectif est de pouvoir en modéliser des centaines, voire quelques milliers.

Pour cela, on va pouvoir trouver deux solutions :

- La première est d'utiliser la 3$^{\text{ième}}$ loi de Newton, ce qui permettra déjà de ne calculer que la partie triangulaire supérieure de notre matrice des forces et réduira notre temps d'exécution par 2.
- La seconde solution est de paralléliser les calculs. Étant sur amd et sur windows (et oui, la vie est dure...) il m'est compliqué de paralléliser sur GPU, donc j'ai fait cela sur CPU à l'aide du module Numba de Python. Ce module permet de paralléliser sur les différents cœurs de mon CPU.
  J'aurais pu essayer de faire ces calculs sur GPU, mais la partie physique de ce projet est selon moi plus intéressante.

J'ai donc mis en place ces deux solutions grâce à cette fonction :

```
@njit(parallel=True)
def calcul_forces(
    pos:     np.ndarray,
    N:       int,
    G:       float,
    m:       float,
    epsilon: float
) -> np.ndarray:
    """
    Calcul des forces gravitationnelles avec :
    - Numba : compilation + parall lisation sur tous les coeurs CPU
      -> On sort ici donc de la vectorisation permise par numpy, car cela est plus rapide
    - Newton 3 me  loi : on calcule seulement la moiti  des paires (F_ij = -F_ji)
    """
    F_on_j = np.zeros((N, 2))
    for i in prange(N):
        for j in range(i + 1, N):  # triangle sup rieur uniquement (car on utilise la 3eme loi de
            Newton)
            r     = pos[j] - pos[i]
            dist2 = r[0]**2 + r[1]**2
            f     = -G * m**2 / (dist2 + epsilon**2)**(3/2)
            fx    = f * r[0]
            fy    = f * r[1]
            F_on_j[i, 0] += fx
            F_on_j[i, 1] += fy
            F_on_j[j, 0] -= fx   # Newton : force oppos e sur j
            F_on_j[j, 1] -= fy
    return F_on_j
```

Remarque

On teste ces deux programmes avec 2000x grumeau, sur 6 unités de temps, avec un dt de 0.1 unité de temps (ce qui d'un point de vue calcul n'est pas une bonne idée, mais le résultat nous importe peu).
On utilisera la fonction `time.perftime()`.

Après lancement on obtient :

```
Le programme parallelise a pris 6.0916884999969625 secondes pour s executer.
Le programme non parallelise a pris 19.0987746000028 secondes pour s executer.
```

2.4 Mise en place du programme sur des valeurs vraisemblables

Nous allons donc maintenant tester le programme sur des valeurs qui sont celles qui pourraient exister dans la vraie vie (si on considère que des immenses nuages d'hydrogène d'environ une masse solaire sont la vraie vie ;)).

Les valeurs sur lesquelles on va lancer le programme sont les suivantes :

- $m = 1{,}0\ M_\odot = 2 \times 10^{30}$ kg : masse d'un grumeau de gaz
  Cohérent avec la masse typique d'un fragment de nuage moléculaire proto-stellaire.
- $N = 1000$: nombre de grumeaux simulés
  Un globule de Bok typique contient 10^2 à 10^4 fragments de gaz à cette échelle de masse.
- $d = 0{,}05$ pc $= 1{,}5 \times 10^{14}$ m $\approx 0{,}16$ année-lumière : décalage aléatoire initial
  Les fluctuations de densité dans un nuage moléculaire sont de l'ordre de quelques centièmes de parsec, cohérent avec les observations de Barnard 68.
- $G = 0{,}0045\ \mathrm{pc}^3 \cdot M_\odot^{-1} \cdot \mathrm{Myr}^{-2} = 6{,}674 \times 10^{-11}\ \mathrm{m}^3 \cdot \mathrm{kg}^{-1} \cdot \mathrm{s}^{-2}$: constante gravitationnelle
- $\varepsilon = 0{,}05$ pc $\approx 0{,}16$ année-lumière : paramètre d'adoucissement
- $\Delta t = 0{,}001$ Myr $= 1\,000$ ans : pas de temps
- $T = 1{,}0$ Myr $= 10^6$ années ≈ 1 temps de Jeans : durée totale de la simulation
  Le temps de Jeans pour un nuage de densité $\rho \sim 10^{-20}$ kg/m^3 est de l'ordre du million d'années, ce qui correspond exactement à notre durée de simulation.

On peut citer des exemples concrets dans l'univers où on peut retrouver les objets que l'on va modéliser.

Source : Wikipédia — Globule de Bok

Un globule de Bok est un amas sombre de poussières et de gaz du milieu interstellaire au sein duquel peut commencer la naissance des étoiles. Ils ont typiquement une masse d'environ 10 à 50 masses solaires contenue dans un volume d'environ une année-lumière. Ils contiennent de l'hydrogène moléculaire (H_2), des oxydes de carbone, de l'hélium et environ 1% de poussières de silicates.

Source : Wikipédia — Barnard 68

Barnard 68 est un nuage moléculaire situé à environ 500 années-lumière de la Terre dans la constellation d'Ophiuchus. D'une masse environ deux fois celle du Soleil, le nuage fait environ une demi-année-lumière d'envergure pour une température moyenne d'environ 16 kelvins. On estime que le nuage pourrait s'effondrer gravitationnellement au cours des 200 000 prochaines années, pour éventuellement former une étoile.

Une fois ces valeurs passées dans mon programme, on obtient la simulation suivante :

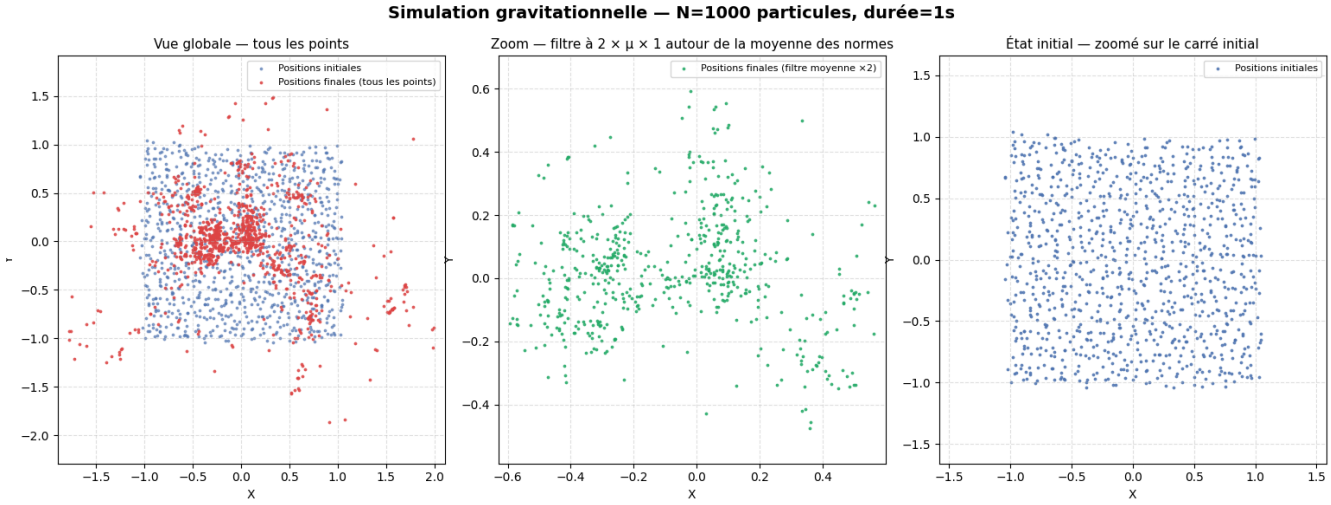


Figure 1: Simulation gravitationnelle de $N = 1000$ grumeaux de gaz sur une durée de 1 Myr. *Gauche* : vue globale montrant le début de la convergence vers le centre. *Centre* : zoom sur les 90% des points les plus proches de l'origine, révélant la formation d'une sur-densité centrale. *Droite* : état initial quasi-uniforme sur $[-1, 1]^2$ pc avec fluctuations $d = 0{,}05$ pc.

