Estimation du diagramme de phase de l'Argon par dynamique moléculaire

Adrien Cotil, Nathan Godey

1 Introduction

Dans certaines limites thermodynamiques (faibles densités par exemple) il est aisé, grâce à l'équation d'état des gaz parfaits, d'étudier les différentes relations entre différentes grandeurs d'états, telles que la pression, la densité, le volume, la température, ou encore l'énergie interne. On remarque alors que ces relations sont en grande majorités linéaires. Cependant, en dehors de ce régime de basse densité, on observe expérimentalement des relations non linéaires.

Afin de quantifier cette non linéarité et d'en comprendre les sources, on peut essayer de retrouver par la simulation les diagrammes de phases relatifs à ces relations.

Ainsi, ce projet s'attelle à retrouver le diagramme (P, ρ, T) de l'argon (diagramme représentant la pression P en fonction de la densité ρ à température T fixée) en utilisant une modélisation issue de la physique statistique.

Nous commencerons par rappeler les objets et notations de base de la physique statistique. Nous étudierons ensuite la dynamique des systèmes isolés ainsi qu'à l'intégration numérique d'une telle dynamique. Nous étudierons de la même manière la dynamique des systèmes fermés en équilibre thermique avec un thermostat (qui découle de celle des systèmes isolés) qui correspond à notre sujet d'étude. Nous parlerons enfin de l'intégration numérique de cette dernière dynamique ainsi que de l'interprétation des résultats ainsi obtenus.

Rappels de physique statistique

Espace des phases, micro-état, macro-état et observables

La physique statistique a pour avantage, par rapport à la thermodynamique classique, de travailler à un niveau plus fondamental de la matière. Aussi, en physique statistique, on modélise classiquement un système physique comme une assemblée de particules évoluant dans un certain domaine et soumises à des

interactions (interactions de Van der Waals, champs de potentiels, collisions, ...).

Un tel système de N particules est alors parfaitement déterminé par la donnée de l'ensemble des positions et des vitesses (ou quantités de mouvement dans la pratique) de ces particules. On parle alors de **micro-état** du système. Plus précisément, si on travail en dimension $d \in \{1, 2, 3\}$, un micro-état (q, p) est un point de **l'espace des phases** $\mathcal{E} = \mathcal{D} \times \mathbb{R}^{dN}$ où $\mathcal{D}$ est le domaine exploré par les positions des particules (celui-ci peut être un ouvert de $\mathbb{R}^{dN}$ ou bien $\mathbb{R}^{dN}$ tout entier ou même le tore $\mathbb{T}^{dN}$ dans le cadre de conditions aux bords périodiques).

L'intérêt d'une telle modélisation est que l'on peut considérer une évolution régie par une dynamique issue de la mécanique classique, ce qui permet d'employer les méthodes et les résultats couramment employés dans ce domaine. Cependant, un micro-état correspond à une quantité d'information trop importante pour être exploitable directement.

Pour palier ce problème, on définit **une observable** comme une fonction $A : \mathcal{E} \to \mathbb{R}^m$ où $m \ll dN$. Par exemple, si on pose $M = diag(m_1, m_2, ..., m_N)$ où m_i est la masse de la particule i, on peut définir l'observable

$$E_c : (q, p) \mapsto \frac{1}{2} p^T M^{-1} p$$

qui correspond à l'énergie cinétique totale du système.

Cependant, même sous des hypothèses très simples, une étude centrée autour des micro-états comporte des difficultés pouvant remettre en cause l'intérêt d'une telle démarche. En effet, le caractère chaotique des équations couplé aux erreurs d'approximations informatiques ne nous permet pas d'assurer qu'une trajectoire simulée numériquement correspond à la trajectoire que l'on a voulu simuler.

On se confronte donc en général à une quantité d'information gigantesque et peu pertinente compte tenu de la remarque précédente.

C'est pour cette raison que la physique statistique utilise des outils probabilistes, afin d'étudier les valeurs moyennes prises par ces observables et l'ordre de grandeur des fluctuations autour de ces valeurs moyennes.

Ainsi, pour décrire macroscopiquement un système, au lieu de se donner des trajectoires dans l'espace des phases, on se donne des distributions de probabilité sur cette espace, appelées **macro-états**, quantifiant la fréquence à laquelle le système visite un ensemble de micro-états partageant certaines propriétés.

Calcul d'une valeur moyenne et échantillonage d'un macro-état

Dans notre problème, on cherche une fonction $f : \rho \mapsto P$ qui prend en compte les paramètres du système, à savoir le nombre N de particules, le domaine $\mathcal{D}$

dans lequel évolue la position des particules et la température T du thermostat. Une telle fonction s'exprime comme l'espérance d'une certaine observable par rapport à la mesure de probabilité μ_{NVT} correspondant à un système à température fixée. Cette distribution de probabilité, appelée **distribution canonique** ou **distribution de Boltzmann-Gibbs** sera introduite ultérieurement.

La Pression thermodynamique se défini au moyen de l'entropie de la manière suivante:

$$\left(\frac{\partial S}{\partial V}\right)_{N,E} = \frac{P}{T} \tag{1}$$

où T est la température du système.

Cette définition est cohérente avec la thermodynamique car en se plaçant dans le cas d'un système fermé (ne pouvant échangé que de l'énergie et du volume), on a :

$$dS = \left(\frac{\partial S}{\partial E}\right)_{N,V} dE + \left(\frac{\partial S}{\partial V}\right)_{N,E} dV$$

Ce qui se réécrit :

$$dE = TdS - PdV$$

Grâce au premier principe on identifie bien la pression définie par (1) à la pression thermodynamique.

Pour retrouver l'expression de l'observable associée à la pression, appelée pression microscopique, on peut par exemple établir l'expression du tenseur des contraintes puis utiliser le fait que $P = \frac{-\operatorname{Tr}(\sigma)}{3}$. L'intégrale que nous devons alors calculer s'écrit :

$$P = \frac{1}{3|\mathcal{D}|}\int_{\mathcal{E}} \psi(q,p)\ d\mu_{NVT}(q,p), \quad \psi(q,p) = p^T M^{-1} p - q^T \nabla V(q) \tag{2}$$

Cependant, la dimension de $\mathcal{E}$ étant de $2dN$ (pour 1000 particules en dimension 3 on a une intégrale en 60000 dimensions), il est impossible de calculer directement et efficacement cette intégrale.

Pour se faire, une des approches numériques envisageable est l'échantillonnage de la mesure canonique, revenant à trouver une suite de points $(q_n, p_n)_{n\geq 0} \in \mathcal{E}^{\mathbb{N}}$ telle que :

$$\lim_{N\to+\infty}\left(\frac{1}{N}\sum_{n=0}^{N}\psi(q_n,p_n)\right) = \int_{\mathcal{E}} \psi(q,p)\ d\mu_{NVT}(q,p)$$

Cette stratégie nécessite de considérer une dynamique pour laquelle la mesure μ_{NVT} est invariante (la dynamique de Langevin) que l'on établira par modification d'une dynamique plus simple : la dynamique Hamiltonienne.

2 Dynamique Hamiltonienne et son intégration

La simulation d'une dynamique moléculaire passe par la résolution numérique d'équations différentielles. La première étape de notre projet consiste en la résolution numérique d'un système d'équations différentielles construit à partir d'hypothèses simples : la dynamique Hamiltonienne.

2.1 Définition et propriétés

2.1.1 Établissement des équations et validité du modèle

Pour construire la dynamique Hamiltonienne, on va appliquer les lois de la mécanique Newtonienne à chacune des particules du système. On considère alors une assemblée de N particules et on note q_i la position de la particule i, p_i sa quantité de mouvement et m_i sa masse. On suppose aussi que chaque particule est soumise à un potentiel V dépendant à priori de sa position et de celles des autres particules.

Les lois de la mécanique appliquées à la particule i s'écrivent alors :

$$\left\{ \begin{array}{lcl} \frac{dq_i}{dt} & = & \frac{p_i}{m} \\ \frac{dp_i}{dt} & = & -\nabla_{q_i} V(q_1, q_2, ..., q_N) \end{array} \right.$$

On remarque que si on pose $H : (q,p) \mapsto \frac{1}{2} p^T M^{-1} p + V(q)$, appelé **Hamiltonien** du système, alors les vecteurs position et quantité de mouvement vérifient dans l'espace des phases :

$$\left\{ \begin{array}{lcl} \frac{dq}{dt} & = & \nabla_p H(q,p) \\ \frac{dp}{dt} & = & -\nabla_q H(q,p) \end{array} \right. \tag{3}$$

Sous certaines hypothèses sur le potentiel V, on montre que ce système admet une unique solution quelque soit la condition initiale et l'horizon temporel choisis. Une preuve de ce dernier résultat ainsi qu'une interprétation physique des hypothèses faites sur le potentiel V sont données en annexe.

Pour établir le modèle Hamiltonien, nous avons appliqué les lois de la mécanique classique (mécanique Newtonienne) à notre assemblée de particules. Cependant, au vu des ordres de grandeurs rencontrés, il est possible que des phénomènes quantiques entrent en jeu (notamment lorsque les densités sont très grandes et donc que les distances entre particules sont très petites). Cela rend la notion de trajectoire déterministe dénuée de sens (on peut penser notamment au principe d'incertitude de Heisenberg qui énonce que le produit de

l'incertitude sur la position Δq et la quantité de mouvement Δp est minoré par $\hbar/2$).

Il faut alors se placer dans des ordres de grandeurs tels que la notion de trajectoire déterministe dans l'espace des phases ait un sens, ce qui revient à dire d'une part que Δq est très petit devant la distance typique entre particules $\rho^{-1/3}$, et d'autre part que Δp est très petit devant la quantité de mouvement typique des particules mk_BT.

En considérant $\Lambda(T) = \hbar\sqrt{\frac{2\pi}{mk_BT}}$, on a alors la condition nécessaire suivante :

$$\rho^{-1/3} \gg \Lambda(T)$$

Par exemple, si on travaille à une température de 300K, la densité limite sera dans le cas de l'argon de $2.47 \times 10^{32}\ m^{-3}$, au delà de cette limite, nous sortons du cadre de la mécanique classique.

2.1.2 Ergodicité selon une trajectoire Hamiltonienne

On peut remarquer que dans la dynamique Hamiltonienne introduite plus haut, les forces considérées sont conservatives (i.e. elles dérivent d'un potentiel). Ceci a pour effet de rendre l'énergie mécanique du système invariante. En effet, on remarque que le Hamiltonien introduit plus haut correspond en réalité à l'énergie mécanique (somme de l'énergie cinétique $\frac{1}{2}p^TM^{-1}p$ et de l'énergie potentielle $V(q)$). Or, en calculant la dérivée temporelle de cet Hamiltonien sur une trajectoire, on a :

$$\begin{aligned}
\partial_t H(q(t),p(t)) &= \langle \nabla_q H(q(t),p(t)), \tfrac{dq}{dt}(t)\rangle + \langle \nabla_p H(q(t),p(t)), \tfrac{dp}{dt}(t)\rangle \\
&= \langle \nabla_q H(q(t),p(t)), \nabla_p H(q(t),p(t))\rangle - \langle \nabla_p H(q(t),p(t)), \nabla_q H(q(t),p(t))\rangle \\
&= 0
\end{aligned}$$

Ceci montre bien que l'énergie mécanique est constante tout au long de l'évolution du système.

En réalité, ce résultat exprime le fait que la dynamique Hamiltonienne correspond à la dynamique d'un **système isolé**. On peut alors chercher la distribution de probabilité correspondant au macro-état de ce système isolé. On appelle cette distribution **distribution micro-canonique** et on la note μ_{NVE}.

Pour traduire la notion **d'équilibre thermodynamique**, on comprend que cette distribution de probabilité devra être invariante par rapport à la dynamique Hamiltonienne, c'est à dire que si $(q(0),p(0)) \sim \mu_{NVE}$ alors $\forall t \in [0,+\infty[,\ (q(t),p(t)) \sim \mu_{NVE}$. On montre alors d'une part que la mesure de Lebesgue sur $\mathcal{E}$ est conservée par la dynamique Hamiltonienne (cf. Annexe) et d'autre part, comme cette distribution de probabilité doit être concentrée sur

la sous-variété $\mathcal{M}(E) = \{(q,p) \in \mathcal{E} \mid H(q,p) = E\}$, que la distribution micro-canonique se construit comme la distribution uniforme sur le volume compris entre $\mathcal{M}(E)$ et $\mathcal{M}(E + \Delta E)$ dans la limite $\Delta E \to 0$. Cette distribution de probabilité s'écrit :

$$\begin{cases} \mu_{NVE}(dq,dp) &= \frac{1}{\Omega(N,V,E)} \frac{\sigma_{\mathcal{M}(E)}(dqdp)}{|\nabla H(q,p)|} \\ \Omega(N,V,E) &= \int_{\mathcal{M}(E)} \frac{1}{|\nabla H(q,p)|} \sigma_{\mathcal{M}(E)}(dqdp) 1 \end{cases} \tag{4}$$

Où $\sigma_{\mathcal{M}(E)}(dqdp)$ est la mesure de Lebesgue induite sur la sous-variété $\mathcal{M}(E)$.

