Corso di Fisica Computazionale

- Quarta Esercitazione -

DINAMICA MOLECOLARE DI UN GAS DI XENON

Cescato Matteo, Garbi Luca, Libardi Gabriele

Issue: 1

19 agosto 2020

Università degli Studi di Trento
Dipartimento di Fisica
Via Sommarive 14, 38123
Povo (TN), Italia

Indice

Abstract

In questa esercitazione viene implementata una simulazione di dinamica molecolare con lo scopo di studiare il comportamento di un gas di Xenon nell'*ensemble* microcanonico. La dinamica del sistema è fatta evolvere mediante l'algoritmo Velocity-Verlet, i cui risultati vengono utilizzati per la stima di alcune osservabili, quali energia e pressione, di cui viene valutata la media temporale. Per meglio caratterizzare il comportamento del gas, si ricavano inoltre la funzione di distribuzione di coppia e la funzione di correlazione velocità-velocità, con cui viene stimato il coefficiente di autodiffusione del fluido. Dopo la presentazione dei fondamenti teorici alla base degli algoritmi utilizzati, ne viene spiegata l'implementazione nel codice, ed infine vengono discussi i risultati ottenuti.

1 Dinamica molecolare

Per studiare la dinamica di un sistema di N particelle, si può trattare il problema considerando il sistema finito o infinito. Nel primo caso, è necessario tenere conto delle complicazioni dovute alla geometria e alla limitatezza del sistema: in particolare, oltre alle forze interne tra le particelle, vanno opportunamente considerate anche quelle esercitate dalle pareti del contenitore in cui il gas è confinato. Nella nostra trattazione evitiamo tali complicazioni considerando il sistema infinito, omogeneo e isotropo. In particolare, il sistema viene descritto come un insieme di infinite scatole cubiche, disposte una affianco all'altra, ognuna contenente un egual numero di particelle, collocate nella medesima configurazione in ciascuna scatola. La nostra attenzione si concentra quindi sulle N particelle presenti nella scatola principale (nel seguito indicata come *scatola di simulazione*), che si immaginerà essere periodicamente ripetuta nello spazio. La scatola di simulazione è stata scelta di forma cubica per semplicità, ma altre scelte sono possibili senza inficiare i risultati. Il volume V, e quindi il lato L, della scatola di simulazione sono fissati dalla densità media ρ del sistema, secondo la relazione

$$\rho = \frac{N}{V} = \frac{N}{L^3}\,,$$

mentre le posizioni al suo interno sono descritte da un sistema di coordinate cartesiane con origine posta nel centro del cubo; di conseguenza, le particelle interne alla scatola hanno coordinate x, y e z che soddisfano le relazioni $-\frac{L}{2} \leq x \leq \frac{L}{2}$, $-\frac{L}{2} \leq y \leq \frac{L}{2}$ e $-\frac{L}{2} \leq z \leq \frac{L}{2}$. Per rendere conto della periodicità del sistema, si fissano le condizioni periodiche al contorno di Born-Von Karman. Queste impongono che se una particella esce dalla scatola di simulazione, ne entri conseguentemente un'altra con velocità uguale dal bordo opposto della scatola, nel punto individuato dalla direzione del moto della prima.

Oltre a densità e numero di particelle, per descrivere opportunamente il sistema, è necessario scegliere il potenziale intermolecolare con cui modellizzare le interazio-

ni tra le molecole costituenti il gas. Nella nostra trattazione viene considerato il cosiddetto potenziale di Lennard-Jones:

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right],$$

dove r è la distanza tra le particelle interagenti, σ e ε sono due costanti con le dimensioni di lunghezza ed energia rispettivamente. Altro parametro importante è la temperatura T iniziale, in base alla quale vengono fissate le velocità iniziali delle particelle, come verrà spiegato nella sezione 1.1.2.

Nella simulazione, si fissano le seguenti scelte per i parametri descriventi il sistema:

$$N = 125, \qquad \rho = 10^{27}\,\mathrm{m}^{-3}, \qquad m = 131.293\,\mathrm{uma},$$
$$\varepsilon = 0.02\,\mathrm{eV}, \qquad \sigma = 3.94 \cdot 10^{-10}\,\mathrm{m}, \qquad T = 300\,\mathrm{K},$$

dove m è la massa di un atomo di Xenon.