La valeur de ε

En effet, la valeur de ε, le paramètre d'adoucissement gravitationnel, n'est pas une simple approximation à prendre à la légère. Ce critère intervient dans l'équation (1) et permet d'empêcher une divergence de la force exercée sur une particule.

Le problème vient de la discrétisation du temps : si deux particules se retrouvent trop proches, la force gravitationnelle diverge et génère des accélérations immenses qui ne sont pas physiquement réalistes.

Le choix optimal de ε a été étudié par **Power et al. (2003)** [1], qui proposent le critère suivant : l'accélération stochastique maximale causée par l'approche d'une seule particule,

$$a_{\max} = \frac{Gm}{\varepsilon^2} \tag{5}$$

doit rester inférieure à l'accélération gravitationnelle moyenne du système entier :

$$a_{\min} \approx \frac{GM_{\text{tot}}}{R^2} \tag{6}$$

Ce qui impose la condition :

$$\varepsilon > \sqrt{\frac{m}{M_{\text{tot}}}} \cdot R = \frac{R}{\sqrt{N}} \tag{7}$$

Dans notre cas, avec $N = 1000$ grumeaux sur un domaine de rayon $R = 1$ pc, on obtient :

$$\varepsilon > \frac{1}{\sqrt{1000}} \approx 0{,}032 \text{ pc} \tag{8}$$

Notre choix de $\varepsilon = 0{,}05$ pc satisfait bien ce critère, garantissant qu'aucune particule individuelle ne peut dominer l'accélération collective du système. De plus, ε étant borné, la force gravitationnelle l'est également :

$$F \leq \frac{Gm^2}{\varepsilon^3} \tag{9}$$

[1] `https://arxiv.org/abs/astro-ph/0201544`

ce qui assure mathématiquement l'absence de divergence dans la simulation.

D'après une étude plus récente de **Zhang et al. (2019)** [2], le schéma de P03 est certes valide mais trop conservateur. Les auteurs montrent qu'un paramètre d'adoucissement deux fois plus petit permet d'atteindre une meilleure résolution spatiale tout en conservant des résultats numériquement convergents. Ils proposent le schéma révisé :

$$\varepsilon_{\text{opt}} \approx \frac{\varepsilon_{\text{opt, P03}}}{2} = \frac{2\,r_{200}}{\sqrt{N_{200}}} \tag{10}$$

On adoptera donc dans notre simulation :

$$\varepsilon = \frac{\varepsilon_{\text{P03}}}{2} = \frac{1}{2} \cdot \frac{R}{\sqrt{N}} = \frac{1}{2\sqrt{1000}} \approx 0{,}016 \text{ pc} \tag{11}$$

Remarque

Dans le cadre de notre simulation, ce raffinement du choix de ε n'aura pas d'impact significatif sur les résultats. En effet, notre objectif n'est pas d'atteindre une précision maximale, mais d'observer qualitativement la formation de structures gravitationnelles. On conservera donc $\varepsilon = 0{,}05$ pc pour des raisons de stabilité numérique.

2.5 Conclusion

Nous sommes donc parvenus à modéliser en python le système que nous voulions, en fonction de nombreux paramètres.

Nous n'étudierons pas l'influence de ces paramètres sur les résultats obtenus, car étudier les interactions macroscopiques entre plusieurs grumeaux n'est pas l'objectif de ce projet.

[2] `https://arxiv.org/abs/1810.07055`

3 Modélisation de l'Effondrement et de la Fragmentation

3.1 Introduction

Nous citerons les définitions suivantes.

Source : Wikipédia — Effondrement gravitationnel

En astronomie, un effondrement gravitationnel est la contraction d'un corps massif sous l'effet de sa propre attraction gravitationnelle. Il se produit lorsque toutes les forces de pression ne peuvent plus compenser cette attraction et maintenir un équilibre : l'astre s'effondre sur lui-même. C'est par effondrement gravitationnel d'une masse de gaz que se forment les étoiles.

Source : Wikipédia — Instabilité gravitationnelle

L'instabilité de Jeans décrit le phénomène d'effondrement gravitationnel qui peut avoir lieu au sein d'un nuage de matière gazeux à partir d'un état faiblement inhomogène. Elle se produit quand l'attraction gravitationnelle causée par une surdensité d'un milieu devient supérieure aux forces de pression qui ont tendance à détendre cette surdensité. Lorsque la masse du nuage dépasse la masse de Jeans, le nuage se fragmente en blocs de plus en plus petits qui s'effondrent à leur tour indépendamment.

Le but de cette partie est maintenant de comprendre et d'illustrer à l'aide encore une fois d'un script python l'effondrement d'un nuage d'hydrogène, pour en étudier les causes et les conséquences.

Ici, les points que nous allons modéliser ne seront plus donc des grumeaux, mais l'intégralité de nos points constitueront un unique grumeau.

Pour chaque grumeau (que nous représenterons une fois de plus en 2D), nous n'étudierons qu'une sphère contenue à l'intérieur de ce même grumeau (donc pour nous un cercle). On fait cela pour garder une simplicité de la modélisation, dépendant de plus du caractère isotrope de ces grumeaux.
On supposera donc que ce système est initialement à l'équilibre et sera perturbé très légèrement.

On modélisera cette perturbation par un très léger décalage des points par rapport à leur grille de départ (de la même manière que dans 1.).

Dans cette simulation, on étudiera une version simplifiée de la théorie de Jeans [3].

Source : Astronomia — Effondrement et fragmentation d'un nuage

La théorie présentée ici n'est pas celle construite par Jeans, mais une version plus simple dont les résultats sont du même ordre de grandeur.

On essaiera de plus (s'il est possible) de confirmer la conservation de l'énergie mécanique, et de valider certaines autres lois et d'illustrer la théorie de Jeans. Si les résultats nous le permettent nous finirons par tenter de comprendre la formation d'étoiles comme le Soleil.

3.2 Théorie de Jeans simplifiée

Hypothèses :

- On considère le nuage assez grand pour que la partie non contenue dans la sphère n'ait pas d'influence gravitationnelle sur la sphère (raisonnable au vu de la symétrie de la sphère)
- Température isotherme (pas assez dense pour être réchauffé par une quelconque onde lumineuse)
- Hypothèse des gaz parfaits monoatomiques.

[3] `https://astronomia.fr/4eme_partie/details_html/masseDeJeans.php`

On va donc chercher à comprendre sous quelle(s) condition(s) un nuage atomique peut s'effondrer.

J'ai cherché comment démontrer la suite, mais soit je n'ai pas trouvé, soit ce que j'ai trouvé était hors de ma compréhension.

Source : Astronomia — Stabilité du nuage

Soit R le rayon de la sphère considérée. Un calcul assez simple montre que la pression P_0 à la surface de la sphère est de la forme :

$$P_0 = \frac{a}{R^3} - \frac{b}{R^4} \tag{12}$$

Le premier terme représente la pression du gaz (qui pousse le gaz vers l'extérieur), le second l'autogravitation du nuage (qui l'attire vers le centre). a et b sont des constantes positives dépendant de la masse de la sphère et de sa température.