Intuitivement, la dynamique Hamiltonienne est censée échantillonner la mesure micro-canonique. C'est une hypothèse forte et fondamentale de la physique statistique : **l'hypothèse d'ergodicité**. Cette hypothèse suppose que pour toute observable A, on a

$$\int_{\mathcal{E}} A(q,p) d\mu_{NVE}(dq,dp) = \lim_{T \to +\infty} \frac{1}{T} \int_0^T A(q(t),p(t)) dt \tag{5}$$

où $(q(t),p(t))$ est la solution de la dynamique Hamiltonienne pour une certaine condition initiale (q_0,p_0).

Cette hypothèse est parfois mise à défaut, notamment lorsque l'énergie est trop petite pour passer des barrières de potentiel. On remarque cependant que dans la limite des grands systèmes, l'hypothèse ergodique est rarement invalidée.

2.2 Intégration numérique de la dynamique Hamiltonienne

2.2.1 Choix du schéma numérique

Nous avons vu à la section précédente que la dynamique Hamiltonienne possède une propriété remarquable : la conservation de la mesure de Lebesgue. En réalité, cette dynamique possède bien d'autres propriétés, dont une qui généralise cette propriété de conservation : la **symplecticité**. Afin d'étudier les propriétés de la dynamique Hamiltonienne nous introduirons le flot Hamiltonien, groupe de fonctions caractérisant la solution du problème (3).
Ces autres propriétés s'avèrent en pratique primordiales dans le choix du schéma numérique à utiliser pour intégrer les équations de la dynamique Hamiltonienne. Nous présenterons donc en premier lieu trois propriétés majeures de cette dynamique.

On considère donc le flot Hamiltonien $\phi_t : (q_0,p_0) \mapsto (q(t),p(t))$, qui est la fonction qui à une condition initiale, associe la solution de la dynamique Hamiltonienne au temps t.

On rappelle de plus qu'une méthode numérique à un pas de temps Δt correspond à une application $\Phi_{\Delta t}$ qui est une approximation de $\phi_{\Delta t}$.

Symétrie et réversibilité en temps

On remarque que, a priori, le flot Hamiltonien n'est défini que pour des temps positifs. Cependant, il est possible de donner du sens à ce flot pour des temps négatifs en posant $\phi_{-t} : (q(t), p(t)) \mapsto (q_0, p_0)$. En effet, comme il existe une unique solution pour n'importe quelle condition initiale et à n'importe quel temps, l'application ϕ_t est injective, ce qui permet de définir son inverse. On pose alors :

$$\phi_{-t} = \phi_t^{-1}$$

Cette propriété de symétrie est par ailleurs couplée à une propriété de réversibilité en temps : l'application ϕ_{-t} décrit physiquement une évolution rétrograde (cf. Annexe pour la démonstration). Cette propriété justifie la notation ϕ_{-t} .

Ces propriétés de symétrie et de réversibilité en temps devront être conservées par le schéma numérique utilisé car jouent un rôle important dans le comportement en temps long de la solution (ce sont même dans certain cas des conditions suffisantes et notamment dans le cadre des systèmes intégrables ou quasi-intégrables où la symétrie suffit à assurer la conservation des invariants pour des temps exponentiellement longs).

Or tandis que la plus part des schémas classiques sont réversibles en temps, la propriété de symétrie est quand à elle assez peu souvent vérifiée et nécessite en général une construction particulière du schéma.

Symplecticité

Une application $f : \mathcal{U} \to \mathbb{R}^{2dN}$, avec $\mathcal{U}$ un ouvert de $\mathcal{E}$, est symplectique si sa matrice jacobienne ∇f vérifie :

$$\forall (q,p) \in \mathcal{U},\ \nabla f(q,p)^T \mathcal{J} \nabla f(q,p) = \mathcal{J}$$

On peut comprendre cette propriété comme la préservation des parallélogrammes élémentaires orientés. En effet, en dimension 2 pour une application linéaire A, la conservation de ces parallélogrammes équivaut à $x_1y_2 - x_2y_1 = (Ax)_1(Ay)_2 - (Ax)_2(Ay)_1$, ce qui se réécrit:

$$\forall (x,y) \in \mathbb{R}^2 \times \mathbb{R}^2,\ x^T \mathcal{J} y = (Ax)^T \mathcal{J} (Ay)$$

Ce qui est équivalent à :

$$A^T \mathcal{J} A = \mathcal{J}$$

Or, on peut montrer que le flot Hamiltonien est symplectique (cf. Annexe). Dans la cadre des schémas numériques, la symplecticité revient à celle de $\Phi_{\Delta t}$

lorsqu'elle est appliqué à un système Hamiltonien régulier. L'intérêt étant que la symplecticité d'un schéma numérique suffit à la conservation de l'énergie sur une trajectoire simulée.

Une chose importante à noter ces schémas symplectiques sont conçus non pas pour générer des solutions approchés de problèmes exacts mais des solutions exacts de problèmes approchés. C'est ce qui explique la conservation de l'énergie sur une trajectoire simulée car, ces problèmes étant symplectiques eux aussi, l'énergie est constante le long de la trajectoire simulée (mais pas forcément identique à l'énergie théorique).

2.2.2 Construction du schéma de Stormer-Verlet

Pour construire le schéma de Stormer-Verlet, on commence par construire les schémas de Euler syplectique A et B.

Pour se faire, on sépare le système Hamiltonien en deux systèmes eux aussi Hamiltoniens :

$$\left\{ \begin{array}{rcl} \frac{dq}{dt} & = & M^{-1}p \\ \frac{dp}{dt} & = & 0 \end{array} \right. , \left\{ \begin{array}{rcl} \frac{dq}{dt} & = & 0 \\ \frac{dp}{dt} & = & -\nabla V(q) \end{array} \right. \tag{6}$$

Ces deux systèmes possèdent une solution analytique :

$$\phi_t^1(q,p) = (q + tM^{-1}p, p) \quad , \quad \phi_t^2(q,p) = (q, p - t\nabla V(q))$$

Les méthodes numériques d'Euler A et B, obtenues par une formule de Lie-Trotter, sont définies par :

$$\Phi_{\Delta t}^A = \phi_{\Delta t}^2 \circ \phi_{\Delta t}^1 \quad , \quad \Phi_{\Delta t}^B = \phi_{\Delta t}^1 \circ \phi_{\Delta t}^2 \tag{7}$$

On vérifie alors que ces deux méthodes sont explicites, d'ordre 1 et réversibles en temps. Elles ne sont cependant pas symétriques.

Cette dernière remarque nous amène donc à symétriser ces schéma appliquant une fois sur deux l'un ou l'autre, c'est le schéma de Stormer-Verlet :

$$\Phi_{\Delta t} = \Phi_{\Delta t/2}^A \circ \Phi_{\Delta t/2}^B = \phi_{\Delta t/2}^2 \circ \phi_{\Delta t}^1 \circ \phi_{\Delta t/2}^2$$

Ce qui revient numériquement à calculer :

$$\left\{ \begin{array}{rcl} p^{\frac{n+1}{2}} & = & p^n - \frac{\Delta t}{2}\nabla V(q^n) \\ q^{n+1} & = & q^n + \Delta t M^{-1} p^{\frac{n+1}{2}} \\ p^{n+1} & = & p^{\frac{n+1}{2}} - \frac{\Delta t}{2}\nabla V(q^{n+1}) \end{array} \right. \tag{8}$$

On vérifie alors que ce schéma possède toutes les propriétés recherchées : Symétrie, réversibilité en temps et symplecticité.

3 Dynamique de Langevin

On à étudié précédemment la dynamique Hamiltonienne, correspondant aux systèmes isolés, et on a vu que au moyen d'une intégration numérique, on pouvait générer une trajectoire qui, selon l'hypothèse ergodique, échantillonne la distribution micro-canonique. Or, dans le cadre de notre problème, on se place dans le cadre des système en équilibre thermique avec un thermostat ce qui a priori n'a pas de rapport avec l'étude précédente.

En réalité, on va montrer par la suite qu'il est possible de modifier la dynamique Hamiltonienne en une dynamique échantillonnant la distribution canonique (ou distribution de Boltzmann-Gibbs) correspondant au macro-état décrivant les systèmes à température fixée.

3.1 La distribution canonique

Pour établir l'expression de la distribution canonique, il est possible de passer par un raisonnement physique en considérant deux systèmes échangeant de l'énergie. Le premier (système 1), appelé **thermostat** est supposé bien plus grand que le second (système 2, qui sera notre objet d'étude) de sorte que sont état n'est que peu affecté par ce dernier. Aussi, on peut montrer que si de tels systèmes n'échangent que de l'énergie, l'équilibre thermodynamique est caractérisé par l'égalité des températures.

En utilisant d'une part l'extensivité de l'entropie, et d'autre part l'équiprobabilité des micro-états de même énergie dans un système isolé, on montre que la distribution canonique s'écrit :

$$\left\{ \begin{array}{rcl} \mu_{NVT}(q,p) & = & Z(\beta)^{-1} e^{-\beta H(q,p)} \\ Z(\beta) & = & \int_{\mathcal{E}} e^{-\beta H(q,p)} d\mu_0(dq,dp) \end{array} \right. \tag{9}$$

où $\mu_0(dq,dp) = \frac{1}{h^{3N} N!} dqdp$ (mesure de Lebesgue corrigée pour des raisons de cohérence avec la mécanique quantique) et $\beta = \frac{1}{k_B T}$.

• Cette approche reste relativement formelle et est détaillée en annexe. Il est cependant possible d'établir ce résultat de manière plus rigoureuse avec la théorie de l'information en voyant l'entropie comme le manque d'information sur le système observé.

3.2 Établissement de la dynamique de Langevin

Nous avons vu que la dynamique Hamiltonienne était construite par application des lois de la mécanique Newtonienne sur chacune des particules composant le système. Cependant, il faut noter que toutes les forces considérées dérivent d'un potentiel, ce qui en font des forces conservatives. Ceci à pour conséquence la conservation de l'énergie mécanique tout au long de l'évolution de la trajectoire,

ce qui interdit au système de visiter certaine zone de l'espace des phases. Or, dans le cadre d'un équilibre thermique, le système peut potentiellement passer par toute les points de l'espace des phases (ceci est clair au vu de l'expression de la distribution canonique).

Si on veut obtenir une dynamique qui décrit ce genre de système, il faut donc nécessairement introduire des forces non conservatives. De plus, il faut que ces forces puissent décrire le phénomène de **thermalisation**, fortement lié au phénomène de diffusion, qui est du d'une part au forces de frottement fluide (que l'on décrira par la loi de Stockes) et d'autre part au collisions entres particules (que l'on modélisera en introduisant un mouvement Brownien).

Finalement, la dynamique de Langevin est décrite par le système d'équations différentielles stochastiques suivant :

$$\left\{ \begin{array}{rcl} \frac{dq}{dt} & = & M^{-1}p \\ \frac{dp}{dt} & = & -\nabla V(q) - \xi M^{-1}p + \sigma \frac{dW_t}{dt} \end{array} \right. \tag{10}$$

où W_t est un mouvement Browien standard de dimension dN.

Pour montrer que cette dynamique rend invariante la mesure canonique, il faut écrire l'équation de Fokker-Plank que ce système (équation vérifiée par la distribution de la solution (p_t, q_t)). En notant $\chi(t,.) : \mathcal{E} \to [0,1]$ la loi du processus (q_t, p_t), l'équation de Fokker-Plank associée à (10) s'écrit :

$$\partial_t \chi = (-M^{-1}p.\nabla_q + \nabla V.\nabla_p)\chi + \xi \operatorname{Tr}\big(\nabla_p(M^{-1}p\chi)\big) + \frac{\sigma^2}{2}\Delta_p\chi \tag{11}$$

Il suffit ensuite de vérifier que $e^{-\beta H}$ est bien une solution stationnaire de cette équation quand σ et ξ sont liés par **la relation de fluctuation-dissipation**:

$$\sigma^2 = \frac{2\xi}{\beta}$$

Une preuve complète de ce résultat est donné en annexe.

3.3 Modification du schéma de Störmer-Verlet

Afin de simuler la dynamique de Langevin, nous avons dû modifier le schéma de Störmer-Verlet en conséquence, afin d'y intégrer le calcul des termes de Fluctuation/Dissipation. Pour cela, nous avons opté pour une stratégie de séparation des équations, en résolvant d'une part la dynamique Hamiltonienne et d'autre part la dynamique de Fluctuation/Dissipation.

$$\left\{ \begin{array}{l} dq_t = M^{-1}p_t dt \\ dp_t = -\nabla V\left(q_t\right) dt \end{array} \right. \oplus \left\{ \begin{array}{l} dq_t = 0 \\ dp_t = -\gamma M^{-1}p_t dt + \sqrt{\frac{2\gamma}{\beta}} dW_t \end{array} \right.$$

Il suffit alors d'ajouter à la fin du schéma de Störmer-Verlet une incrémentation des quantités de mouvement correspondant à la résolution du second système. Le nouveau schéma d'intégration s'écrit :

$$\begin{cases} p^{n+1/2} = p^n - \dfrac{\Delta t}{2}\nabla V\left(q^n\right) \\ q^{n+1} = q^n + \Delta t M^{-1} p^{n+1/2} \\ \widetilde{p}^{n+1} = p^{n+1/2} - \dfrac{\Delta t}{2}\nabla V\left(q^{n+1}\right) \\ p^{n+1} = \alpha_{\Delta t}\widetilde{p}^{n+1} + \sqrt{\dfrac{1-\alpha_{2\Delta t}}{\beta}} M G^n \end{cases}$$

où G^n suit une loi $\mathcal{N}(0, I_3)$ et $\alpha_{\Delta t} = \mathrm{e}^{-\gamma M^{-1}\Delta t}$.