1.1 Condizioni iniziali

Le condizioni iniziali da fissare per dare inizio alla simulazione consistono nella scelta delle posizioni e delle velocità delle N particelle all'istante di tempo $t = 0$. Si noti che, lavorando in uno spazio tridimensionale, vanno in realtà fissate $3N$ componenti per le velocità e $3N$ coordinate per le posizioni, da scegliere indipendentemente.

1.1.1 Posizioni iniziali

Una possibilità per posizionare le N particelle nella scatola di simulazione consiste nell'assegnare loro posizioni casuali. Tale procedura non è però ottimale, sussistendo la possibilità (soprattutto per densità ρ elevate) che due particelle vengano posizionate a una distanza tale da provocare sulle stesse una forza repulsiva molto grande, che faccia rapidamente crescere in modo incontrollato l'energia cinetica del sistema.

Per ovviare a tale problema, si sceglie di posizionare le particelle su un reticolo cubico. In particolare, la scatola viene suddivisa in $n^3 \equiv N$ celle cubiche elementari di lato $a \equiv L/n$, ciascuna contenente una singola particella. La scatola è riempita collocando progressivamente le particelle nelle posizioni $(x, y, z) = (-L/2+ia, -L/2+ja, -L/2+ka)$ con $i, j, k = 0, 1, 2, ..., n-1$. Questa è la scelta adottata in questa trattazione; possibili alternative spesso utilizzate consistono nel posizionare 2 particelle per cella elementare, una in un vertice e l'altra nel centro (disposizione *bcc*), oppure nel riempire ogni cella elementare con 4 particelle, 3 al centro di 3 facce e la rimanente nel vertice in cui tali facce s'intersecano (disposizione *fcc*). Si osserva che la procedura adottata per la disposizione iniziale delle particelle, tenendo conto delle infinite repliche periodiche della scatola di simulazione, rende la forza iniziale (e di conseguenza l'accelerazione) esercitata su ciascuna particella perfettamente nulla per simmetria.

1.1.2 Velocità iniziali

Come le posizioni, anche le velocità iniziali $\mathbf{v}_i(0)$ (con $i = 1, 2, ..., N$) delle particelle non vengono assegnate in modo completamente casuale. Queste vengono infatti generate in base alla temperatura iniziale T del sistema, secondo la densità di probabilità di Maxwell-Boltzmann:

$$\begin{aligned} P(\mathbf{v}) &= \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \exp\left(-\frac{1}{2}\frac{m\mathbf{v}^2}{k_B T}\right) \\ &= \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \exp\left[-\frac{1}{2}\frac{m}{k_B T}\left(v_x^2 + v_y^2 + v_z^2\right)\right] \\ &= P_x(v_x)\, P_y(v_y)\, P_z(v_z)\,, \end{aligned}$$

dove

$$P_j(v_j) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2}\frac{m}{k_B T} v_j^2\right),$$

con $j = x, y, z$, è una gaussiana. Per campionare due variabili aleatorie η_1 e η_2 distribuite in modo gaussiano, con media nulla e varianza σ_v^2, a partire da due variabili stocastiche ξ_1 e ξ_2 distribuite uniformemente in [0, 1], si possono utilizzare le cosiddette formule di Box-Muller:

$$\begin{cases} \eta_1 &= \sigma_v \sqrt{-2\ln(1-\xi_1)}\cos(2\pi\xi_2) \\ \eta_2 &= \sigma_v \sqrt{-2\ln(1-\xi_1)}\sin(2\pi\xi_2) \end{cases} \tag{1}$$

Nel nostro caso, si ha $\sigma_v \equiv \sqrt{k_B T/m}$. Tali formule si utilizzano per campionare le tre componenti della velocità di ciascuna particella a partire da 4 numeri *random* generati con distribuzione uniforme in [0, 1]: i primi 2 per le componenti v_x e v_y, i rimanenti per la componente v_z. Si sceglie di utilizzare 4 numeri anziché 2 per evitare correlazioni tra le componenti della velocità della stessa particella. Con il medesimo fine, nel codice si invertiranno ξ_1 e ξ_2 nella seconda delle formule (1).