La courbe $P_0(R)$ admet un maximum en R_m, illustré sur la figure suivante :

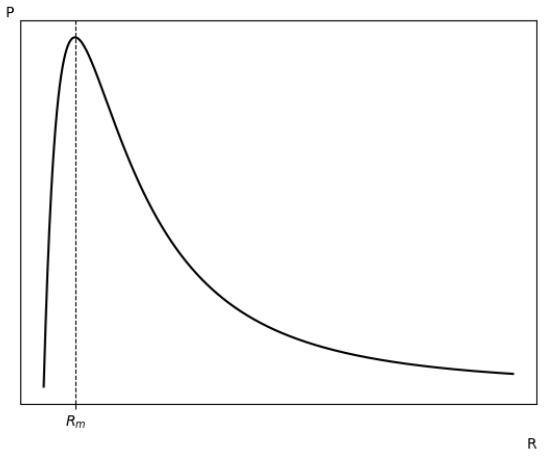


Figure 2: Pression superficielle P_0 en fonction du rayon R. Le maximum en R_m détermine la stabilité du nuage.

Deux cas se présentent :

- Si $R < R_m$: une compression entraîne une diminution de la pression, la gravité l'emporte ⇒ **le nuage s'effondre**.
- Si $R > R_m$: une compression entraîne une augmentation de la pression, qui ramène le nuage à son état initial ⇒ **la perturbation est effacée**.

On montre que le rayon critique vaut $R_m = k\sqrt{T/\rho}$, ce qui conduit à la **masse de Jeans** : l'effondrement se produit lorsque la masse du nuage dépasse :

$$M_J = k' \frac{T^{3/2}}{\rho^{1/2}} \tag{13}$$

Remarque

Pour un nuage typique (10 à 100 K, $\sim 10^3$ atomes/cm^3), on obtient $M_J \sim 10^5\, M_\odot$, soit ~ 1000 fois la masse d'une étoile. Le nuage doit donc d'abord se **fragmenter** en sous-structures de plus en plus petites avant de former des étoiles — c'est ce que notre simulation cherche à reproduire qualitativement.

3.3 Modélisation python

Nous utiliserons le modèle SPH pour modéliser les grumeaux.

En plus de la force gravitationnelle (déjà implémentée au §1.), nous devons ajouter une force de pression.

3.3.1 Modèle SPH

Le modèle SPH (Smoothed Particle Hydrodynamics) est une méthode de simulation des milieux continus, développée initialement pour des problèmes d'astrophysique [4]. Il permet de modéliser à la fois la gravité et la pression du gaz, ce qui est nécessaire pour simuler l'effondrement d'un nuage d'hydrogène.

Dans cette partie, on va donc toujours garder une description lagrangienne.

On modélisera chaque nuage par un fluide, donc chaque particule suivra l'équation d'Euler pour un fluide idéal auquel on choisira d'appliquer au système une force f.

$$\boxed{\frac{d\mathbf{v}}{dt} = -\frac{1}{\rho}\nabla P + \frac{1}{\rho}\mathbf{f}} \tag{14}$$

On part du principe que ce gaz supposé parfait suivra l'équation d'état polytropique (ce qui pose comme condition une transformation adiabatique).

On part de la loi des gaz parfaits :

$$PV = Nk_BT \tag{15}$$

On divise par V :

$$P = \frac{N}{V}k_BT = nk_BT \tag{16}$$

où $n = N/V$ est le nombre de particules par unité de volume. On pose $\rho = nm_H$ donc $n = \rho/m_H$:

$$P = \frac{\rho k_B T}{m_H} \tag{17}$$

Pour une transformation adiabatique, on admet que $TV^{\gamma-1} = \text{cste}$, soit $T \propto V^{1-\gamma} \propto \rho^{\gamma-1}$. On substitue :

$$P = \frac{k_B}{m_H}\rho \cdot \rho^{\gamma-1} = \frac{k_B}{m_H}\rho^{\gamma} \tag{18}$$

On pose $k = k_B/m_H$ et $\gamma = 1 + 1/n$:

$$\boxed{P = k\rho^{1+1/n}} \tag{19}$$

Pour le cas isotherme ($T = \text{cste}$), T ne dépend pas de ρ donc :

$$P = \frac{k_BT}{m_H}\rho = c_s^2\rho \tag{20}$$

ce qui correspond à $\gamma = 1$, soit $n \to \infty$.

Forces extérieures

La force extérieure donnée dans notre équation d'Euler sera de la forme :

$$\boxed{\mathbf{f} = -\lambda\mathbf{r} - \nu\mathbf{v}} \tag{21}$$

avec, λ le potentiel gravitationnel et ν un coefficient de frottement fluide, pour permettre au système de se mettre à l'équilibre.

[4] https://fr.wikipedia.org/wiki/Hydrodynamique_des_particules_liss%C3%A9es

Pour plus de précisions, voir Monaghan & Price (2004)[5].

On modélise le modèle SPH de cette manière :

Source : Philip Mocz — Create your own SPH simulation

SPH gets its name from the fact that each particle has its mass m distributed in space according to a smoothing kernel with a smoothing length h concentrated about the particle. For example, we can construct a Gaussian smoothing kernel with the function:

$$W(\mathbf{r}; h) = \frac{1}{h^3 \pi^{3/2}} \exp\left(\frac{-\|\mathbf{r}\|^2}{h^2}\right) \tag{22}$$

3.3.2 Calcul de la densité locale

On peut maintenant calculer la densité locale de chaque particule i à partir du noyau W. L'idée est simple : chaque particule j contribue à la densité en i proportionnellement à sa proximité [6].

$$\rho_i = \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j; h) \tag{23}$$

Remarque

On retrouve bien la cohérence dimensionnelle : $[W] = \mathrm{pc}^{-3}$ (en 3D) donc $[\rho] = M_\odot \cdot \mathrm{pc}^{-3}$, ce qui est bien une densité volumique. En 2D (notre cas), $[W] = \mathrm{pc}^{-2}$ donc $[\rho] = M_\odot \cdot \mathrm{pc}^{-2}$.

3.3.3 Force de pression SPH

Une fois la densité et la pression calculées, la force de pression sur la particule i s'écrit, dans la formulation SPH symétrique [7] :

$$\mathbf{F}_{\text{pression},i} = -m_i \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \nabla W(\mathbf{r}_i - \mathbf{r}_j; h) \tag{24}$$

Remarque

Le gradient du noyau gaussien vaut :

$$\nabla W(\mathbf{r}; h) = -\frac{2\mathbf{r}}{h^2} W(\mathbf{r}; h) \tag{25}$$

Ce qui est très pratique à coder — il suffit de multiplier W par $-2\mathbf{r}/h^2$.

3.3.4 Force totale et boucle principale

La force totale sur chaque particule est donc la somme de la force gravitationnelle (déjà implémentée au §2.) et de la force de pression :

$$\mathbf{F}_{\text{tot},i} = \mathbf{F}_{\text{grav},i} + \mathbf{F}_{\text{pression},i} \tag{26}$$

On réutilise le même intégrateur Velocity-Verlet qu'au §2., en remplaçant simplement le calcul des forces.

[5] https://arxiv.org/abs/astro-ph/0310790
[6] https://philip-mocz.medium.com/create-your-own-smoothed-particle-hydrodynamics-simulation-with-python-76e1cec505f1
[7] https://philip-mocz.medium.com/create-your-own-smoothed-particle-hydrodynamics-simulation-with-python-76e1cec505f1

Je n'ai donc pas du tout réussi à paramétrer mon code. Le problème vient du fait que pour l'exemple duquel je suis parti, la force gravitationnelle a été modélisée par une constante λ (compliquée à calculer, i.e. utilisation de la fonction gamma...). Je devais donc gérer de nombreux paramètres différents qui étaient : h la longueur de lissage du modèle SPH, ν le coefficient de frottement fluide, ϵ le paramètre d'adoucissement gravitationnel.