4 Application au fluide d'Argon

Nous allons désormais simuler numériquement les différentes dynamiques évoquées au cours des chapitres précédents et vérifier que le modèle considéré a bien une pertinence physique, i.e. que les observables mesurées (pression P, température T, energie E) se comportent bien conformément aux observations physiques. Notamment, notre objectif final est de tracer, à température T fixée, une courbe représentant $P = f(\rho)$, la pression en fonction de la densité, et de la comparer à des résultats expérimentaux obtenus par le NIST (National Institute of Standards and Technology).

4.1 Fluide d'Argon (Ar) et sa modélisation

Afin de réaliser notre simulation, nous utilisons un fluide d'Argon ($M = 39,9 g/mol$), le troisième gaz noble du tableau périodique. L'utilisation d'un gaz noble n'est pas due au hasard : par observation d'une part, et par la règle de l'octet d'autre part, les gaz nobles tels que l'Argon forment des gaz monoatomiques au voisinage des conditions normales de température et de pression ($CNTP$).

Ils permettent donc d'obtenir des données sur les différentes variables thermodynamiques étudiées ici sur une assez large gamme. Par ailleurs, il est possible de modéliser assez simplement les interactions entre les atomes du gaz noble.

Une modélisation simple du potentiel d'interaction est le modèle des sphères dures : chaque atome est considéré comme une boule solide de rayon d_{argon}, le rayon atomique de l'Argon, et il n'y a pas d'interaction à distance. Voici l'allure du potentiel associé à ce modèle :

Cependant, ce modèle ne rend pas vraiment compte de deux phénomènes physiques observés en pratique :

- **l'interaction de répulsion de Pauli** : à courte distance, les orbitales "pleines" des atomes d'Argon (qui on le rappelle est un gaz noble) s'interpénètrent

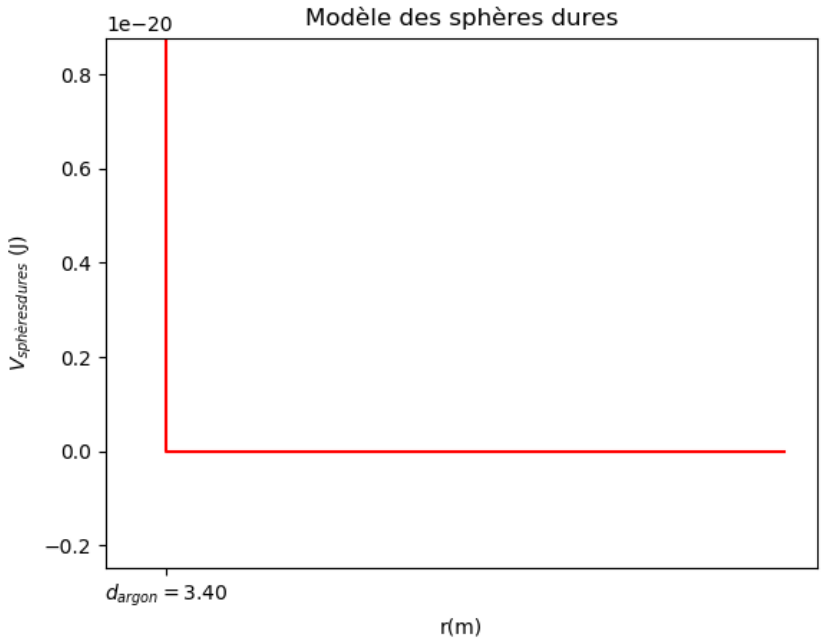

Figure 1: Potentiel d'interaction pour le modèle des sphères dures

et les fonctions d'onde des électrons se chevauchent, ce qui tend à les éloigner en moyenne. Il en résulte une interaction de répulsion à des distances d'ordre atomique.

- **l'interaction de van der Waals** : à des distances légèrement supérieures à celles du rayon atomique, la polarité électrostatique des atomes génère une interaction attractive. Cette interaction électrostatique agit à distance.

Ainsi, nous allons choisir une forme de potentiel modélisant assez fidèlement ces deux interactions : **le potentiel de Lennard-Jones**. Le potentiel de Lennard-Jones s'écrit :

$$V_{Lennard-Jones}(r) = 4\,E_0\left[\left(\frac{d_{argon}}{r}\right)^{12} - \left(\frac{d_{argon}}{r}\right)^{6}\right]$$

où E_0 est un paramètre fixé ($E_0 = 119.8k_b$)

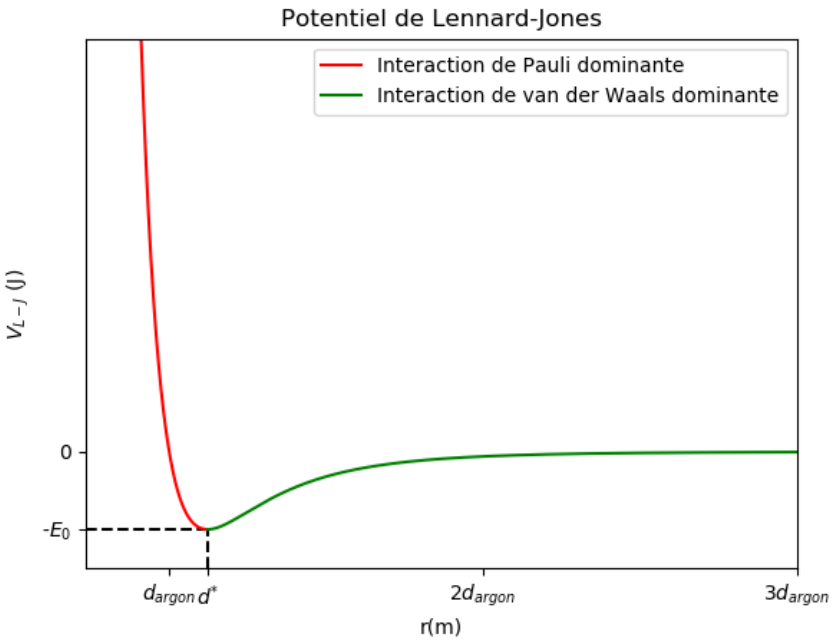

Figure 2: Potentiel d'interaction pour le fluide d'Argon de Lennard-Jones

En pratique, la valeur de ce potentiel d'interaction est faible après $2 * d_{argon}$ ce qui incite à introduire un rayon de coupure r_{cut} que l'on fixera à $2,5 * d_{argon}$, et à partir duquel on considérera que le potentiel est nul. On peut donc redéfinir le potentiel de Lennard-Jones comme suit :

$$\widetilde{V_{L-J}}(r) = (V_{L-J}(r) - V_{L-J}(r_{cut}))1_{r<r_{cut}}$$

ce qui permettra une amélioration du temps de calcul.

4.2 Adimensionnalisation

Beaucoup des ordres de grandeur avec lesquels nous travaillons sont infinitésimaux dans le Système International. Si nous simulons directement notre modèle à partir des paramètres en unités SI, il faut s'attendre au mieux à une complexité spatiale augmentée, et au pire à des erreurs d'approximations numériques voire à des valeurs informatiquement infinies.

Par ailleurs, il est peu aisé de vérifier rapidement la cohérence d'un résultat en unités SI lors de la phase de débugage par exemple. Pour toutes ces raisons, il est plus simple de travailler dans un système d'unités "adimensionnalisé", et de considérer comme étalon des grandeurs caractéristiques de la simulation que nous allons effectuer.

Ainsi, nous fixons l'unité d'énergie E_0, l'unité de masse m où m désigne la masse d'un atome d'Argon $(39,948u)$, et l'unité de distance $2^{1/6}d_{argon}$ qui réalise le minimum du potentiel de Lennard-Jones. En désignant par une astérisque Z^* la valeur en unités réduites de la grandeur Z en unités SI, notée en majuscule on a :

- $T^* = \frac{k_b T}{E_0} = \frac{T}{119.8}$
- $\beta^* = \frac{1}{T^*}$
- $d^*_{argon} = \frac{1}{2^{1/6}}$
- $m^* = 1$
- $< ||p^*|| >= \sqrt{3T^*m^*}$ (en écrivant l'équilibre entre énergie thermique ($\frac{3k_bT}{2}$) et énergie cinétique)
- $\Delta t^* = \Delta t \sqrt{\frac{E_0}{m}} * \frac{1}{d_{argon}}$
- $P^* = P \frac{E_0}{\sqrt{2} * {d_{argon}}^3}$

4.3 Initialisation du système et paramètres

Pour simuler la dynamique de Langevin sur notre système de particules d'Argon, il faut spécifier à notre algorithme quels paramètres T et ρ utiliser pour le modèle. Ces deux paramètres vont en réalité nous permettre d'établir un état initial $(q,p)_{ini}$ restituant au mieux ces paramètres.

4.3.1 État initial des positions

Fixer les positions nous invite à nous demander dans quel espace ces positions vont être amenées à évoluer. Nous avons choisi un volume cubique de côté L à comportement torique, i.e. tel que les coordonnées d'une particule sont définies "modulo L".

Nous considérerons aussi des conditions de bord périodiques : les particules se comportent comme si le volume se répétait dans toutes les directions, et l'interaction considérée entre 2 particules A et B est l'interaction entre A et la particule "reproduite" à partir de B la plus proche de A (voir 3).

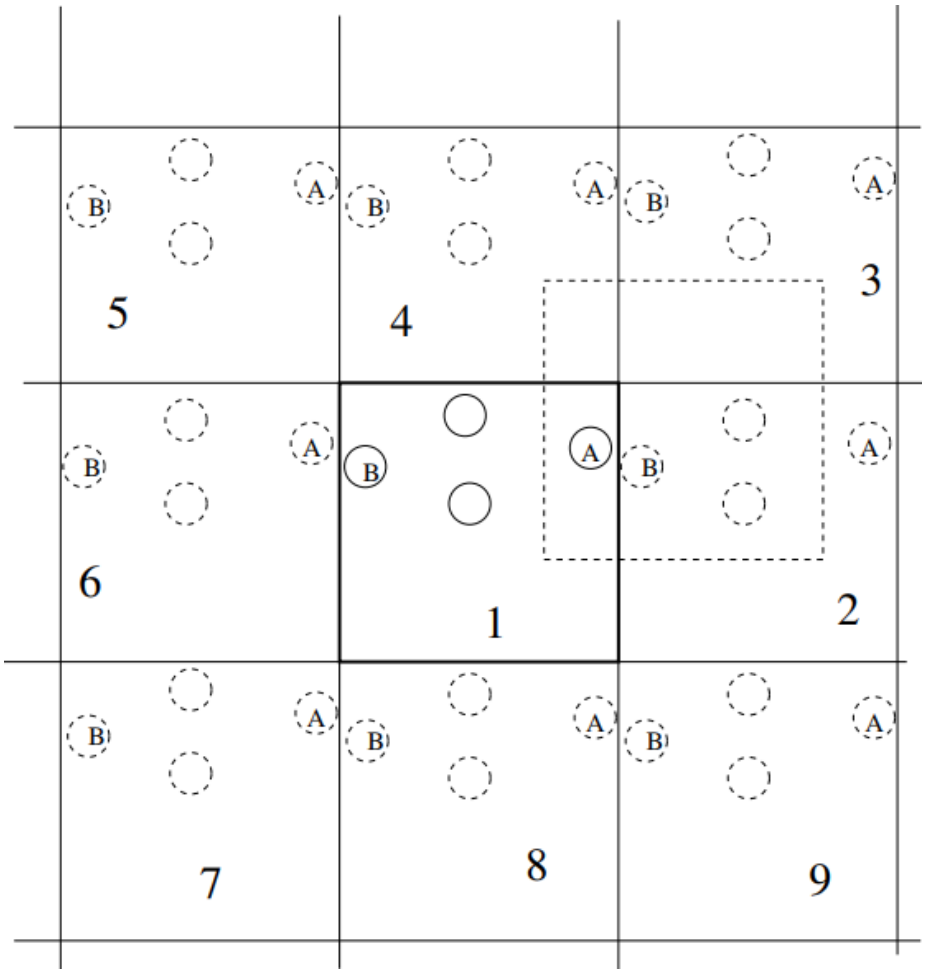

Figure 3: Illustation des conditions de bord périodiques en 2D

Une fois le contexte spatial fixé, nous devons déterminer comment répartir initialement les particules dans le volume de côté L. Une manière simple et utile à la qualité de la simulation de positionner les particules est le maillage cubique simple. Il s'agit simplement de séparer l'espace en cubes de même côté et de placer au centre de chaque cube une particule. On fixe donc un paramètre de maille a qui correspond au côté des cubes divisant notre volume de côté L.