È importante infine sottolineare che, nella nostra trattazione, la temperatura T utilizzata per generare le velocità iniziali non è una quantità conservata. Il sistema è infatti descritto nell'*ensemble* microcanonico e quindi è l'energia totale del gas ad essere mantenuta costante, non la sua temperatura, che può variare sensibilmente, soprattutto nella fase iniziale della dinamica, quando il sistema non è ancora giunto a termalizzazione, perdendo memoria delle condizioni iniziali. Dopo tale fase iniziale, la temperatura sarà più stabile, ma sarà comunque interessata da importanti oscillazioni attorno al suo valor medio. Come verrà spiegato nella sezione 2, la fase iniziale di *equilibratura* del sistema non sarà considerata per il calcolo delle osservabili.

1.2 Algoritmo Velocity-Verlet

Per l'evoluzione temporale del sistema si fa uso di quello che è noto come algoritmo Velocity-Verlet. La posizione $\mathbf{r}_i(t+\delta t)$ della particella i-esima al tempo $t+\delta t$, sviluppandola in serie di Taylor attorno al tempo t, è data da

$$\mathbf{r}_i(t+\delta t) = \mathbf{r}_i(t) + \mathbf{v}_i(t)\delta t + \frac{\mathbf{F}_i\big(\mathbf{r}_{ij}(t)\big)}{2m}\delta t^2 + \mathcal{O}\big(\delta t^3\big)\,, \tag{2}$$

dove $\mathbf{v}_i$ è la velocità della particella i-esima e $\mathbf{F}_i$ è la forza esercitata sulla medesima, data da

$$\mathbf{F}_i\big(\mathbf{r}_{ij}(t)\big) = -\sum_{j\neq i} \left.\frac{\partial V(r)}{\partial r}\right|_{r_{ij}(t)} \frac{\mathbf{r}_{ij}(t)}{r_{ij}(t)}\,,$$

con $\mathbf{r}_{ij} \equiv \mathbf{r}_i - \mathbf{r}_j$. La velocità $\mathbf{v}_i(t)$ può essere invece approssimata con la seguente quantità:

$$\mathbf{v}_i = \frac{\mathbf{r}_i(t+\delta t) - \mathbf{r}_i(t-\delta t)}{2\delta t} + \mathcal{O}\big(\delta t^2\big)\,.$$

Essendo l'equazione del moto invariante per inversione temporale, la posizione $\mathbf{r}_i(t)$ può essere scritta anche come

$$\mathbf{r}_i(t) = \mathbf{r}_i(t+\delta t) - \mathbf{v}_i(t+\delta t)\delta t + \frac{\mathbf{F}_i\big(\mathbf{r}_{ij}(t+\delta t)\big)}{2m}\delta t^2 + \mathcal{O}\big(\delta t^3\big)\,. \tag{3}$$

Sommando le equazioni (2) e (3) e isolando la velocità al tempo $t+\delta t$, si ottiene

$$\mathbf{v}_i(t+\delta t) = \mathbf{v}_i(t) + \frac{\delta t}{2m}\left[\mathbf{F}_i\big(\mathbf{r}_{ij}(t)\big) + \mathbf{F}_i\big(\mathbf{r}_{ij}(t+\delta t)\big)\right] + \mathcal{O}\big(\delta t^2\big)\,. \tag{4}$$

Questa espressione fa il paio con l'equazione (2): insieme costituiscono l'algoritmo Velocity-Verlet. Si può dimostrare, mediante una trattazione che fa uso dell'equazione di Liouville e del teorema di Trotter, che l'espressione (4) è in realtà un'approssimazione della velocità all'ordine $\mathcal{O}\big(\delta t^3\big)$.

Si osserva infine che il calcolo della forza agente sulla particella i-esima andrebbe eseguito considerando le interazioni con tutte le infinite particelle presenti nella scatola di simulazione e nelle sue repliche periodiche. Nell'implementazione dell'algoritmo, come sarà spiegato nella sezione 2, si considereranno invece solo le particelle (nella scatola di simulazione o nelle sue repliche) distanti da quella in esame per meno di $L/2$, assumendo trascurabili le interazioni con le rimanenti (essendo il potenziale di Lennard-Jones nullo per $r \to \infty$).

1.3 Funzione di correlazione velocità-velocità

Una funzione di correlazione temporale $C_{AB}(t_1,\ t_2)$ fornisce informazioni sulla relazione tra due grandezze $A(t)$ e $B(t)$, calcolate rispettivamente al tempo t_1 e t_2.