Je n'arrivais pas à avoir des résultats cohérents lors de la simulation (points qui divergeaient, chaos dans la simulation, masse de Jeans pas du tout respectée...)
J'ai donc adapté le modèle SPH (enfin j'ai appris comment faire cela, je n'ai rien inventé ;))

3.4 Améliorations apportées au modèle

3.4.1 Noyau cubique B-spline

Pour commencer, j'ai remplacé le noyau gaussien présenté plus haut par un noyau cubique B-spline, introduit par Monaghan & Lattanzio (1985)[8].

La raison principale est simple : le noyau gaussien a un support infini, c'est-à-dire que chaque particule contribue à la densité de *toutes* les autres, même celles très lointaines. Le noyau cubique, lui, s'annule exactement pour $q \geq 2$ (où $q = r/h$), ce qui le rend bien plus adapté au calcul numérique.

Le noyau s'écrit :

$$W(r;\, h) = \frac{\sigma_2}{h^2} f(q), \qquad q = \frac{r}{h}, \qquad \sigma_2 = \frac{10}{7\pi} \tag{27}$$

avec :

$$f(q) = \begin{cases} 1 - \dfrac{3}{2}q^2 + \dfrac{3}{4}q^3 & 0 \leq q < 1 \\ \dfrac{1}{4}(2-q)^3 & 1 \leq q < 2 \\ 0 & q \geq 2 \end{cases} \tag{28}$$

Remarque

Le gradient analytique de ce noyau s'écrit $\nabla_i W_{ij} = \frac{\sigma_2}{h^3} \frac{f'(q)}{r} (\mathbf{r}_i - \mathbf{r}_j)$, ce qui est très simple à implémenter.

[8] Monaghan, J.J. & Lattanzio, J.C. (1985), *A refined particle method for astrophysical problems*, A&A 149, 135

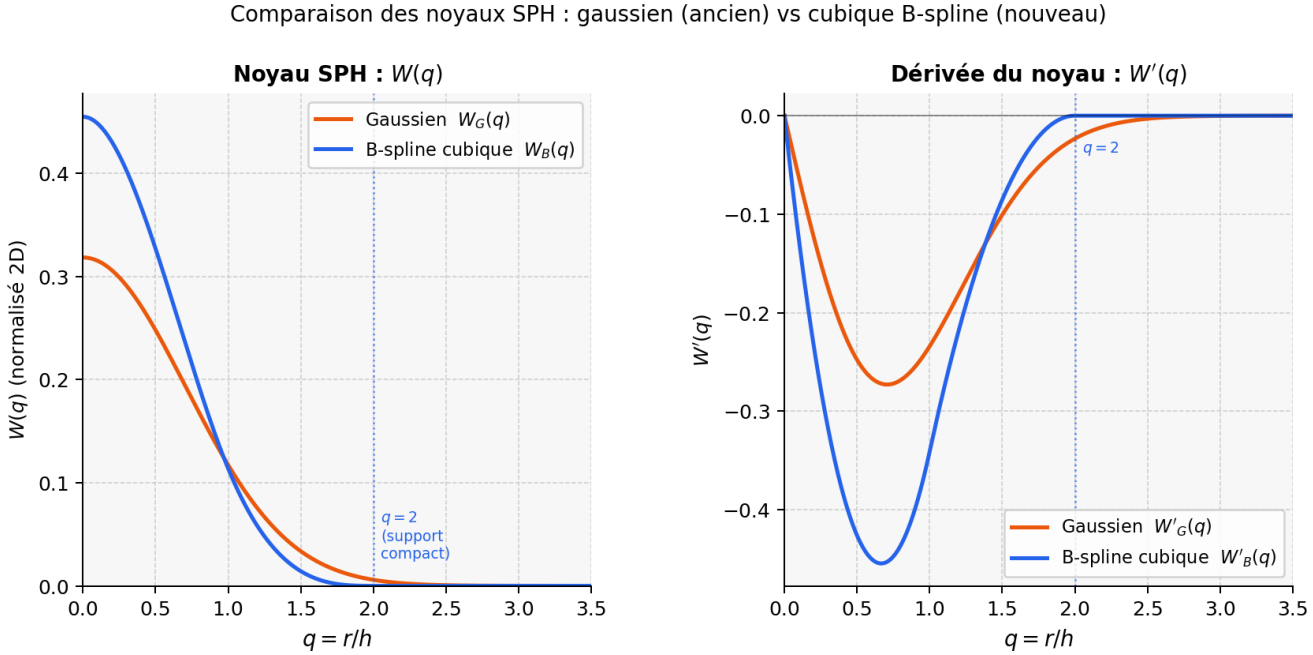


Figure 3: Comparaison du modèle SPH de la gaussienne et du noyau B-spline.

En Python avec Numba, cela donne :

```
@njit(cache=True, fastmath=True)
def noyau(q):
    if q < 1.0: return 1.0 - 1.5*q*q + 0.75*q*q*q
    if q < 2.0: t = 2.0 - q; return 0.25*t*t*t
    return 0.0

@njit(cache=True, fastmath=True)
def noyau_deriv(q):
    if q < 1.0: return q*(-3.0 + 2.25*q)
    if q < 2.0: t = 2.0 - q; return -0.75*t*t
    return 0.0
```

3.4.2 Grille hexagonale pour les conditions initiales — piste à explorer

Dans notre simulation, les particules sont initialisées sur une grille carrée classique avec un filtre circulaire, puis perturbées par un bruit gaussien d'écart-type d.

```
nx   = int(math.ceil(math.sqrt(N)))
x1d  = np.linspace(-R, R, nx)
X, Y = np.meshgrid(x1d, x1d)
pts  = np.column_stack((X.ravel(), Y.ravel()))
pts  = pts[pts[:, 0]**2 + pts[:, 1]**2 <= R**2][:N]
pts += d * rng.standard_normal(pts.shape)
```

Une amélioration possible serait d'utiliser une grille hexagonale (pavage le plus dense du plan, densité $2/\sqrt{3} \approx 1.155$) pour initialiser les positions.

Remarque

Ce choix de grille hexagonale est inspiré du peu de cristallographie que j'ai faite, où ce pavage correspond à la structure cubique face centrée.