Ce type de maillage nous impose que le nombre de particules N doit être un cube parfait ($\exists n \in \mathbb{N}^*, N = n^3$). Cette configuration initiale est intéressante

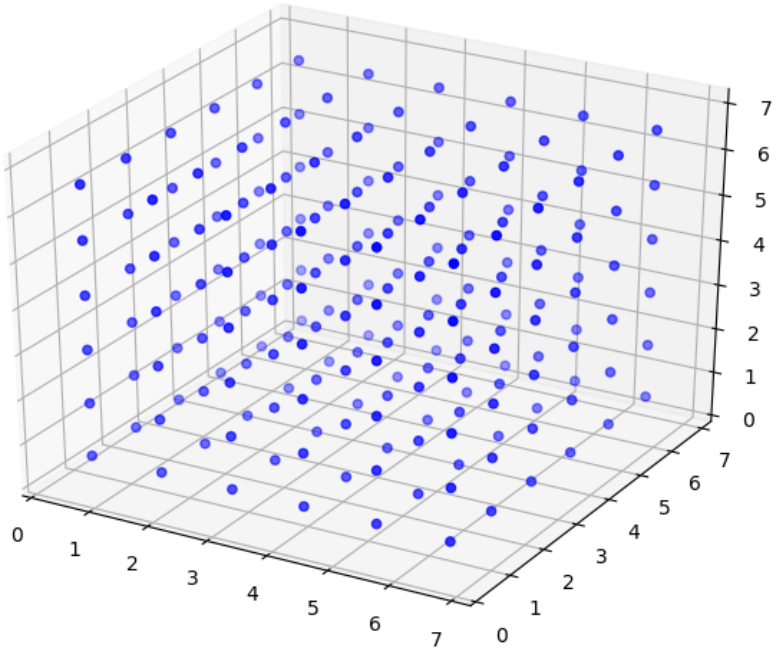

Figure 4: Un maillage cubique simple pour 216 particules

car elle est assez instable, et, la dynamique étant chaotique, on observe que le système "fond" très vite. Après avoir remarqué que $L = N^{1/3} * a$, nous allons utiliser le paramètre ρ pour fixer a : $a = (\frac{\rho}{m})^{1/3}$ où m est la masse d'un atome d'Argon. Ainsi, nous avons défini totalement le vecteur position initial q_i du système.

4.3.2 État initial des quantités de mouvement

Une fois les positions initiales des atomes fixées, il faut leur attribuer des vecteurs quantité de mouvement (p_i) initiaux. Pour cela, nous pouvons écrire l'équilibre thermodynamique entre l'énergie cinétique et l'énergie thermique puisqu'il s'agit ici d'un système isolé. Par conséquent, l'Argon étant un gaz monoatomique (de capacité thermique connue et égale à $\frac{3}{2}$), on a :

$$\frac{1}{2}m < u^2 >= \frac{3}{2}k_b T$$

On en déduit:

$$< ||p||^2 >= \sqrt{3k_b mT}$$

Nous allons donc pouvoir de manière réaliste générer les (p_i) initiaux en tirant dans les 3 directions des composantes distribuées selon $\mathcal{N}(0, \sqrt{k_b mT})$ (en veillant à normaliser par $\sqrt{3}$ pour obtenir la valeur correcte de la norme moyenne).

Une telle distribution initiale des quantités de mouvements nous permet d'observer une répartition des températures cinétiques des atomes suivant une loi $\chi^2(3)$ (non centrée et non réduite) de moyenne T, ce qui est bien ce que l'on souhaitait initialement. En pratique, pour 216 particules, cette méthode nous permet d'obtenir une erreur de seulement 1% (cf figure) sur la température cinétique initiale du système.

Nous avons donc construit $(q, p)_{ini}$ de manière à ce que le paramètre T que nous avions fixé pour notre système soit au mieux approché. Il reste alors un dernier paramètre à déterminer : γ. Qualitativement, nous avons fixé ce paramètre à $\gamma = \frac{m}{dt}$.

4.4 Simulation de la dynamique Hamiltonienne

Dans un premier temps, nous allons observer les propriétés de l'Argon soumis à la dynamique Hamiltonienne. Pour cette simulation, nous avons fixé $T = 300K$ et $\rho = 300kg/m^3$. Le pas de temps choisi est $\Delta t = 5 * 10^{-15}s$. D'après les données du NIST, dans de telles conditions de température et de pression, l'énergie interne mesurée est de $2,66kJ/mol$. L'erreur entre la valeur moyenne de l'énergie interne mesurée dans le modèle Hamiltonien et la valeur mesurée physiquement est de 4,7%. Par ailleurs, le comportement du modèle confirme le caractère conservatif de l'énergie interne obtenue grâce au schéma de Störmer-Verlet.

Pour ce qui est de la température et de la pression, nous observons que le modèle ne converge pas en moyenne vers les valeurs physiquement observées ou

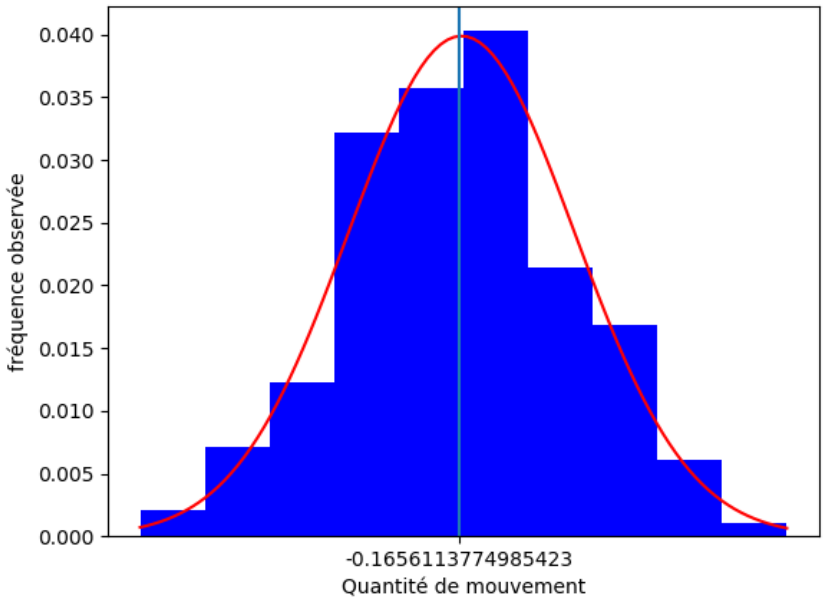

Figure 5: Superposition de la distribution théorique (en rouge) des quantités de mouvements initiales (en 1D) et de la répartition obtenue (en bleu) pour N = 216 particules à T=300K

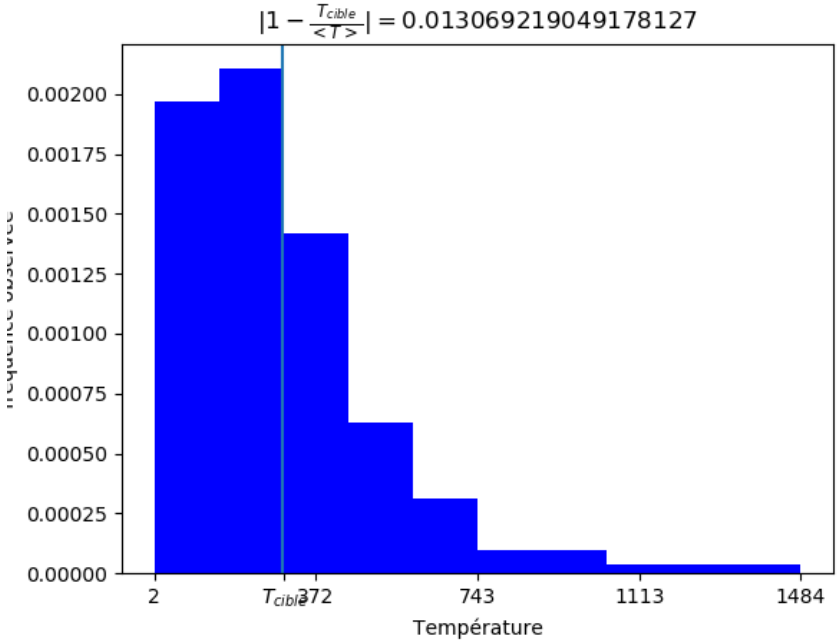

Figure 6: Répartition des températures cinétiques initiales (en Kelvin)

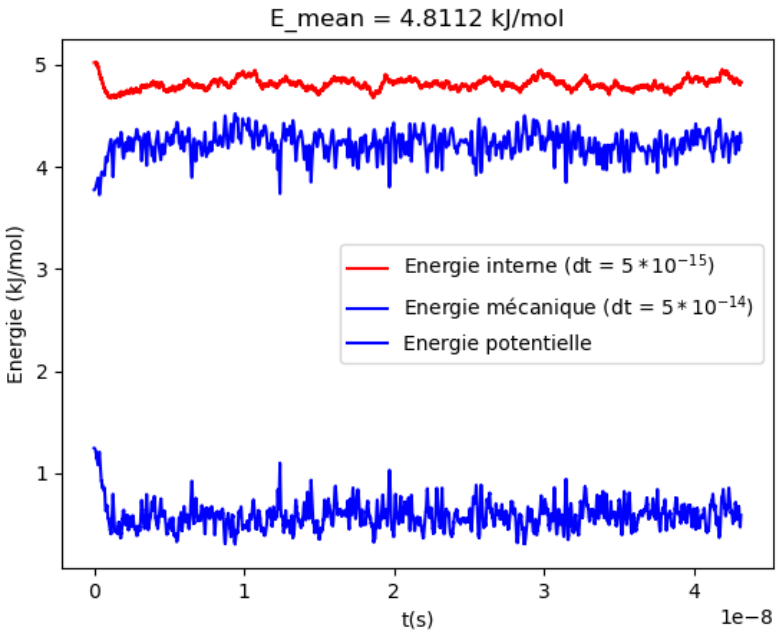

Figure 7: Évolution des énergies interne, cinétique et potentielle du système soumis à la dynamique Hamiltonienne (N = 216)

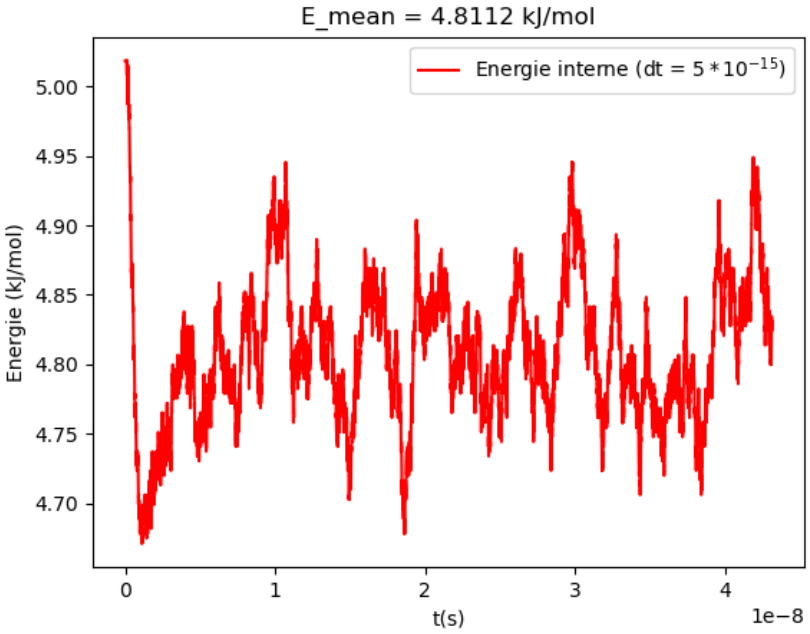

Figure 8: Évolution de l'énergie interne seule du système soumis à la dynamique Hamiltonienne (N = 216)

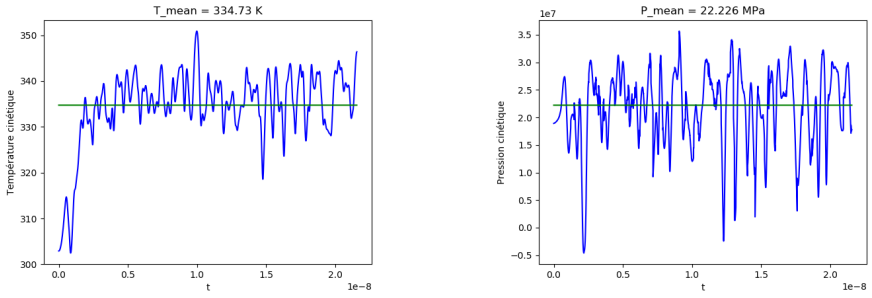

Figure 9: Température et pression cinétiques obtenues pour la dynamique Hamiltonienne

cibles (ici $T_{cible} = 300K$ et $P_{obs} = 17MPa$), dépassant ces valeurs d'environ 15%. Il est donc pertinent de simuler la dynamique de Langevin,

4.5 Simulation de la dynamique de Langevin

Nous avons ensuite simulé la dynamique de Langevin suivant le schéma ci-dessus, pour $T = 300K$ et $\rho = 300kg/m^3$, et obtenu les résultats visibles sur la Figure 9.