In generale, si definisce

$$\widetilde{C}_{AB}(t_1,\ t_2) \equiv \langle A(t_1)B(t_2)\rangle\,,$$

dove la media è calcolata nello spazio delle fasi; nel caso in cui valga $A(t) = B(t)$, si parla di autocorrelazione. Per comodità, tipicamente si fissano $t_1 = 0$ e $t_2 = t$. In un *ensemble* statistico, sotto l'ipotesi ergodica, l'espressione precedente (nel caso di autocorrelazione) si riscrive come

$$\widetilde{C}_{AA}(t) = \langle A(0)A(t)\rangle_t\,,$$

dove con $\langle\,\cdot\,\rangle_t$ si indica l'operatore di media temporale. Spesso si rivela opportuno introdurre una normalizzazione in modo da meglio interpretare il valore della funzione; una scelta pratica consiste nel normalizzare alla varianza $\langle A^2(0)\rangle_t - \langle A(0)\rangle_t^2$; sottraendo inoltre $\langle A(0)\rangle_t^2$ a numeratore, si ottiene una funzione che vale convenientemente 1 nell'origine.

Di particolare utilità per la descrizione di una dinamica molecolare è la funzione di autocorrelazione delle velocità $\mathbf{v}(t)$, che normalizzata diventa

$$C_{\mathbf{v}}(t) \equiv C_{\mathbf{vv}}(t) = \frac{\langle \mathbf{v}(0)\cdot\mathbf{v}(t)\rangle_t - \langle \mathbf{v}(0)\rangle_t^2}{\langle \mathbf{v}^2(0)\rangle_t - \langle \mathbf{v}(0)\rangle_t^2} = \frac{\langle \mathbf{v}(0)\cdot\mathbf{v}(t)\rangle_t}{\langle \mathbf{v}^2(0)\rangle_t}, \tag{5}$$

dove l'ultima uguaglianza è verificata in quanto, nel nostro caso, non agendo sul sistema alcuna forza esterna, la velocità (vettoriale) di ciascuna particella è mediamente nulla.

È interessante vedere come la funzione di correlazione delle velocità sia legata al laplaciano del potenziale medio efficacie $V_e(\mathbf{r})$; a tale scopo, si espande in serie il numeratore:

$$\langle \mathbf{v}(0)\cdot\mathbf{v}(t)\rangle_t = \big\langle \mathbf{v}^2(0)\big\rangle_t + \langle \dot{\mathbf{v}}(0)\cdot\mathbf{v}(0)\rangle_t\ t + \frac{1}{2}\,\langle \ddot{\mathbf{v}}(0)\cdot\mathbf{v}(0)\rangle_t\ t^2 + \mathcal{O}\big(t^3\big)$$

e si osserva che il termine lineare in t si annulla. Inoltre, si trova che $\langle \ddot{\mathbf{v}}(0)\cdot\mathbf{v}(0)\rangle_t = -\langle \dot{\mathbf{v}}^2(0)\rangle_t$ e quindi la funzione di correlazione può essere riscritta come

$$C_{\mathbf{v}}(t) = 1 - \frac{1}{2}\frac{\langle \dot{\mathbf{v}}^2(0)\rangle_t}{\langle \mathbf{v}^2(0)\rangle_t}\,t^2 + \mathcal{O}\big(t^3\big)\,.$$

A questo punto, sostituendo l'accelerazione presente a numeratore con il rapporto tra la forza $\mathbf{F}$ e la massa m e ricordando che dalla meccanica statistica si ha

$$\big\langle F_\alpha^2\big\rangle = k_B T\left\langle \frac{\partial^2 V_e}{\partial r_\alpha^2}\right\rangle, \qquad \text{con } \alpha = x,\ y,\ z,$$

si può scrivere

$$C_{\mathbf{v}}(t) = 1 - \frac{1}{2}k_B T\frac{\langle \nabla^2 V_e\rangle_t}{m^2\,\langle \mathbf{v}^2(0)\rangle_t}\,t^2 + \mathcal{O}\big(t^3\big)\,.$$

Ricordando infine che, per il teorema di equipartizione, vale

$$\frac{1}{2}m\left\langle \mathbf{v}^2(0)\right\rangle_t = \frac{3}{2}k_B T\,,$$

la funzione di correlazione velocità-velocità può essere riscritta come

$$C_{\mathbf{v}}(t) = 1 - \frac{1}{6}\frac{\langle\nabla^2 V_e\rangle_t}{m}\,t^2 + \mathcal{O}\big(t^3\big)\,.$$