3.5 Lien avec la thèse de Roland (2021)

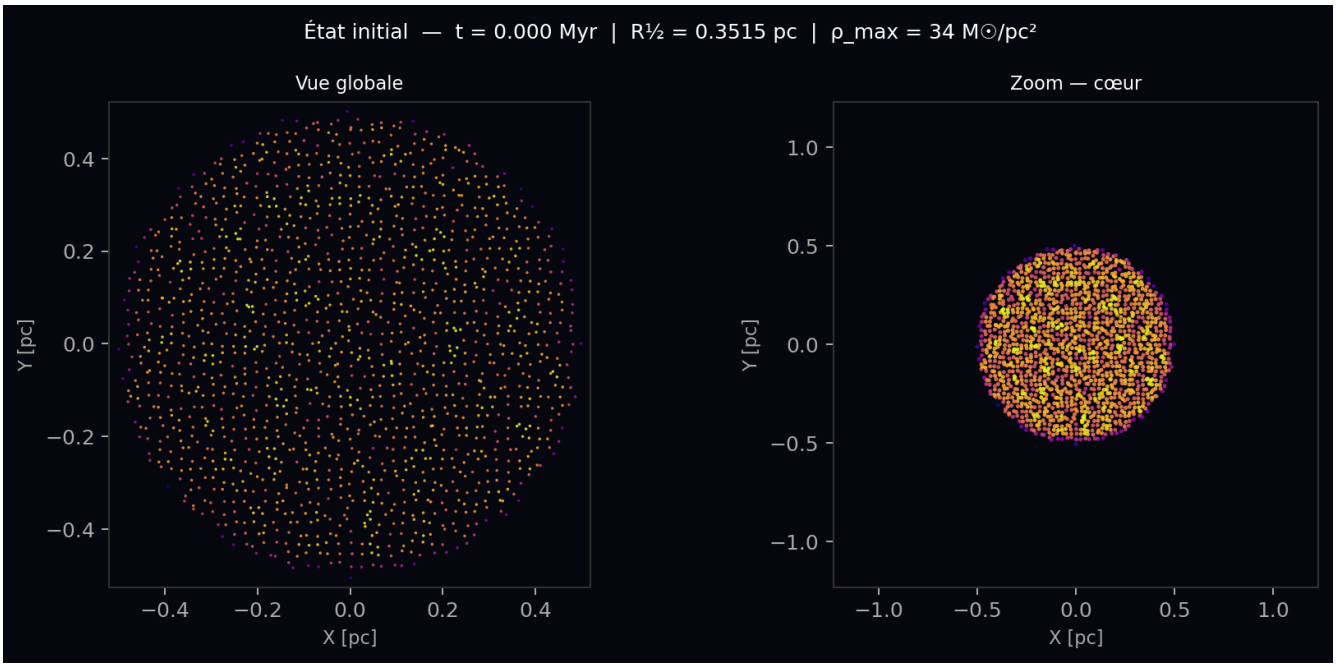
J'ai lu quelques parties de la thèse de Timothé Roland (Université de Strasbourg, 2021), intitulée *Identification et analyse des structures stellaires émergeant des régions de formation d'étoiles*[9]. Ce travail porte sur la fragmentation gravitationnelle de nuages de gaz et la formation d'amas stellaires — ce qui est exactement ce qu'on cherche à reproduire ici, en beaucoup plus simple. Une chose qui m'a frappé : Roland souligne que travailler en 2D (comme nous le faisons) sous-estime les propriétés dynamiques réelles, ce qui confirme que notre modèle ne peut être qu'une illustration qualitative du phénomène.

3.6 Résultats de la simulation

3.6.1 Visualisation

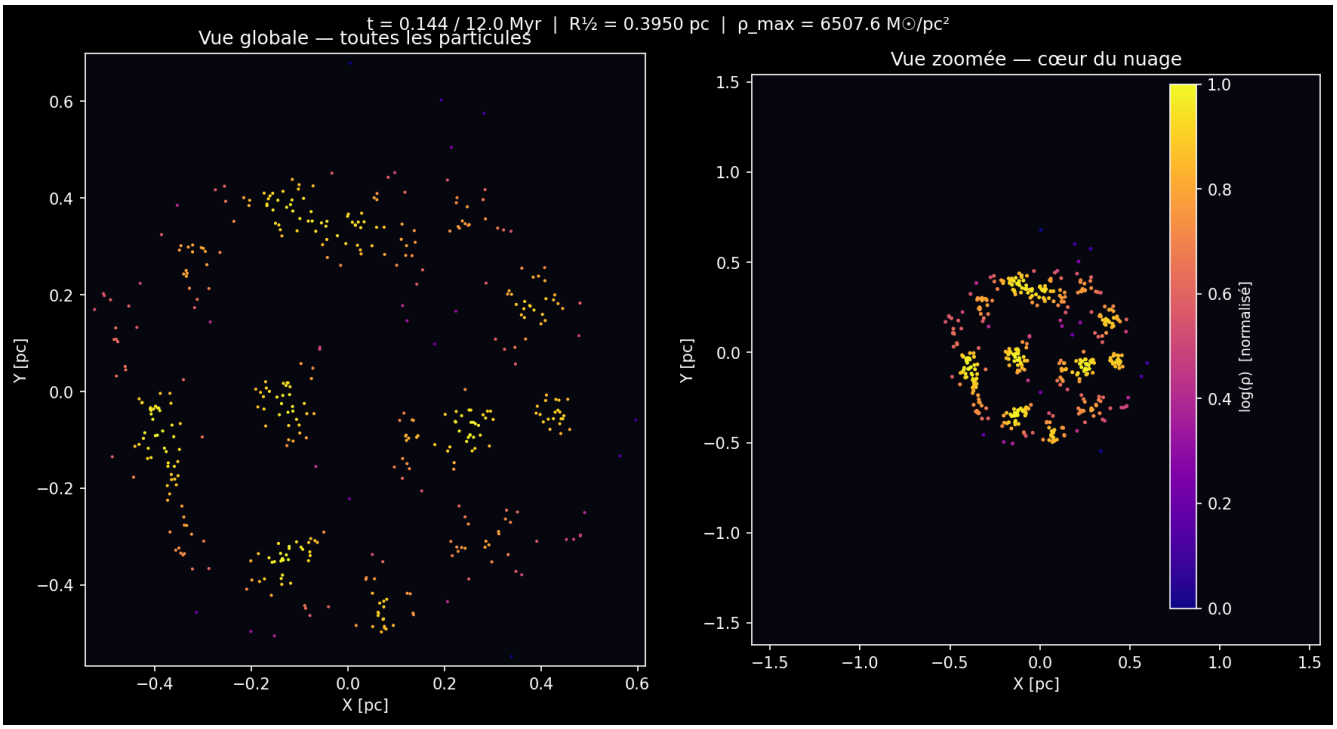
Le programme produit en temps réel deux graphiques côte à côte cf Figure 4. :

- **Vue globale** : toutes les particules, avec les axes adaptés au min/max réel. On y voit notamment les particules éjectées lors des interactions gravitationnelles.
- **Vue zoomée** : les particules situées dans un rayon $2.5 \times R_{1/2}$ autour du centre de masse, pour suivre l'effondrement du cœur.

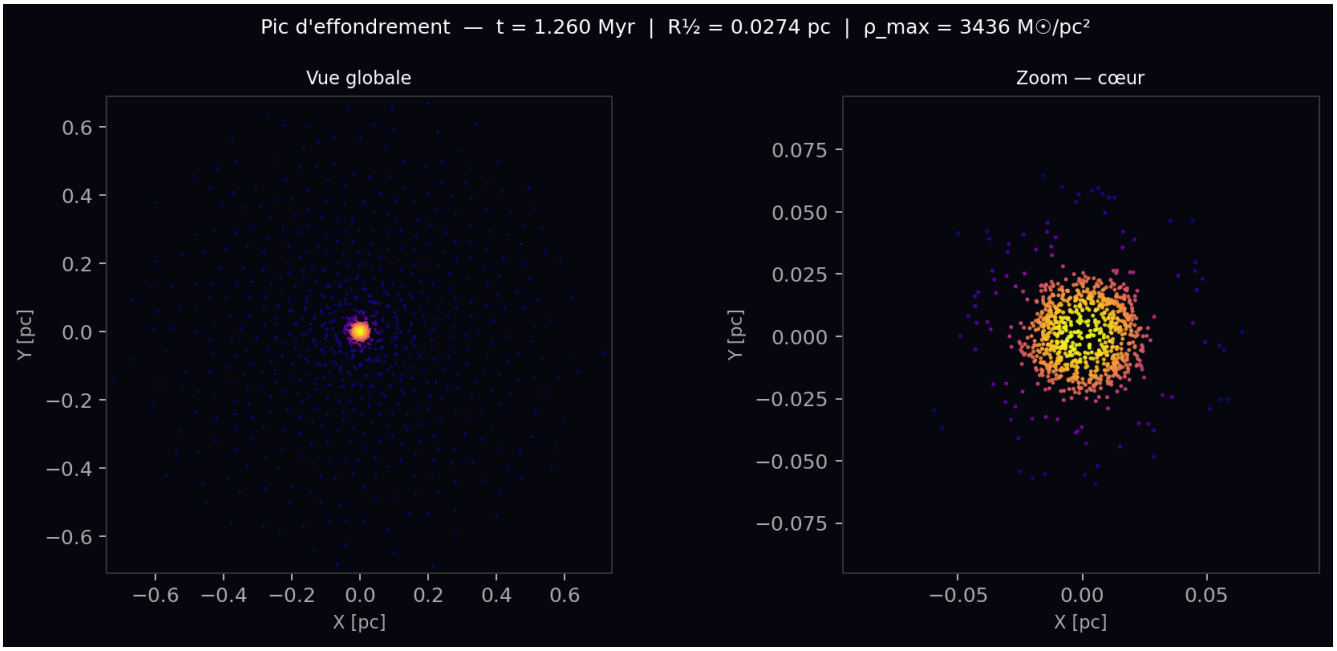


(1) État initial quasi-uniforme.