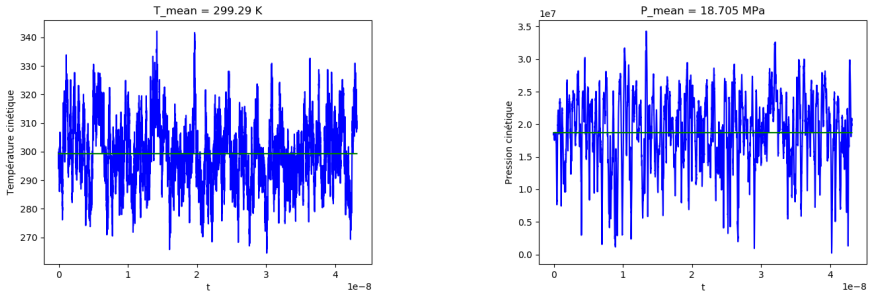

Figure 10: Température et pression cinétiques obtenues pour la dynamique de Langevin (N=363)

On remarque que les valeurs asymptotiques moyennes obtenues (299K et 18,7MPa) sont assez proches des données expérimentales, puisqu'on observe environ 0.2% d'erreur sur la température, et de 5% sur la pression.

Enfin, nous avons pu obtenir une courbe reliant la densité ρ et la pression P à la température de 300K, et la comparer aux données expérimentales. (voir figure 10)

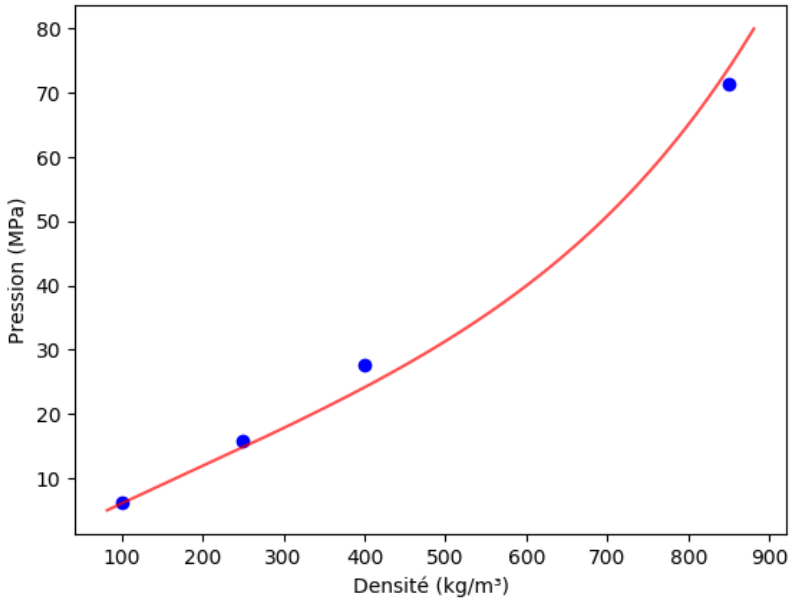

Figure 11: Pression en fonction de la densité à 300K, comparaison entre les données expérimentales du NIST (en rouge), et les résultats de l'intégration par le schéma de Störmer-Verlet adapté à la dynamique de Langevin (en bleu) pour N = 216 particules

Annexe

Existence et unicité d'une solution à la dynamique Hamiltonienne

Dans la suite, nous allons utiliser les définitions et hypothèses suivantes :

1. Soit $T \in \mathbb{R}$ un horizon temporel
2. Soit $(q_0, p_0) \in \mathcal{E}$ une condition initiale
3. on suppose que $\mathcal{D}$ est un ouvert de $\mathbb{R}^{dN}$
4. On suppose que V est au moins $C^1(\mathcal{D})$ et bornée inférieurement. On suppose de plus que son gradient est localement Lipschitzien.
5. On pose $\forall (p,q) \in \mathcal{E},\ H(p,q) = \frac{1}{2} p^T M^{-1} p + V(q)$.

On introduit de plus la matrice $\mathcal{J}$ définie par :

$$\mathcal{J} = \begin{pmatrix} 0 & I_{dN} \\ -I_{dN} & 0 \end{pmatrix}$$

On peut alors réécrire le problème (3) comme :

Trouver une fonction $\zeta : [0,T] \to \mathcal{E}$ *telle que* $\forall t \in [0,T]$

$$\begin{cases} \frac{d\zeta}{dt}(t) &= \mathcal{J} \nabla H(\zeta(t)) \\ \zeta(0) &= (q_0, p_0) \end{cases} \tag{12}$$

Comme $\mathcal{D}$ est ouvert, $\mathcal{E}$ est aussi ouvert . De plus, comme ∇V est localement Lipschitzien, $\nabla H = \begin{pmatrix} \nabla_q H \\ \nabla_p H \end{pmatrix} = \begin{pmatrix} \nabla V \\ M^{-1} p \end{pmatrix}$ est aussi localement Lipschitzien et est de plus continue (par le caractère C^1 de V).

On a donc, d'après le théorème de Cauchy-Lipschitz local l'existence et l'unicité d'une solution locale maximum du problème (12) au sens où, il existe $J \subset [0,T]$ tel que $J = [0, T^*[$ ou $J = [0,T]$ et $\zeta^* \in AC(J, \mathcal{E})$ solution de (12) sur J et tel que $\lim_{t \to T^*} (\|\zeta^*(t)\|_{\mathbb{R}^{2dN}}) = +\infty$ si $J \subsetneq [0,T]$.

Pour montrer que $\zeta^*(t)$ est bien définie sur $[0,T]$ tout entier, on va montrer que le fait que l'Hamiltonien du problème soit constant sur $[0, T^*[$ et que V soit bornée intérieurement implique que la solution est bornée et donc nécessairement que $J = [0,T]$. Pour la suite de la preuve, on pose $H_0 = H(q_0, p_0)$.

Par conservation de l'énergie sur une trajectoire dans l'espace des phases solution de la dynamique Hamiltonienne, et par le caractère borné de V, on a les deux propriétés suivantes :

$$\begin{array}{l} \forall t \in J,\ H(\zeta^*(t)) = H_0 \\ \exists v \in \mathbb{R},\ \forall q \in \Omega,\ V(q) > v \end{array}$$

On en déduit que $\frac{1}{2} p^{*T} M^{-1} p^* < H_0 - v$ ce qui implique que :

$$\|p^*(t)\|^2_{\mathbb{R}^{dN}} < 2 \min_{1 \leq i \leq N}(m_i)(H_0 - v)$$

Enfin, on a

$$\begin{array}{rcl} \frac{d}{dt}(\|q^*(t)\|^2) & = & 2 < q^*(t), p^*(t) > \\ & \leq & 2\|q^*(t)\|\|p^*(t)\| \\ & < & 2\sqrt{2 \min_{1 \leq i \leq N}(m_i)(H_0 - v)}\|q^*(t)\| \end{array}$$

Or $\frac{d}{dt}(\|q^*(t)\|^2) = 2\|q^*(t)\| \frac{d}{dt}(\|q^*(t)\|)$, ce qui permet de conclure que

$$\frac{d}{dt}(\|q^*(t)\|) < \sqrt{2 \min_{1 \leq i \leq N}(m_i)(H_0 - v)}$$

Ainsi, la croissance de la norme du vecteur des positions est au plus linéaire (et donc bornée sur J).

On conclue finalement que ζ^* est bornée sur J, et donc que $\lim_{t \to T^*}(\|\zeta^*(t)\|_{\mathbb{R}^{2dN}}) \neq +\infty$ et donc que $J = [0, T]$.

Interprétation physique

Le fait que l'existence d'une solution globale repose sur le caractère borné inférieurement du potentiel s'explique par le fait que le système évoluera vers une configuration minimisant l'énergie potentielle (ceci vient du fait que $\frac{dp}{dt} = -\nabla V$ et donc que le vecteur vitesse évolue toujours dans le sens des V décroissant). Or, si V est bornée inférieurement, on s'assure donc que l'énergie potentiel n'évoluera pas vers $-\infty$, et par conservation de l'énergie mécanique, que l'énergie cinétique (et donc la vitesse) n'évoluera pas vers $+\infty$, et donc que la solution n'explosera pas en temps fini.

Invariance de la mesure de Lebesgue par rapport à la dynamique Hamiltonienne

• On pose dans un premier temps le flot Hamiltonien $\phi_t(q_0, p_0) = (q(t), p(t))$ où $(q(t), p(t))$ est la solution de 3 pour la condition initiale (q_0, p_0).

• Ensuite, on remarque que montrer l'invariance de la mesure de Lebesgue par rapport à la dynamique Hamiltonienne revient à montrer que :

$$\forall t \in [0, +\infty[,\ \forall B \in \mathcal{B}(\mathcal{E}),\ \int_B dqdp = \int_{\phi_t(B)} dqdp$$

où $\mathcal{B}(\mathcal{E})$ est la tribu borelienne associé à l'espace des phases.

Ce qui revient à montrer que ϕ_t réalise un changement de variable de Jacobien, c'est à dire :

$$\forall t \in [0, +\infty[,\ |(\phi_t(q,p))| = |\det(\nabla_{q,p}\phi_t(q,p))| = 1$$

• Dans la suite, on pose pour un certain $(q,p) \in \mathcal{E}$, $Y(t) = \nabla_{q,p}\phi_t(q,p)$. On cherche alors à montrer que $\forall t \in [0, +\infty[,\ \det(Y(t)) = 1$.

On introduit aussi la matrice $\mathcal{J} \in \mathbb{R}^{2dN \times 2dN}$:

$$\mathcal{J} = \begin{pmatrix} 0 & Id_{dN} \\ -Id_{dN} & 0 \end{pmatrix}$$

D'après les équations de la dynamique Hamiltonienne, on a :

$$\partial_t \phi_t(q,p) = \mathcal{J}\nabla H(\phi_t(q,p))$$

• D'une part, en dérivant par rapport au temps $Y(t)$, on obtient :

$$\partial_t Y(t) = \partial_t(\nabla_{q,p}\phi_t(q,p)) = \nabla_{q,p}(\mathcal{J}\nabla H(\phi_t(q,p))) = \nabla(\mathcal{J}\nabla H(\phi_t(q,p)))\nabla_{q,p}\phi_t(q,p)$$

Ce qui nous donne par la formule de Abel-Liouville-Ostrogradskii :

$$\partial_t(\det(Y(t)) = \mathrm{Tr}(\nabla(\mathcal{J}\nabla H(\phi_t(q,p)))) \det(Y(t)) = \mathrm{div}(\mathcal{J}\nabla H(\phi_t(q,p))) \det(Y(t))$$

• Or on peut calculer $\mathrm{div}(\mathcal{J}\nabla H)$ en remarquant que :

$$\mathrm{div}(\mathcal{J}\nabla H) = \sum_{i=0}^{dN} \partial_{q_i}\partial_{p_i} H - \sum_{i=0}^{dN} \partial_{p_i}\partial_{q_i} H = 0$$

Ce qui montre que :

$$\partial_t(det(Y(t)) = 0$$

• D'autre part, comme $\phi_0 = Id$, $\nabla_{q,p}\phi_0(q,p) = Id_{2dN}$ on a $Y(0) = 1$.

Ceci montre finalement que $\forall t \in [0, +\infty[,\ |(\phi_t(q,p)| = 1$ et donc que la mesure de Lebesgue sur $\mathcal{E}$ est conservée .

Réversibilité en temps de la dynamique Hamiltonienne

La réversibilité en temps traduit l'idée que parcourir la trajectoire dans le sens des temps décroissant revient à inverser les vitesses et a parcourir la trajectoire dans le sens des temps croissant.
Mathématiquement, une dynamique est réversible en temps si sont flot f_t vérifie :

$$f_t \circ S = S \circ f_{-t}$$

où $S : (q, p) \mapsto (q, -p)$.

Or soit $(\bar{q}(t), \bar{p}(t)) = \phi_t(S(q_0, p_0))$, on a :

$$\left\{ \begin{array}{rcl} (\frac{d\bar{q}(t)}{dt}, \frac{d\bar{p}(t)}{dt}) & = & \mathcal{J}\nabla H(\bar{q}(t), \bar{p}(t)) \\ S(\frac{dq(-t)}{dt}, \frac{dp(-t)}{dt}) & = & \mathcal{J}\nabla H(S(q(-t), p(-t))) \end{array} \right.$$

Ce qui montre que $\phi_t(S(q_0, p_0))$ et $S \circ \phi_t(q_0, p_0)$ vérifie la même équation différentielle et parte de la même condition initiale, par unicité de la solution de la dynamique Hamiltonienne. On a bien $\phi_t \circ S = S \circ \phi_{-t}$.

Symplecticité de la dynamique Hamiltonienne

Posons en premier lieu $f(t) = \nabla\phi_t$ et calculon la dérivée temporel de f.

$$\frac{df}{dt} = \nabla(\frac{d\phi_t}{dt}) = \nabla(\mathcal{J}\nabla H(\phi_t)) = \mathcal{J}.\nabla^2 H(\phi_t).f$$

Ainsi, on peut calculer la dérivé temporelle de $f(t)^T.\mathcal{J}.f(t)$.

$$\begin{array}{rcl} \frac{d}{dt}(f(t)^T.\mathcal{J}.f(t)) & = & (\frac{df}{dt})^T.\mathcal{J}.f(t) + f(t)^T.\mathcal{J}.\frac{df}{dt} \\ & = & f(t)^T.\nabla^2 H(\phi_t).\mathcal{J}^T.\mathcal{J}.f(t) + f(t)^T.\mathcal{J}.\mathcal{J}\nabla^2 H(\phi_t).f(t) \end{array}$$

Comme $\mathcal{J}^T = -\mathcal{J}$, on montre bien que $\frac{d}{dt}(f(t)^T.\mathcal{J}.f(t)) = 0$. Il suffit alors d'utiliser le fait que $\phi_0 = Id$ pour conclure que $\forall t,\ f(t)^T.\mathcal{J}.f(t) = f(0)^T.\mathcal{J}.f(0) = \mathcal{J}$. Ce qui montre que la dynamique Hamiltonienne est symplectique à tout temps.