$C_{\mathbf{v}}(t)$, per tempi piccoli, è dunque approssimata da una parabola e il coefficiente del termine quadratico ci fornisce informazioni sul laplaciano del potenziale medio totale sentito dalla particella. Quest'ultimo è un potenziale efficace, che è in generale differente dal potenziale utilizzato per definire la singola interazione tra due particelle (quello di Lennard-Jones, nel nostro caso).
Come indicazione qualitativa, nel caso di liquidi, o addirittura cristalli, in cui le particelle sono maggiormente vincolate a rimanere in una data posizione a causa di una buca di potenziale più o meno profonda, il termine quadratico sarà più grande che nel caso di un gas. In ogni caso, per $t \to \infty$, la funzione di correlazione tenderà ad annullarsi (in quanto la velocità di una particella, al passare del tempo, è sempre meno dipendente dalla velocità al tempo iniziale), ma per densità di particelle maggiori la $C_{\mathbf{v}}(t)$ si avvicinerà a 0 più velocemente. Inoltre, per densità sufficientemente alte la funzione di correlazione può presentare anche valori negativi; tale fenomeno di anticorrelazione si verifica in quanto gli urti tra particelle molto vicine sono spesso quasi frontali e di conseguenza le velocità prima e dopo l'urto differiscono solo per il verso e non per la direzione.

Chiaramente in un contesto statistico, considerando una media su un tempo infinito, la funzione di correlazione delle velocità dovrebbe risultare uguale per tutte le N particelle del sistema. Al fine di ottimizzare il calcolo numerico, risulta però vantaggioso considerare una media di tale funzione su tutte le particelle.

1.3.1 Stima del coefficiente di autodiffusione

La funzione di correlazione velocità-velocità risulta utile anche per la stima del coefficiente di diffusione di una sostanza all'interno di un'altra. Nel caso in cui la sostanza diffusa sia la stessa di quella diffondente si parla di *autodiffusione*.
Si può dimostrare che, detto γ l'integrale

$$\gamma \equiv \frac{1}{3}\int_0^\infty \mathrm{d}t\ \left\langle \mathbf{v}(0)\cdot\mathbf{v}(t)\right\rangle_t\,,$$

attraverso la relazione di Einstein[1], si ottiene

$$\gamma = \frac{\left\langle \big(\mathbf{r}(t) - \mathbf{r}(0)\big)^2 \right\rangle_t}{6t} \,.$$

Ma per definizione lo spostamento quadratico medio delle particelle è dipendente dal tempo con una costante di proporzionalità che è il coefficiente di autodiffusione D_a. Di conseguenza, si ottiene

$$D_a = 2 \int_0^\infty \mathrm{d}t \, \left\langle \mathbf{v}(0) \cdot \mathbf{v}(t) \right\rangle_t \,. \tag{6}$$

1.4 Funzione di distribuzione di coppia

In una simulazione di dinamica molecolare, la funzione di distribuzione di coppia (o funzione di distribuzione radiale) $g(r)$ fornisce informazioni su come varia la densità di particelle in funzione della distanza r da una particella assegnata. Nel caso, come quello considerato, in cui non siano presenti forze esterne, la $g(r)$ risulta indipendente dalla particella considerata ed è proporzionale alla probabilità di trovare due particelle qualsiasi a distanza r una dall'altra.
Dalla meccanica statistica si sa che, date N particelle, la densità di probabilità della configurazione con una qualsiasi particella in $\mathbf{r}_1$, una qualsiasi in $\mathbf{r}_2$, e così via, fino a una qualsiasi in $\mathbf{r}_n$, indipendentemente dalle rimanenti $N - n$, è data da

$$\rho^{(n)}(\mathbf{r}_1, \ldots, \mathbf{r}_n) = \frac{N!}{(N-n)!} P^{(n)}(\mathbf{r}_1, \ldots, \mathbf{r}_n) \,, \tag{7}$$

dove

$$P^{(n)}(\mathbf{r}_1, \ldots, \mathbf{r}_n) = \frac{\int f(\mathbf{r}_1, \ldots, \mathbf{r}_n) \, \mathrm{d}\mathbf{r}_{n+1} \ldots \mathrm{d}\mathbf{r}_N}{\int f(\mathbf{r}_1, \ldots, \mathbf{r}_n) \, \mathrm{d}\mathbf{r}_1 \ldots \mathrm{d}\mathbf{r}_N} \tag{8}$$