[9] `https://publication-theses.unistra.fr/public/theses_doctorat/2021/Roland_Timothe_2021_ED182.pdf`



(2) Début de la fragmentation.



(3) Pic d'effondrement.

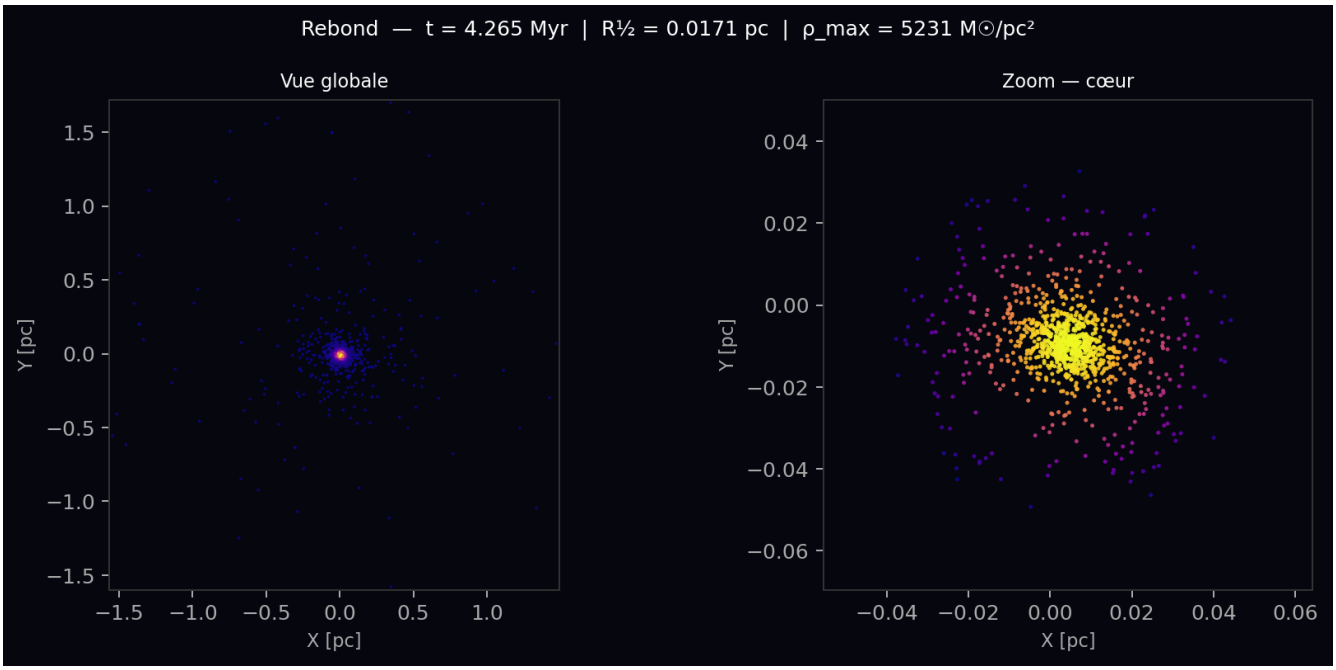


Figure 4: Évolution du nuage au cours de la simulation SPH : état initial, fragmentation, pic d'effondrement, et rebond.

3.6.2 Rebond numérique et ses causes

On observe dans la simulation que, après l'effondrement vers $t \approx 5$–8 Myr, les particules centrales ont tendance à se redilater : c'est le **rebond de Jeans numérique**. Il a trois causes distinctes.

1. La pression SPH résiste à la compression. Quand les particules s'accumulent au centre, la densité locale ρ_i monte rapidement. Or l'équation d'état donne $P = c_s^2 \rho^\gamma$, donc la pression croît avec la densité. Une fois que P dépasse localement la pression gravitationnelle, la force de pression SPH repousse les particules vers l'extérieur. C'est un mécanisme purement mécanique, réversible.

2. L'énergie cinétique de chute n'est pas dissipée. Dans notre modèle, le frottement ν est la seule source de dissipation. Avec $\nu = 0.5$ Myr^{-1}, l'énergie accumulée pendant la chute est insuffisamment amortie et se reconvertit en énergie cinétique d'expansion au moment du rebond. En augmentant ν, on observe un amortissement plus fort et un équilibre atteint sans rebond.

> **Remarque**
>
> En augmentant le coefficient de frottement à $\nu = 3.0$ Myr^{-1}, le rebond disparaît presque entièrement : les particules se stabilisent autour d'un rayon d'équilibre. C'est une confirmation que le rebond est bien d'origine dissipative (ou plutôt : due à l'absence de dissipation suffisante).

3. Les éjectés emportent de l'énergie. Certaines particules interagissent de manière très rapprochée et sont éjectées à grande vitesse (on les voit à ± 6 pc dans la vue globale). Ces éjections redistribuent de l'énergie dans le système, ce qui déséquilibre le cœur et amplifie le rebond.

3.6.3 Pourquoi n'y a-t-il pas de rebond dans la vraie vie ?

Dans la réalité, le rebond n'a pas lieu car l'énergie cinétique de la chute est dissipée de manière **irréversible** à chaque étape.

Phase isotherme — l'effondrement libre. Tant que le nuage est peu dense, il est optiquement mince : les photons émis par compression s'échappent librement et emportent l'énergie thermique hors du système. La température reste constante ($T \approx 16$ K pour Barnard 68), et la pression $P = c_s^2 \rho$ ne monte pas assez vite pour s'opposer à la gravité. L'effondrement est donc libre et accélère — c'est la phase que notre simulation isotherme ($\gamma = 1$) reproduit correctement.

Transition vers l'opacité. Lorsque la densité du cœur dépasse $\rho \sim 10^{-13}$ kg/m^3, le nuage devient opaque aux photons infrarouges. La chaleur de compression ne peut plus s'échapper : la température monte brutalement, et avec elle la pression. Ce changement de régime stoppe l'effondrement de manière irréversible : l'énergie thermique est piégée, elle ne peut pas se reconvertir en énergie cinétique d'expansion. C'est la naissance d'un **premier cœur de Larson**[10], précurseur de la proto-étoile.

Dissipation par chocs. Dans un vrai nuage en effondrement, les différentes régions tombent avec des vitesses différentes et se rencontrent en créant des ondes de choc. Ces chocs dissipent l'énergie cinétique en chaleur de manière irréversible — un mécanisme absent de notre modèle. La viscosité artificielle de Monaghan (1997)[11] est précisément conçue pour modéliser cette dissipation dans les codes SPH avancés.

Fragmentation — pourquoi pas une seule grosse étoile ? Pendant la chute isotherme, la masse de Jeans $M_J \propto T^{3/2}/\rho^{1/2}$ *diminue* au fur et à mesure que ρ augmente. Des sous-régions du nuage deviennent donc elles-mêmes gravitationnellement instables et s'effondrent indépendamment avant que le cœur principal ne rebondisse. C'est le mécanisme de **fragmentation hiérarchique**, illustré dans notre simulation par les quelques condensations secondaires visibles dans la vue zoomée (figure **??**).