Expression de la distribution canonique

• En premier lieu, on rappelle que la distribution microcanonique s'exprime de la manière suivante :

$$\left\{ \begin{array}{rcl} \mu_{NVE}(dq, dp) & = & \frac{1}{\Omega(N,V,E)} \frac{\sigma_{\mathcal{M}(E)}(dqdp)}{|\nabla H(q,p)|} \\ \Omega(N, V, E) & = & \int_{\mathcal{M}(E)} \frac{1}{|\nabla H(q,p)|} \sigma_{\mathcal{M}(E)}(dqdp) 1 \end{array} \right.$$

où $\Omega(N, V, E)$ correspond à l'aire de l'hypersurface $\mathcal{M}(E)$ dans $\mathcal{E}$ et est appelée **fonction de partition microcanonique**.

L'entropie et la température se définissent alors par :

$$\begin{array}{lcl} S(N,V,E) & = & k_B \ln(\Omega(N,V,E)) \\ \frac{1}{T} & = & \left(\frac{\partial S}{\partial E}\right)_{N,V} \end{array}$$

Or, cette entropie ainsi définie est extensive. Pour le montrer, on considère que notre système isolé est composé de deux systèmes faiblement couplés ne pouvant échanger que de l'énergie, et on montre que la fonction de partition du grand système s'approxime très bien de la manière suivante :

$$\Omega(N,V,E) = A\Omega_1(N_1,V_1,\overline{E_1})\Omega_2(N_2,V_2,E-\overline{E_1})$$

Où Ω_1 et Ω_2 sont les fonctions de partitions des deux systèmes et $\overline{E_1}$ est la valeur la plus probable de l'énergie du système 1 (et donc, comme l'énergie total des deux systèmes est conservée, $E-\overline{E_1}$ est la valeur la plus probable de l'énergie du système 2).

Comme les micro-états de même énergie sont équiprobables, $\overline{E_1}$ réalise le maximum de $\Omega_1(N_1,V_1,\overline{E_1})\Omega_2(N_2,V_2,E-\overline{E_1})$ ce qui revient en concidérant le logarithme du produit à :

$$\frac{\partial S(N_1,V_1,\overline{E_1})}{\partial E_1} = \frac{\partial S(N_2,V_2,\overline{E_2})}{\partial E_2}$$

Ce qui montre que, par définitions de la température, l'équilibre thermique d'un système composé de deux systèmes n'échangeant que de l'énergie correspond à l'égalité de leurs températures.

• On considère maintenant que le système 2 est beaucoup plus gros que le système 1, de sorte que couplage entre le système 2 (appelée thermostat) et le système 1 (appelée système étudié) affecte de façon négligeable le système 2. Les variations d'ordres 2 ou supérieurs de l'entropie du thermostat par rapport à l'énergie du système peuvent donc être négliger ce qui donne, par définition de la température :

$$S_{th} = S_0 - \frac{E}{T}$$

où $S_0 = S_{th}(N_2,V_2,E_T)$ avec E_T l'énergie du système isolé composé de la réunion des deux systèmes, E l'énergie du système étudié et T la température du thermostat.

Il vient donc par extensivité de l'entropie, que l'entropie total vaut à une constante additive prêt $S(N,V,E) - \frac{E}{T}$ où $S(N,V,E)$ est l'entropie du système étudié. Finalement, par équiprobabilité des micro-états de même énergie dans le systèmes isolés et par définition de l'entropie, on obtient que la distribution canonique est proportionelle à $exp\left(-\frac{E}{k_B T}\right)$ ce qui en normalisant par $Z(\beta)^{-1}$ permet de conclure.

Invariance de la distribution canonique par rapport à la dynamique de Langevin

Dans un premier temps, nous allons établir l'équation de Fokker-Plank associé à une EDS plus générale que celle de la dynamique de Langevin.

Soit $b : \mathcal{E} \to \mathcal{E}$ et $\Sigma \in \mathbb{R}^{2dN}$. On considère l'équation différentielle stochastique sur $\mathcal{E}$:

$$\frac{dx}{dt} = b(x) + \Sigma \frac{dW_t}{dt}$$

Avec W_t un mouvement Brownien standart de dimension $2dN$, ce qui implique que :

$$\forall (t_1, t_2) \in [0, +\infty[,\ t_1 < t_2,\ W_{t_2 - t_1} \sim \mathcal{N}(0, (t_2 - t_1) Id_{2dN})$$

Compte tenu de cette dernière remarque, cette EDS peut être vue comme la limite du schéma numérique suivant quand le pas de temps Δt tend vers 0:

$$x^{n+1} = x^n + \Delta t b(x^n) + \sqrt{\Delta t} \Sigma G^n$$

où $(G^n)_n \in \mathbb{N}$ est une suite de variable aléatoire indépendantes suivant une loi normale centrée réduite de dimension $2dN$.

Pour établir formellement l'équation de Fokker-Plank vérifiée par la loi de la solution, on va tout d'abord étudier la variation par rapport aux itérations d'une fonction A à valeur dans $\mathbb{R}$ du précédent schéma numérique, quand le pas de temps devient petit. Par la suite, on note $\otimes$ le produit tensorielle et $\overline{\otimes}^r$ le produit contracté r fois

$$\begin{aligned} A(x^{n+1}) &= A(x^n + \Delta t b(x^n) + \sqrt{\Delta t} \Sigma G^n) \\ &= A(x_n) + \nabla A(x^n) \overline{\otimes}^1 (\Delta t b(x^n) + \sqrt{\Delta t} \Sigma G^n \\ &\quad + \tfrac{1}{2} \nabla^2 A(x^n) \overline{\otimes}^2 (\Delta t b(x^n) + \sqrt{\Delta t} \Sigma G^n)^{\otimes 2} \\ &\quad + \tfrac{1}{6} \Delta t^{\frac{3}{2}} \nabla^3 A(x^n) \overline{\otimes}^3 (\Sigma G^n)^{\otimes 3} + O(\Delta t^2) \end{aligned}$$

On a ainsi, comme G^n est x^n sont indépendant et que $\mathbb{E}(G^n) = 0$:

$$\begin{aligned} \mathbb{E}(A(x^{n+1})) &= \mathbb{E}(A(x^n)) + \Big(\mathbb{E}(\nabla A(x^n) \overline{\otimes}^1 b(x^n)) \\ &\quad + \tfrac{1}{2} \mathbb{E}(\nabla^2 A(x^n) \overline{\otimes}^2 (\Sigma G^n)^{\otimes 2}) \Big) \Delta t + O(\Delta t^2) \end{aligned}$$

De plus,

$$\begin{aligned}\mathbb{E}(\nabla^2 A(x^n)\overline{\otimes}^2(\Sigma G^n)^{\otimes 2}) &= \sum_{i,j=1}^{2dN}\mathbb{E}(\partial^2_{ij}A(x^n))\mathbb{E}((\Sigma G^n)_i(\Sigma G^n)_j)\\ &= \sum_{i,j,k,l=1}^{2dN}\mathbb{E}(\partial^2_{ij}A(x^n))\Sigma_{ik}\Sigma_{jl}\mathbb{E}(G^n_k G^n_l)\\ &= \sum_{i,j,k=1}^{2dN}\mathbb{E}(\partial^2_{ij}A(x^n))\Sigma_{ik}\Sigma_{jk}\\ &= \mathbb{E}(\nabla^2 A(x^n)\overline{\otimes}^2\Sigma\Sigma^T)\end{aligned}$$

En se plaçant alors dans la limite, $\Delta t \to 0$ et $n\Delta t \to t$, On montre que :

$$\int_{\mathcal{E}} A(x)\partial_t\chi(t,x)dx = \int_{\mathcal{E}}\mathcal{L}A(x)\chi(t,x)dx$$

où $\mathcal{L} = b.\nabla + \Sigma\Sigma^T : \nabla^2$.

Ceci montre que χ vérifie :

$$\partial_t\chi(t,.) = \mathcal{L}^*\chi(t,.)$$

où $\mathcal{L}^*$ correspond à l'opérateur adjoint de $\mathcal{L}$.

Or, on remarque que la dynamque de Langevin correspond à la dynamique précédente avec :

$$x = \begin{pmatrix} q \\ p \end{pmatrix} \quad b(x) = \begin{pmatrix} M^{-1}p \\ -\nabla V(q) + \xi M^{-1}p \end{pmatrix} \quad \Sigma = \begin{pmatrix} 0 & 0 \\ 0 & \sigma Id_{dN} \end{pmatrix}$$

Par intégration par partie, on montre que dans ce cas :

$$\mathcal{L}^* = -M^{-1}p.\nabla_q + \nabla V.\nabla_p + \xi\,\mathrm{Tr}\big(\nabla_p(M^{-1}p\ .)\big) + \frac{\sigma^2}{2}\Delta_p$$

Or en posant $\mu_{NVT}(q,p) = e^{-\beta(\frac{1}{2}p^T M^{-1}p + V(q))}$, on a :

$$\mathcal{L}^*\mu_{NVT}(q,p) = (\xi - \frac{\beta\sigma^2}{2})\,\mathrm{Tr}\big(\nabla_p(M^{-1}p.\mu_{NVT})\big)$$

Ce qui montre bien que la distribution canonique est invarient par rapport à la dynamique de Langevin à condition que $\sigma^2 = \frac{2\xi}{\beta}$

Appendice (implémentation en C++ et Python)

Quelques explications sur le code

L'utilisation de C++ était nécessaire pour des raisons de puissance de calcul limitée et de besoin d'optimisation du temps de calcul. En revanche, il est peu aisé de produire une animation ou des courbes à l'aide de C++.

Pour cela, nous avons opté pour une solution "hybride" : les simulations dynamiques sont réalisées en C++ dans une architecture en fichiers séparés, et toutes les données simulées sont renseignées à chaque itération dans des fichiers textes. A la fin de la simulation, le programme C++ lance via une commande système un script Python permettant le parsing des données stockées dans les fichiers textes, puis l'animation de la dynamique en 3 dimensions, suivi de l'affichage des courbes intéressantes présentées plus haut dans ce rapport.