è la marginalizzazione rispetto alle prime n variabili della densità di probabilità della configurazione con la particella 1 in $\mathbf{r}_1$, la 2 in $\mathbf{r}_2$, e così via. La forma di f dipende dall'*ensemble* statistico che si sta considerando; per esempio, nell'*ensemble* canonico, si ha

$$f(\mathbf{r}_1, \ldots, \mathbf{r}_n) = \exp\big(- \beta U(\mathbf{r}_1, \ldots, \mathbf{r}_n)\big) \,,$$

dove $U(\mathbf{r}_1, \ldots, \mathbf{r}_n)$ è l'energia potenziale del sistema, $\beta \equiv (k_B T)^{-1}$.
Per un fluido omogeneo, si ha $\rho^{(1)}(\mathbf{r}_1) = \rho \equiv N/V$. In questo contesto, si definisce la funzione di correlazione tra n particelle come

$$g^{(n)}(\mathbf{r}_1, \ldots, \mathbf{r}_n) \equiv \frac{\rho^{(n)}(\mathbf{r}_1, \ldots, \mathbf{r}_n)}{\rho^n} \,.$$

[1] Più precisamente è un'applicazione delle relazioni di Green-Kubo, che legano i coefficienti di trasporto γ agli integrali di funzioni di correlazione temporale. A tale proposito si veda ad esempio K. N. Dzhumagulova et al., *Velocity Autocorrelation Functions and Diffusion Coefficient of Dusty Component in Complex Plasmas*, Contrib. Plasma Phys./Volume 52, 2012.

Per le relazioni (7) e (8), si ha quindi

$$g^{(n)}(\mathbf{r}_1, \ldots, \mathbf{r}_n) = \frac{N!}{(N-n)!\,\rho^n} \frac{\int f(\mathbf{r}_1, \ldots, \mathbf{r}_n)\, \mathrm{d}\mathbf{r}_{n+1} \ldots \mathrm{d}\mathbf{r}_N}{\int f(\mathbf{r}_1, \ldots, \mathbf{r}_n)\, \mathrm{d}\mathbf{r}_1 \ldots \mathrm{d}\mathbf{r}_N} \,.$$

Per $n = 2$, si ottiene la funzione di distribuzione di coppia, che per un sistema omogeneo e isotropo può essere riscritta come funzione della sola distanza relativa $r = |\mathbf{r}_1 - \mathbf{r}_2|$. Sotto l'ipotesi ergodica, si dimostra facilmente che

$$g(r) \equiv g^{(2)}(\mathbf{r}_1,\, \mathbf{r}_2) = \frac{N-1}{4\pi r^2 \rho} \left\langle \delta\big(r - r_{12}(t)\big) \right\rangle_t ,$$

dove $r_{12}(t)$ è la distanza al tempo t tra le particelle 1 e 2 e la media è calcolata su un intervallo di tempo $\Delta t \to \infty$, in modo che $g(r)$ risulti indipendente dalle particelle considerate.

Per il calcolo numerico della $g(r)$, la delta di Dirac $\delta(r - r')$ deve essere sostituita con la seguente funzione:

$$\Delta(r - r') \equiv \begin{cases} \frac{1}{\mathrm{d}r} & \text{se } r' \in \left(r - \frac{\mathrm{d}r}{2},\, r + \frac{\mathrm{d}r}{2}\right) \\ 0 & \text{altrimenti} \end{cases}$$

con $\mathrm{d}r$ distanza sufficientemente piccola. Mediando inoltre su tutte le N particelle del sistema, la $g(r)$, in forma discretizzata, diventa

$$g(r) \to \frac{1}{4\pi r^2 \rho N} \sum_{i \neq j} \left\langle \Delta\big(r - r_{ij}(t)\big) \right\rangle_t .$$

In questa forma, $g(r)$ è dunque pari al conteggio delle distanze r_{ij} comprese nell'intervallo $(r - \mathrm{d}r/2,\, r + \mathrm{d}r/2)$, mediato nel tempo e diviso per la costante di normalizzazione $4\pi r^2 \mathrm{d}r \rho N$.
Come nota finale, osserviamo che $4\pi r^2 \mathrm{d}r \rho$ non è altro che il numero di particelle che si avrebbe in un guscio sferico di spessore infinitesimo $\mathrm{d}r$, a distanza r da una particella fissata, se la densità fosse ovunque uniforme. Se $\mathrm{d}r$ non è però realmente infinitesimo, il volume del guscio non risulta ben approssimato da $4\pi r^2 \mathrm{d}r$. Per ovviare a tale problema, una valida soluzione consiste nello scrivere il volume del guscio come

$$\frac{4}{3}\pi \left[\left(r + \frac{\mathrm{d}r}{2} \right)^3 - \left(r - \frac{\mathrm{d}r}{2} \right)^3 \right] .$$