Source : Wikipédia — Instabilité gravitationnelle

Lorsque la masse du nuage dépasse la masse de Jeans, le nuage se fragmente en blocs de plus en plus petits qui s'effondrent à leur tour indépendamment.

3.7 Conclusion

Notre simulation SPH reproduit fidèlement la **phase isotherme** de l'effondrement gravitationnel : la contraction libre, la montée de densité au cœur, et les premières amorces de fragmentation. En revanche, le rebond numérique observé après $t \approx 7$ Myr est un artefact du modèle — conséquence directe de l'absence de dissipation radiative et de la viscosité insuffisante.

Pour aller plus loin et supprimer ce rebond, il faudrait implémenter :

- Une **viscosité artificielle SPH** (terme de Monaghan 1997) pour dissiper l'énergie lors des chocs.
- Une **équation d'état barotropique** : isotherme ($\gamma = 1$) à basse densité, puis adiabatique ($\gamma = 5/3$) au-dessus d'un seuil critique $\rho_{\text{crit}} \sim 10^{-13}$ kg/m^3, pour modéliser le changement de régime vers l'opacité.

3.8 Nouvelles Améliorations

Dans cette partie nous allons essayer de contrecarrer le phénomène de rebond, qui n'est pas fidèle à la réalité.

Ce rebond est un artefact du modèle. Dans la réalité, il n'a pas lieu car on l'a vu au dessus, beaucoup d'énergie est dissipée par différents phénomènes. Nous allons donc maintenant ajouter à notre modèle une viscosité SPH.

Le fonctionnement de cette nouvelle feature est donné par la thèse de Bertrand Morel.

3.8.1 Implémentation

La viscosité artificielle est implémentée via la fonction `sph_viscosity_ij` qui calcule le terme Π_{ij} pour chaque paire de particules :

$$\rho_{ij}\Pi_{ij} = \begin{cases} -\alpha \cdot c_{s_{ij}} \cdot h \cdot \mu_{ij} + \beta \cdot h^2 \cdot \mu_{ij}^2 & \text{si } \mu_{ij} \leq 0 \\ 0 & \text{si } \mu_{ij} \geq 0 \end{cases} \tag{29}$$

Elle n'est activée que lorsque les particules se rapprochent ($\mu_{ij} < 0$), ce qui est cohérent avec l'idée de dissiper uniquement l'énergie des chocs. Ce terme est ensuite intégré dans `calc_acc`, où son gradient vient s'ajouter à l'accélération de chaque particule au même titre que la pression et la gravité :

[10] Larson, R.B. (1969), *Numerical calculations of the dynamics of a collapsing proto-star*, MNRAS 145, 271.

[11] Monaghan, J.J. (1997), *SPH and Riemann solvers*, J. Comput. Phys. 136, 298.

```
Pi_ij = sph_viscosity_ij(dv, dx, dy, r2, rho[i], rho[j],
                         h, eps, alpha, beta, cs)
ax -= m * Pi_ij * grad_W * dx
ay -= m * Pi_ij * grad_W * dy
```

Les paramètres $\alpha = 0.8$ et $\beta = 1.0$ ont été choisis comme point de départ raisonnable : α contrôle la viscosité linéaire (dissipation douce), β la viscosité quadratique (dissipation aux forts chocs).

Afin de vérifier l'absence de rebond, nous traçons l'évolution temporelle du rayon médian $R_{1/2}$ au cours de la simulation. En l'absence de rebond, ce rayon doit décroître monotonement jusqu'à saturation, sans remontée notable, comme illustré en Figure 5.

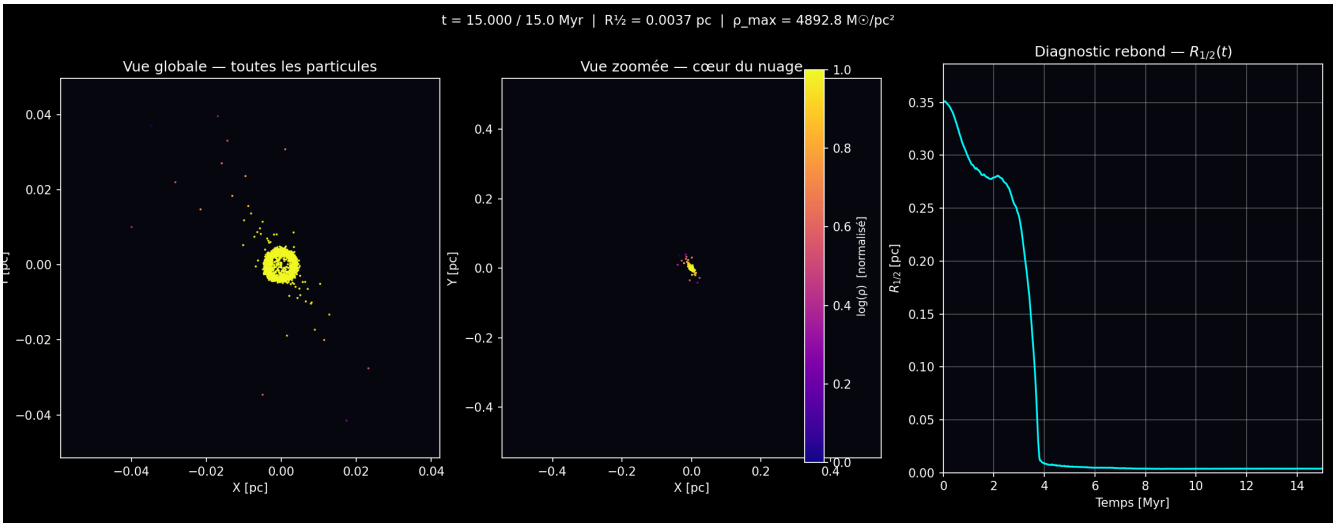


Figure 5: Évolution du rayon médian $R_{1/2}$ en fonction du temps.

Le rayon décroit casiment monotonement, on à une sorte de mini rebond, mais suffisament instable et court, pour avoir un modèle déjà plus cohérent que le précédent.

On remarque de plus qu'on à effectivement une sorte d'étoile d'un rayon égal au rayon de Jeans.

4 Extension en 3D et parallélisation sur GPU

4.1 Introduction

Jusqu'ici, toute la simulation était réalisée en 2D. Comme le souligne Roland (2021), travailler en 2D sous-estime les propriétés dynamiques réelles du système. J'ai donc décidé d'étendre la simulation à la troisième dimension.

Par ailleurs, dans le cadre de mes cours, j'apprends à utiliser **Taichi**, un module Python permettant de paralléliser des calculs sur GPU. J'ai un GPU AMD, et Numba ne supporte que les GPU NVIDIA via CUDA — j'ai donc utilisé Taichi, qui supporte AMD via Vulkan. C'était aussi l'occasion de mettre en pratique ce que j'apprends en cours ;).

4.2 Passage en 3D

Le passage en 3D est assez simple : on ajoute une composante z à chaque vecteur, et on adapte la normalisation du noyau cubique B-spline :

$$\sigma_3 = \frac{1}{\pi} \tag{30}$$

Le reste (Verlet, viscosité, gravité adoucie) reste identique à la version 2D.