Fichier main.cpp (routine principale)

```cpp
#include "utils.h"
#include "params.h"
#include "forces.h"
#include "test.h"
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
#include <limits>
#include <fstream>
#include <random>
#include <stdlib.h>



int main(){
  bool TEST_MODE = false;
  if (TEST_MODE){
    Test test = Test(2);
    test.exe(LJ_potential);
```

```
24    }
25    else{
26      bool SHOW_ITER = true;
27      bool DENSITY_PRESSURE_MODE = false;
28      params my_params;
29      srand(std::time(0));
30
31      std::cout <<"Gaz généré" <<std::endl;
32      std::vector<double> E_vect;
33      std::ofstream pos_output;
34      std::ofstream energy_output;
35      std::ofstream tempcin_output;
36      std::ofstream p_output;
37      std::ofstream pressure_output;
38      pos_output.open("./saves/positions.txt");
39      energy_output.open("./saves/energy.txt");
40      tempcin_output.open("./saves/tempcin.txt");
41      p_output.open("./saves/momentum.txt");
42
43      pos_output <<my_params.L <<std::endl;
44      pos_output <<my_params.NB_PARTICLES <<std::endl;
45      pos_output <<my_params.NB_ITER <<std::endl;
46      energy_output <<my_params.TIME_FACTOR <<std::endl;
47      energy_output <<my_params.RAW_E_0 <<std::endl;
48      energy_output <<my_params.NB_ITER <<std::endl;
49      tempcin_output <<my_params.TIME_FACTOR <<std::endl;
50      tempcin_output <<my_params.NB_ITER <<std::endl;
51      p_output <<my_params.NB_ITER <<std::endl;
52      p_output <<my_params.NB_PARTICLES <<std::endl;
53      double ta = std::time(0);
54      double tb = 0;
55      double V = 0;
56      if (DENSITY_PRESSURE_MODE){
57        pressure_output.open("./saves/pressures3.txt");
58        for (double density = 100; density<=850; density += 150){
59          my_params.RAW_DENSITY = density;
60          my_params.compute();
61          std::vector<particle> Arg_Particles = init_Argon(my_params.A, my_param
62          std::vector<std::vector<double>> dV(my_params.NB_PARTICLES);
```

```
63          pressure_output <<my_params.TIME_FACTOR <<std::endl;
64          pressure_output <<my_params.PRESS_FACTOR <<std::endl;
65          pressure_output <<my_params.NB_ITER <<std::endl;
66          for (int t = 0; t<my_params.NB_ITER; t++){
67            if (SHOW_ITER && ((t%100)==0)) {std::cout <<"Itération : " <<t <<"/"
68            for (int i=0; i<my_params.NB_PARTICLES;i++){
69              particle p = Arg_Particles[i];
70              p_output <<p.px <<std::endl;
71              pos_output <<p.x <<" " <<p.y <<" " <<p.z <<std::endl;
72            }
73            pressure_output <<P_inst(Arg_Particles, dV, my_params.L) <<std::endl
74            double current_E = E(Arg_Particles, V, my_params.L);
75            E_vect.push_back(current_E);
76            energy_output  <<current_E <<std::endl;
77            tempcin_output  <<T_cin(Arg_Particles, my_params.K_BOLTZ) <<std::end
78            Stormer_Verlet(Arg_Particles, dV, my_params.SCHEME_DT, my_params.L,
79            FD_term(Arg_Particles, my_params.SCHEME_DT, my_params.GAMMA, my_para
80            // if (SHOW_ITER) {std::cout <<"\r";}
81          }
82        }
83      }
84      else{
85        pressure_output.open("./saves/pressure.txt");
86        std::vector<particle> Arg_Particles = init_Argon(my_params.A, my_params.
87
88        pressure_output <<my_params.TIME_FACTOR <<std::endl;
89        pressure_output <<my_params.PRESS_FACTOR <<std::endl;
90        pressure_output <<my_params.NB_ITER <<std::endl;
91        std::vector<std::vector<double>> dV(Arg_Particles.size());
92        for (int t = 0; t<my_params.NB_ITER; t++){
93
94
95          ta = std::time(0);
96          if (SHOW_ITER && ((t%100)==0)) {std::cout <<"Itération : " <<t <<"/" <
97          for (int i=0; i<my_params.NB_PARTICLES;i++){
98            particle p = Arg_Particles[i];
99            p_output <<(pow(p.px,2)+pow(p.py,2)+pow(p.pz,2))/(3*p.mass) <<std::e
100           pos_output <<p.x <<" " <<p.y <<" " <<p.z <<std::endl;
101         }
```

```
102
103
104             tempcin_output  <<T_cin(Arg_Particles, my_params.K_BOLTZ) <<std::endl;
105             Stormer_Verlet(Arg_Particles, dV, my_params.SCHEME_DT, my_params.L, V)
106             double current_E = E(Arg_Particles, V, my_params.L);
107             E_vect.push_back(current_E*pow(10,-3)/my_params.N_MOL);
108             energy_output  <<current_E*pow(10,-3)/my_params.N_MOL <<" " <<E_cin(Ar
109             FD_term(Arg_Particles, my_params.SCHEME_DT, my_params.GAMMA, my_params
110             tb = std::time(0);
111             //std::cout <<"  -   Durée de l'itération : " <<tb-ta <<" s" <<std::
112             pressure_output <<P_inst(Arg_Particles, dV, my_params.L) <<std::endl;
113             if (SHOW_ITER && ((t%100)==0)) {std::cout <<"";}
114           }
115         }
116
117
118         pos_output.close();
119         std::cout <<"Moyenne de E : " <<mean(E_vect) <<std::endl;
120         std::cout <<"Ecart-type de E : " <<std_dev(E_vect) <<std::endl;
121         energy_output <<mean(E_vect) <<std::endl;
122         energy_output <<std_dev(E_vect) <<std::endl;
123         energy_output.close();
124         pressure_output.close();
125         system("python3 visualizer.py");
126       }
127
128       return 0;
129     }
```

Fichier forces.cpp (calcul des grandeurs physiques)

```
1   #include "forces.h"
2   #include <math.h>
3   #include <ctime>
4   #include <stdlib.h>
5
6
7   double LJ_potential(particle p_1, particle p_2, double L){
8     double r1_2_6_i = pow(1/r(p_1, p_2, L) , 3);
```

```
  return 4*(pow(r1_2_6_i, 2) - r1_2_6_i);
}


double d_LJ_potential(particle p_1, particle p_2, double L){
  double r1_2_i = 1./r(p_1, p_2, L);
  double r1_2_6_i = pow(r1_2_i,3);
  double unnormed_potential = 48. * r1_2_i * r1_2_6_i * (r1_2_6_i-0.5);
  return unnormed_potential;
}


std::vector<std::vector<double>> dV_update(std::vector<particle> Particles, st
  V = 0;
  for (int i=0; i<Particles.size(); i++){
    dV[i].assign(3, 0.);
  }
  double r = 0;
  double r1_2_6_i = pow(1/rc , 3);
  double Vrc = 4*(pow(r1_2_6_i, 2) - r1_2_6_i);
  for (int i = 0; i<Particles.size()-1; i++){
    for (int j = i+1; j<Particles.size(); j++){

      double d_LJ = d_LJ_potential(Particles[i], Particles[j], L);
      std::vector<double> U = direction(Particles[i], Particles[j], L);
      r = L2_norm(U);
      if (r <= rc){
        V += LJ_potential(Particles[i], Particles[j], L) - Vrc;
        dV[i][0] -= d_LJ * U[0];
        dV[i][1] -= d_LJ * U[1];
        dV[i][2] -= d_LJ * U[2];
        dV[j][0] += d_LJ * U[0];
        dV[j][1] += d_LJ * U[1];
        dV[j][2] += d_LJ * U[2];
      }

    }
  }
  return dV;
```

```
}


void Stormer_Verlet(std::vector<particle>& Particles, std::vector<std::vector<
  dV_update(Particles, dV,V, L);
  for (int i = 0; i<Particles.size(); i++){
    Particles[i].px -= dt/2. * dV[i][0];
    Particles[i].py -= dt/2. * dV[i][1];
    Particles[i].pz -= dt/2. * dV[i][2];
    Particles[i].evolve(L, dt);
  }
  dV = dV_update(Particles, dV,V,  L);
  for (int i = 0; i<Particles.size(); i++){
    Particles[i].px -= dt/2. * dV[i][0];
    Particles[i].py -= dt/2. * dV[i][1];
    Particles[i].pz -= dt/2. * dV[i][2];
  }
}


void FD_term(std::vector<particle>& Particles, double dt, double gamma, double
  double alpha_1, alpha_2;
  std::normal_distribution<double> G(0.0,1.0);

  std::default_random_engine generator(rand());
  double m = Particles[0].mass;
  alpha_1 = exp(-gamma * dt/m);
  alpha_2 = pow(alpha_1, 2);
  double fluct_factor = pow((1-alpha_2)*m/beta,0.5);
  for (int i = 0; i<Particles.size(); i++){
    Particles[i].px = alpha_1 * Particles[i].px + fluct_factor*G(generator);
    Particles[i].py = alpha_1 * Particles[i].py + fluct_factor*G(generator);
    Particles[i].pz = alpha_1 * Particles[i].pz + fluct_factor*G(generator);
  }
}


double V(std::vector<particle> Particles, double L){
 double V = 0.;
```

```
 for (int i = 0; i < Particles.size(); i++){
   particle particle_i = Particles[i];
   for (int j = i+1; j < Particles.size() ; j++){
     particle particle_j = Particles[j];
     V += LJ_potential(particle_i, particle_j, L);
   }
 }
 return V;
}


double E(std::vector<particle> Particles, double V,double L){
  double E = 0.;
  particle p;
  for (int i = 0; i < Particles.size(); i++){
    p = Particles[i];
    E += (pow(p.px,2)+pow(p.py,2)+pow(p.pz,2)) / (2*p.mass);
  }
  return E+V;
}

double E_cin(std::vector<particle> Particles, double L){
  double E = 0.;
  particle p;
  for (int i = 0; i < Particles.size(); i++){
    p = Particles[i];
    E += (pow(p.px,2)+pow(p.py,2)+pow(p.pz,2)) / (2*p.mass);
  }
  return E;
}

double T_cin(std::vector<particle> Particles, double kb){
  double u2_mean = 0;
  for (int i = 0; i<Particles.size(); i++){
    u2_mean += (pow(Particles[i].px, 2) + pow(Particles[i].py, 2) + pow(Partic
  }
  u2_mean = u2_mean/(Particles.size());
  double T = u2_mean/3;
  return T;
```

```
}

double P_inst(std::vector<particle> Particles, std::vector<std::vector<double>
  double p_mean = 0, V;
  dV_update(Particles, dV, V, L);
  for (int i = 0; i<Particles.size(); i++){
    p_mean += (pow(Particles[i].px, 2) + pow(Particles[i].py, 2) + pow(Particl
    p_mean -= Particles[i].x * dV[i][0];
    p_mean -= Particles[i].y * dV[i][1];
    p_mean -= Particles[i].z * dV[i][2];
  }
  p_mean = p_mean/(3*pow(L, 3));
  return p_mean;
}
```

Fichier utils.cpp (définitions des objets et initialisation du gaz)

```
#include "utils.h"
#include <iostream>

particle::particle(double x, double y, double z, double px, double py, double
  this->x = x;
  this->y = y;
  this->z = z;
  this->px = px;
  this->py = py;
  this->pz = pz;
}
particle::particle(){
}

void torify(double& param, double L){
  // if (param<-L || param > 2*L){
  //   std::cout <<L <<" " <<param;
  //   std::cout <<std::endl <<"Bad parameter choice : particles moving too
  // }
  param -= L * floor(param/L);

```

```
}

void particle::evolve(double L, double dt){
  this->x += (dt * this->px / this->mass);
  torify(this->x, L);
  this->y += (dt * this->py / this->mass);
  torify(this->y, L);
  this->z += (dt * this->pz / this->mass);
  torify(this->z, L);
}

double minabs(double a, double b){
  if (std::abs(a) >= std::abs(b)){
    return b;
  }
  else{
    return a;
  }
}


std::vector<double> direction(particle p_1, particle p_2, double L){
  std::vector<double> U;
  double X=0, Y=0, Z=0;
  if (p_1.x <= p_2.x){
    X = minabs(p_1.x - p_2.x, p_1.x - (p_2.x-L));
  }
  else{
    X = minabs(p_1.x - p_2.x, (p_1.x-L) - p_2.x);
  }
  if (p_1.y <= p_2.y){
    Y = minabs(p_1.y - p_2.y, p_1.y - (p_2.y-L));
  }
  else{
    Y = minabs(p_1.y - p_2.y, (p_1.y-L) - p_2.y);
  }
  if (p_1.z <= p_2.z){
    Z = minabs(p_1.z - p_2.z, p_1.z - (p_2.z-L));
  }
```

```
61    else{
62      Z = minabs(p_1.z - p_2.z, (p_1.z-L) - p_2.z);
63    }
64    U.push_back(X);
65    U.push_back(Y);
66    U.push_back(Z);
67    return U;
68  }
69
70  double r(particle p_1, particle p_2, double L){
71    std::vector<double> U = direction(p_1, p_2, L);
72    return pow(U[0],2)+pow(U[1],2)+pow(U[2],2);
73  }
74
75
76  double mean(std::vector<double> sample){
77    double S =0.;
78    for (int i=0; i<sample.size(); i++){
79      S+=sample[i];
80    }
81    return S/sample.size();
82  }
83
84  double std_dev(std::vector<double> sample){
85    double S = 0.;
86    double this_mean = mean(sample);
87    for (int i=0; i<sample.size(); i++){
88      S+=pow(sample[i]-this_mean,2);
89    }
90    return sqrt(S);
91  }
92
93
94  double L2_norm(std::vector<double> u){
95    double s=0;
96    for (int i=0;i<u.size();i++){
97      s+=pow(u[i],2);
98    }
99    return sqrt(s);
```

```
100 }
101
102
103 std::vector<double> random_vector(double norm, int dimension){
104   std::vector<double> d;
105   for (int i=0;i<dimension;i++){
106     d.push_back(((double)rand()/RAND_MAX)*2 -1);
107   }
108   double n = L2_norm(d);
109   for (int i=0;i<dimension;i++){
110     d[i] *= norm / n;
111   }
112   return d;
113 }
114
115
116 std::vector<particle> init_Argon(double A, int NB_PART_PER_DIM, double INI_P_S
117   std::vector<particle> Particles;
118   std::default_random_engine generator;
119   std::normal_distribution<double> G(0.0,INI_P_SCALE/pow(3,0.5));
120   double current_X, current_Y, current_Z;
121   for (int i = 0; i<NB_PART_PER_DIM; i++){
122     current_X = i * A + A/2;
123     for (int j = 0; j<NB_PART_PER_DIM; j++){
124       current_Y = j * A + A/2;
125       for (int k = 0; k<NB_PART_PER_DIM; k++){
126         current_Z = k * A + A/2;
127
128         double PX = G(generator);
129         double PY = G(generator);
130         double PZ = G(generator);
131         //std::cout <<PX <<" " <<PY <<" " <<PZ <<" ";
132         particle P(current_X, current_Y, current_Z, PX, PY, PZ);
133         P.mass = MASS;
134         Particles.push_back(P);
135       }
136     }
137   }
138   return Particles;
```

```
}
```