1.5 Stima della pressione

Una quantità importante facilmente ricavabile con una simulazione di dinamica molecolare è la pressione.
Dalla meccanica statistica si ricava che una particella, identificata dal pedice i, con posizione $\mathbf{r}_i$ e sottoposta a una forza complessiva $\mathbf{F}_i$, avrà un'energia cinetica media

data da

$$\left\langle E_i^{kin} \right\rangle = -\frac{\langle \mathbf{F}_i \cdot \mathbf{r}_i \rangle}{2},$$

dove le medie, calcolate sullo spazio delle fasi, sotto l'ipotesi ergodica possono essere sostituite da medie temporali. Sfruttando il teorema di equipartizione e sommando sulle N particelle del sistema, si ottiene quindi

$$3Nk_BT = -\left\langle \sum_{i=1}^{N} \mathbf{F}_i \cdot \mathbf{r}_i \right\rangle,$$

dove il membro di destra è il valor medio del viriale calcolato sulle forze complessive agenti sulle particelle; comprende dunque un contributo dato dalle forze interne al sistema $\mathbf{F}_i^{int}$ e da quelle esterne $\mathbf{F}_i^{ext}$. Nel caso di un gas in una scatola di volume V, è intuitivo ricondurre l'unica forza esterna all'effetto contenitivo delle pareti e legarla quindi alla pressione P. Si vede facilmente che il valor medio del viriale calcolato limitatamente alle $\mathbf{F}_i^{ext}$ vale $-3PV$. Di conseguenza, si ha

$$3Nk_BT = 3PV - \left\langle \sum_{i=1}^{N} \mathbf{F}_i^{int} \cdot \mathbf{r}_i \right\rangle,$$

ovvero

$$P = \rho k_B T + \frac{1}{3V} \left\langle \sum_{i=1}^{N} \mathbf{F}_i^{int} \cdot \mathbf{r}_i \right\rangle. \tag{9}$$

La forza interna, al solito, è facilmente calcolabile conoscendo la forma del potenziale d'interazione tra le particelle $V(r)$:

$$\mathbf{F}_i^{int} = -\sum_{j \neq i} \left. \frac{\partial V(r)}{\partial r} \right|_{r_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}}.$$

Ora, per eliminare nella (9) la dipendenza dalla posizione del centro di massa del sistema, è sufficiente sfruttare il terzo principio della dinamica per arrivare al risultato finale

$$P = \rho k_B T - \frac{1}{3V} \left\langle \sum_{i=1}^{N} \sum_{j<i} \left. \frac{\partial V(r)}{\partial r} \right|_{r_{ij}} r_{ij} \right\rangle, \tag{10}$$

che risulta particolarmente comodo da valutare numericamente.
È interessante notare che il termine dovuto alle forze interne non è altro che una correzione alla nota equazione di stato del gas perfetto. La forma del potenziale fornisce informazioni utili a determinare se la correzione alla pressione del gas ideale sia positiva (per potenziali prevalentemente repulsivi) o negativa (per potenziali prevalentemente attrattivi).

2 Descrizione del codice

Al fine di simulare il comportamento del gas di Xenon nell'*ensemble* microcanonico, si è realizzato un programma in linguaggio C che implementa l'algoritmo Velocity-Verlet e il calcolo delle osservabili precedentemente menzionate, di cui viene valutata la media temporale. Per l'implementazione si è ritenuto opportuno considerare unità adimensionali così definite:

$$\tilde{r} \equiv \frac{r}{\sigma}\,, \qquad \tilde{E} \equiv \frac{E}{\varepsilon}\,, \qquad \tilde{T} \equiv \frac{k_B}{\varepsilon}\,T\,,$$
$$\tilde{t} \equiv \frac{1}{\sigma}\sqrt{\frac{\varepsilon}{m}}\,t\,, \qquad \tilde{\rho} \equiv \sigma^3\rho\,, \qquad \tilde{v} \equiv \sqrt{\frac{m}{\varepsilon}}\,v\,, \qquad \tilde{F} \equiv \frac{\sigma}{\varepsilon}\,F\,,$$

dove r, E, T, t, ρ, v, F sono rispettivamente distanza, energia, temperatura, tempo, densità, velocità e forza, e le lettere "tildate" rappresentano le corrispondenti quantità adimensionali. Di seguito, tutte le quantità "tildate" sono da ritenersi espresse in unità adimensionali.