4.2.1 Conditions initiales : sphère uniforme

En 2D, les particules étaient sur un disque. En 3D, on initialise une sphère uniforme par **rejet aléatoire** : on tire des points dans le cube $[-R, R]^3$ et on ne garde que ceux dont la norme est inférieure à R.

```
def init_positions(N, R, d, seed=42):
    rng = default_rng(seed)
    pts = []
    while sum(len(b) for b in pts) < N:
        batch = rng.uniform(-R, R, size=(N * 3, 3))
        mask  = np.sum(batch**2, axis=1) <= R**2
        pts.append(batch[mask])
    pts = np.vstack(pts)[:N].astype(np.float64)
    pts += d * rng.standard_normal(pts.shape)
    return pts, np.zeros((len(pts), 3))
```

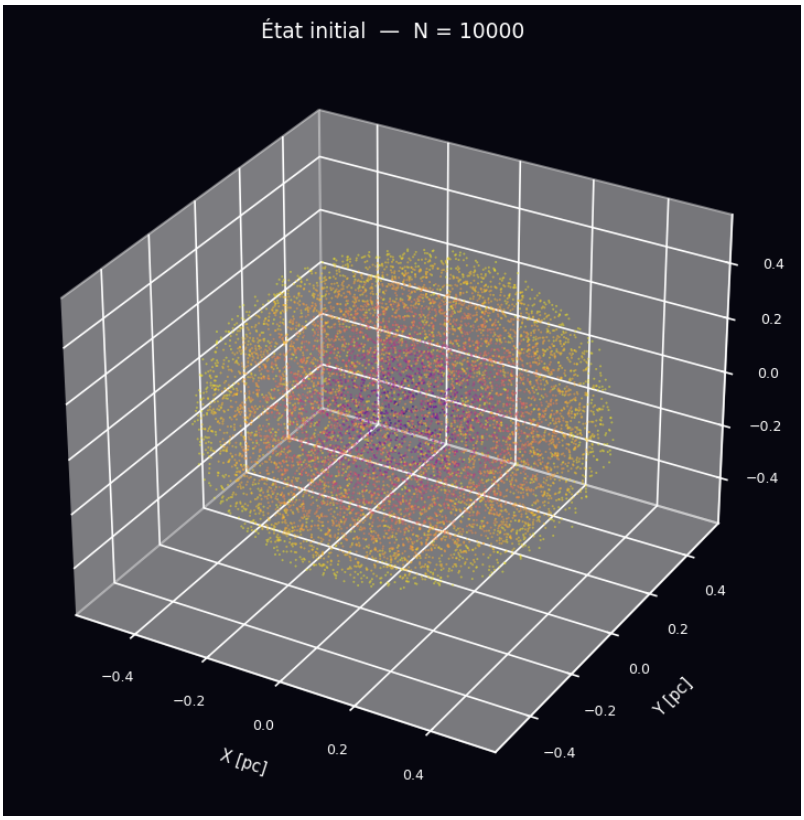


Figure 6: État initial : $N = 10\,000$ particules dans une sphère de rayon $R = 5$ pc, avec perturbation $d = 0{,}005$ pc.

4.3 Parallélisation GPU avec Taichi

Le principe de Taichi est simple : on déclare des **champs** (`ti.field`) qui vivent en mémoire GPU, et on écrit des **kernels** (`@ti.kernel`) qui s'exécutent en parallèle sur le GPU — le tout depuis Python. Les `@njit` de Numba deviennent des `@ti.func`, et les `@njit(parallel=True)` deviennent des `@ti.kernel`.

```
ti.init(arch=ti.vulkan, default_fp=ti.f32)  # AMD -> vulkan

@ti.kernel
def calc_densite(pos: ti.template(), rho: ti.template(), m: float, h: float, N: int):
    prefac = m * SIGMA_3D / h**3
    inv_h  = 1.0 / h
    for i in range(N):      # <- parall lis  automatiquement sur le GPU
        s = 0.0
        for j in range(N):
            dx = pos[i][0] - pos[j][0]
            dy = pos[i][1] - pos[j][1]
            dz = pos[i][2] - pos[j][2]
            s += noyau(ti.sqrt(dx*dx + dy*dy + dz*dz) * inv_h)
        rho[i] = prefac * s
```

La pression est également calculée sur le GPU via un kernel dédié, ce qui évite tout aller-retour CPU↔GPU dans la boucle principale. La seule interaction CPU↔GPU a lieu lors de la sauvegarde HDF5, tous les n_{affich} pas de temps.

Remarque

Contrairement à la version 2D qui affichait en temps réel avec Matplotlib, la version 3D sauvegarde les positions dans un fichier HDF5 — format standard en astrophysique numérique, facilement relisible après simulation.

4.4 Résultats

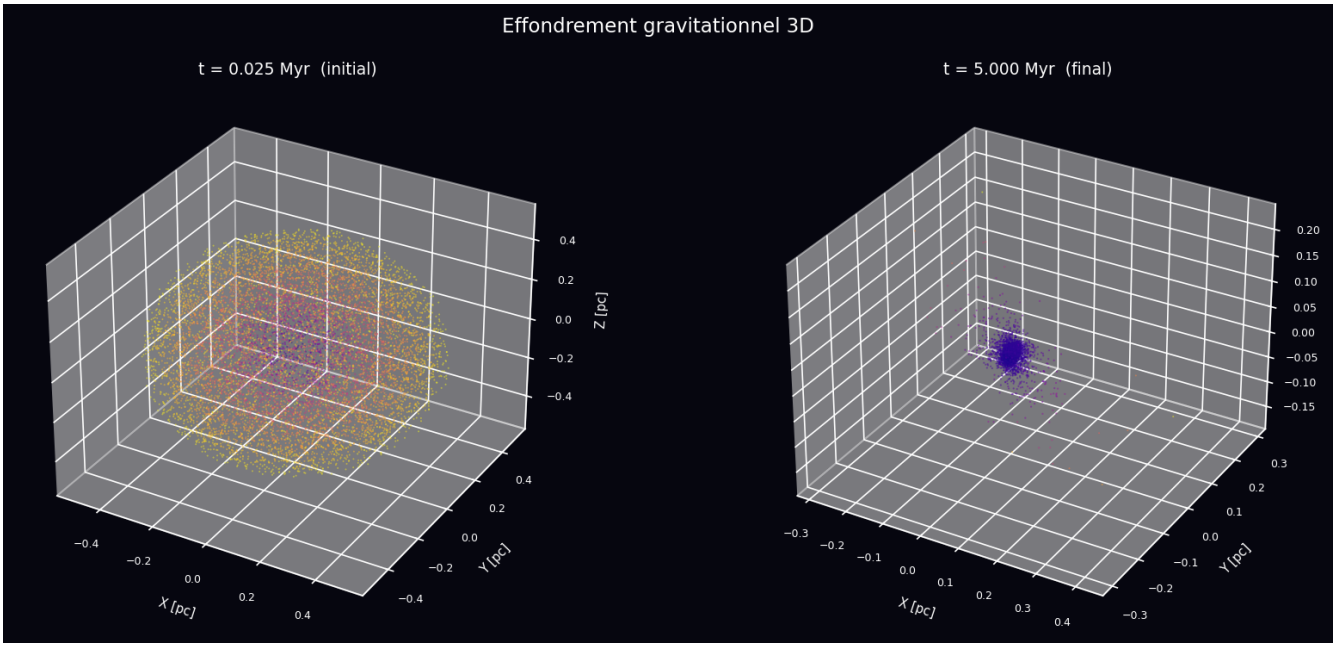


Figure 7: Effondrement gravitationnel 3D : $N = 10\,000$ particules, $M = 6000\,M_\odot$, $T = 16$ K, $R = 5$ pc. *Gauche* : état initial. *Droite* : après effondrement.

4.5 Conclusion

J'ai donc réussi à étendre la simulation en 3D et à la paralléliser sur GPU avec Taichi. Les résultats sont qualitativement cohérents avec la version 2D, et les mêmes limitations s'appliquent (pas de dissipation radiative, pas d'équation d'état barotropique). Mais c'est déjà une belle étape ;).