Fichier params.h (entrée des paramètres en SI)

```
#pragma once
#include <cmath>
class params{
public:
  double RAW_DENSITY = 300;
  double RAW_TEMPERATURE = 300;
  double RAW_K_BOLTZ = 1.38064852 * pow(10,-23);
  double RAW_E_0 = RAW_K_BOLTZ * 119.8;
  double RAW_D_ARGON = 0.3405 * pow(10,-9);
  double RAW_BETA = 1. / (RAW_K_BOLTZ * TEMPERATURE);
  double U = 1.6605389 * pow(10,-27);
  double N_AVOG = 6.022 * pow(10,23);
  double RAW_ARGON_MASS = 39.948 * U;
  double RAW_SCHEME_DT = 10* pow(10,-15);
  int NB_ITER = 20000;
  int DIMENSION = 3;
  double GAMMA = 1;
  int NB_PART_PER_DIM = 4, NB_PARTICLES;
  double INI_P_SCALE, DURATION, A, L, DENSITY, TEMPERATURE, K_BOLTZ, E0, D_ARG
  params();
  void compute();
};
```

Fichier params.cpp (calcul des paramètres en unités réduites)

```
#include "params.h"
#include <cmath>
#include <iostream>

params::params(){
  std::cout <<std::endl <<"----SI UNITS----" <<std::endl;
  std::cout <<"TEMPERATURE : "<<this->RAW_TEMPERATURE <<" K"<<std::endl;
  std::cout <<"DENSITY : "<<this->RAW_DENSITY <<" kg/m³"<<std::endl;
```

```
    this->NB_PARTICLES = pow(this->NB_PART_PER_DIM, 3);
    this->N_MOL = this->NB_PARTICLES / this->N_AVOG;
    this->A = pow(this->RAW_ARGON_MASS / this->RAW_DENSITY, 1/3.);
    std::cout <<"NB_PARTICLES : "<<this->NB_PARTICLES <<std::endl;
    this->L = this->NB_PART_PER_DIM * this->A;
    std::cout <<"L : "<<this->L <<" m"<<std::endl;
    std::cout <<"A : "<<this->A <<" m" <<std::endl;
    std::cout <<"dt : "<<this->RAW_SCHEME_DT <<" s" <<std::endl;
    this->L /= pow(2, 1/6.) * this->RAW_D_ARGON;
    this->A /= pow(2, 1/6.) * this->RAW_D_ARGON;


    this->K_BOLTZ = 1./119.8;
    this->TEMPERATURE = this->RAW_TEMPERATURE / 119.8;
    this->BETA = 1. / (this->TEMPERATURE);


    this->D_ARGON = 1./pow(2, 1/6.);

    this->DURATION = this->NB_ITER * this->SCHEME_DT;
    this->ARGON_MASS = this->RAW_ARGON_MASS/this->U;
    //this->ARGON_MASS = 1;

    this->INI_P_SCALE = pow(3*this->TEMPERATURE * this->ARGON_MASS, 0.5);
    this->E0 = 1.;
    this->TIME_FACTOR = (this->RAW_D_ARGON * pow(this->RAW_ARGON_MASS / this->RA
    this->SCHEME_DT = this->RAW_SCHEME_DT/this->TIME_FACTOR;

    this->GAMMA *= this->ARGON_MASS/this->SCHEME_DT;

    this->DENSITY = this->ARGON_MASS * this->NB_PARTICLES / pow(this->L, 3);

    this->PRESS_FACTOR = this->RAW_E_0/(pow(2, 1/2.)*pow(this->RAW_D_ARGON, 3));


    std::cout <<std::endl <<"----REDUCED UNITS----" <<std::endl;
    std::cout <<"L : "<<this->L <<std::endl;
    std::cout <<"A : "<<this->A <<std::endl;
    std::cout <<"TEMPERATURE : "<<this->TEMPERATURE <<std::endl;
```

```
48    std::cout <<"DENSITY : "<<this->DENSITY <<std::endl;
49    std::cout <<"DELTA_T : "<<this->SCHEME_DT <<std::endl;
50    std::cout <<"BETA : " <<this->BETA <<std::endl;
51    std::cout <<"GAMMA : " <<this->GAMMA <<std::endl;
52
53
54  }
55
56  void params::compute(){
57    std::cout <<std::endl <<"----SI UNITS----" <<std::endl;
58    std::cout <<"TEMPERATURE : "<<this->RAW_TEMPERATURE <<" K"<<std::endl;
59    std::cout <<"DENSITY : "<<this->RAW_DENSITY <<" kg/m³"<<std::endl;
60    this->NB_PARTICLES = pow(this->NB_PART_PER_DIM, 3);
61    this->A = pow(this->RAW_ARGON_MASS / this->RAW_DENSITY, 1/3.);
62    std::cout <<"NB_PARTICLES : "<<this->NB_PARTICLES <<std::endl;
63    this->L = this->NB_PART_PER_DIM * this->A;
64    std::cout <<"L : "<<this->L <<" m"<<std::endl;
65    std::cout <<"A : "<<this->A <<" m" <<std::endl;
66    std::cout <<"dt : "<<this->RAW_SCHEME_DT <<" s" <<std::endl;
67    this->L /= pow(2, 1/6.) * this->RAW_D_ARGON;
68    this->A /= pow(2, 1/6.) * this->RAW_D_ARGON;
69
70
71    this->K_BOLTZ = 1./119.8;
72    this->TEMPERATURE = this->RAW_TEMPERATURE / 119.8;
73    this->BETA = 1. / (this->TEMPERATURE);
74
75
76    this->D_ARGON = 1./pow(2, 1/6.);
77
78    this->DURATION = this->NB_ITER * this->SCHEME_DT;
79    this->ARGON_MASS = this->RAW_ARGON_MASS/this->U;
80    //this->ARGON_MASS = 1;
81
82    this->INI_P_SCALE = pow(3*this->TEMPERATURE * this->ARGON_MASS, 0.5);
83    this->E0 = 1.;
84    this->TIME_FACTOR = (this->RAW_D_ARGON * pow(this->RAW_ARGON_MASS / this->RA
85    this->SCHEME_DT = this->RAW_SCHEME_DT/this->TIME_FACTOR;
86
```

```
  this->GAMMA *= this->ARGON_MASS/this->SCHEME_DT;

  this->DENSITY = this->ARGON_MASS * this->NB_PARTICLES / pow(this->L, 3);

  this->PRESS_FACTOR = this->RAW_E_0/(pow(2, 1/2.)*pow(this->RAW_D_ARGON, 3));


  std::cout <<std::endl <<"----REDUCED UNITS----" <<std::endl;
  std::cout <<"L : "<<this->L <<std::endl;
  std::cout <<"A : "<<this->A <<std::endl;
  std::cout <<"TEMPERATURE : "<<this->TEMPERATURE <<std::endl;
  std::cout <<"DENSITY : "<<this->DENSITY <<std::endl;
  std::cout <<"DELTA_T : "<<this->SCHEME_DT <<std::endl;
  std::cout <<"BETA : " <<this->BETA <<std::endl;
  std::cout <<"GAMMA : " <<this->GAMMA <<std::endl;

}
```

Fichier visualizer.py (animation des particules et tracé des courbes)

```
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation


dens_press_mode = False

stop = False
def handle_close(e):
    global stop
    stop = True

if not dens_press_mode:

    input = open("./saves/positions.txt")
    L = float(input.readline())
```

```
    N_particle = int(input.readline())
    N_iter = int(input.readline())

    fig = plt.figure()
    fig.canvas.mpl_connect('close_event', handle_close)
    ax = Axes3D(fig)
    Xs = []
    Ys = []
    Zs = []
    for t in range(N_iter):

        X=[]
        Y=[]
        Z=[]
        for i in range(N_particle):
            l = input.readline().split(' ')
            for i_s in range(len(l)):
                l[i_s] = float(l[i_s])
            X.append(l[0])
            Y.append(l[1])
            Z.append(l[2])
        Xs.append(X)
        Ys.append(Y)
        Zs.append(Z)
    current_X = Xs[0]
    current_Y = Ys[0]
    current_Z = Zs[0]
    ax.set_xlim(0,L)
    ax.set_ylim(0,L)
    ax.set_zlim(0,L)
    wframe=None
    for t in range(1,N_iter):
        if stop:
            stop = False
            break
        if wframe:
            ax.collections.remove(wframe)
        wframe = ax.scatter(current_X, current_Y, current_Z, c='blue')
        current_X = Xs[t]
```

```
58          current_Y = Ys[t]
59          current_Z = Zs[t]
60          plt.pause(0.001)
61      plt.show()
62      input = open("./saves/momentum.txt")
63      N_iter = int(input.readline())
64      N_particle = int(input.readline())
65
66
67      fig, ax = plt.subplots()
68      fig.canvas.mpl_connect('close_event', handle_close)
69      Xs = []
70      for t in range(N_iter):
71
72          X=[]
73          for i in range(N_particle):
74              l = float(input.readline())*119.8
75              X.append(l)
76          Xs.append(X)
77      current_X = Xs[0]
78      wframe=None
79      for t in range(1,N_iter):
80          if stop:
81              stop = False
82              break
83          if wframe:
84              _ = [b.remove() for b in wframe[-1]]
85              #_ = wframe2.pop(0).remove()
86              _ = wframe3.remove()
87          wframe = ax.hist(current_X, density = True, color= 'b')
88          plt.xlabel("Température")
89          plt.ylabel("fréquence observée")
90          plt.title(r"$|1 - \frac{T_{cible}}{<T>}| = $"+str(abs(1-300/np.mean(cu
91          plt.xticks([300]+list(np.linspace(np.min(current_X),np.max(current_X),
92          #params = stats.chi2.fit(current_X,3)
93
94          x = np.linspace(np.min(current_X)-50, np.max(current_X), 100)
95          #wframe2 = ax.plot(x, stats.chi2.pdf(x, 3,params[1],params[2]), c='r
96          wframe3 = ax.axvline(np.mean(current_X))
```

```
97          current_X = Xs[t]
98
99          plt.pause(0.0001)
100     plt.show()
101
102     input = open("./saves/energy.txt")
103     time_factor = float(input.readline())
104     Eunit = float(input.readline())
105     N_iter = int(input.readline())
106     E = []
107     E_cin = []
108     for t in range(N_iter):
109         line = input.readline().split(" ")
110         E.append(float(line[0])*Eunit)
111         E_cin.append(float(line[1])*Eunit)
112     E_mean = np.mean(E[-int(N_iter/2):])
113     E_std_dev = stats.tstd(E)
114     time_axis = [i*time_factor for i in range(len(E))]
115     E_std_dev = stats.tstd(E)
116     plt.plot(time_axis, E, c='r', label="Energie interne (dt = $5*10^{-15}$)")
117     plt.plot(time_axis, E_cin, c='b', label = "Energie mécanique (dt = $5*10^{
118     plt.plot(time_axis, np.array(E)-np.array(E_cin), c='b', label = "Energie p
119     plt.legend()
120     plt.title("E_mean = "+str(E_mean)[:6]+" kJ/mol")
121     plt.xlabel("t(s)")
122     plt.ylabel("Energie (kJ/mol)")
123     plt.show()
124
125     input = open("./saves/tempcin.txt")
126     time_factor = float(input.readline())
127     Tunit = 119.8
128     N_iter = int(input.readline())
129     T = []
130
131     for t in range(N_iter):
132         a = input.readline()
133         T.append(float(a) * Tunit)
134
135     time_axis = [i*time_factor for i in range(len(T))]
```

```
136         T_mean = np.mean(T[N_iter//5:])
137         plt.plot(time_axis, T, c='b')
138         plt.plot(time_axis, [T_mean for i in range(N_iter)], c='g')
139         plt.title("T_mean = "+str(T_mean)[:6]+" K")
140         plt.xlabel("t")
141         plt.ylabel("Température cinétique")
142         plt.show()
143 
144         input = open("./saves/pressure.txt")
145         time_factor = float(input.readline())
146         Punit = float(input.readline())
147         N_iter = int(input.readline())
148         P = []
149 
150         for t in range(N_iter):
151             a = input.readline()
152             P.append(float(a) * Punit)
153 
154         time_axis = [i*time_factor for i in range(len(P))]
155         P_mean = np.mean(P[-N_iter//2:])
156         plt.plot(time_axis, P, c='b')
157         plt.plot(time_axis, [P_mean for i in range(N_iter)], c='g')
158         plt.title("P_mean = "+str(P_mean/10**6)[:6]+" MPa")
159         plt.xlabel("t")
160         plt.ylabel("Pression cinétique")
161         plt.show()
162 
163     else:
164         input = open("./saves/pressures3.txt")
165         densities = [100 + i*150 for i in [0,1,2,5]]
166         P_means = []
167         for n in range(6):
168             time_factor = float(input.readline())
169             Punit = float(input.readline())
170             N_iter = int(input.readline())
171             P = []
172 
173             for t in range(N_iter):
174                 a = input.readline()
```

```
175             P.append(float(a) * Punit)
176
177         time_axis = [i*time_factor for i in range(len(P))]
178         P_mean = np.mean(P[-N_iter//2:])
179         P_means.append(P_mean*10**-6)
180     input = open("./expdata/300K.txt")
181     lines = input.readlines()[1:]
182     D = []
183     P = []
184     for line in lines:
185         datalist = line.split("\t")
186         d, p = float(datalist[2]), float(datalist[1])
187         D.append(d)
188         P.append(p)
189     plt.plot(D, P, c='r', alpha = 0.7)
190     plt.scatter(densities, P_means, c='b')
191     plt.xlabel("Densité (kg/m³)")
192     plt.ylabel("Pression (MPa)")
193     plt.show()
```