Al fine di ottimizzare la gestione e l'esecuzione del codice, nel programma alcune variabili sono dichiarate globali. Tra queste, di particolare importanza, sono gli *array* bidimensionali `double r[N][3]` e `double F[N][3]`, che contengono rispettivamente le posizioni e le forze all'istante di tempo attuale (memorizzato nella variabile globale `double tempo`); in entrambi i casi, il primo indice indica la particella, il secondo la coordinata x, y o z. Per le velocità, invece, viene utilizzato un *array* tridimensionale `double v[N_v][N][3]` (con `N_v` numero di punti per cui sarà stimata la funzione di correlazione velocità-velocità), dove il primo indice identifica progressivamente le velocità calcolate ad istanti di tempo diversi. `v`, nella prima dimensione, è utilizzato come un *array* circolare; a tale scopo, l'indice che individua le velocità calcolate all'istante attuale è memorizzato nella variabile globale `double idx_v`. Quando `v` risulta pieno, il *set* di velocità più vecchio viene sostituito con il più recente, mantenendo memoria dell'ordine temporale delle velocità grazie a `idx_v`. Sono dichiarati globali anche l'*array* `double E[3]`, che, nelle tre posizioni, contiene nell'ordine l'energia cinetica, potenziale e totale del sistema all'istante attuale, e la variabile `double viriale`, che contiene il viriale calcolato all'istante attuale. Infine, gli *array* globali `double gr[N_bin]` (`N_bin = 500`) e `double Cv[N_v]`, se opportunamente normalizzati, contengono rispettivamente la funzione di distribuzione di coppia $g(r)$ e la funzione di correlazione velocità-velocità $C_{\mathbf{v}}(t)$ discretizzate.

Il programma, nella funzione principale `main()`, prevede, in sequenza, le seguenti operazioni:

1. generazione delle condizioni iniziali tramite la funzione `genera_cond_ini()`, che a sua volta richiama le funzioni `genera_r()` e `genera_v()`, adibite rispettivamente al posizionamento delle particelle su un reticolo cubico come riferito nella sezione 1.1.1 e alla generazione delle velocità iniziali mediante le formule di Box-Muller (1);

2. calcolo di energia cinetica, energia potenziale ed energia totale iniziali, mediante la funzione `calcola_energie()` (che richiama a sua volta `calcola_Ek()` e `calcola_Epot()` per il calcolo di energia cinetica e potenziale);

3. calcolo delle posizioni e delle forze sulle particelle dopo l'intervallo di tempo $\delta\tilde{t}$, mediante la funzione `primo_passo_verlet()` (che a sua volta richiama `calcola_forze()` per il calcolo delle forze), tenendo conto del fatto che la forza iniziale su ciascuna particella è nulla per la simmetria delle posizioni iniziali;

4. fase di equilibratura, durante la quale il sistema viene fatto evolvere secondo l'algoritmo Velocity-Verlet fino alla sua termalizzazione;

5. fase statistica, che prevede l'evoluzione del sistema mediante l'algoritmo Velocity-Verlet e, ad ogni iterazione, il calcolo delle osservabili di cui verrà valutata la media temporale;

6. calcolo di una stima del coefficiente di autodiffusione D_a;

7. normalizzazione della funzione di distribuzione di coppia e della funzione di correlazione velocità-velocità;

8. calcolo delle medie temporali delle osservabili e delle relative incertezze.

Più in dettaglio, nella fase di equilibratura, vengono richiamate ciclicamente in sequenza le funzioni `passo_verlet()` e `calcola_energie_2()`. La prima esegue un passo dell'algoritmo Velocity-Verlet, calcolando posizioni e velocità delle particelle al tempo $\tilde{t} + \delta\tilde{t}$ a partire da quelle al tempo $\tilde{t}$; inoltre, richiamando la funzione `calcola_forze_Epot_viriale()`, aggiorna le forze sulle particelle, l'energia potenziale e il viriale. La funzione `calcola_energie_2()`, invece, aggiorna l'energia cinetica e totale. Il ciclo termina dopo un numero di iterazioni sufficiente alla termalizzazione del sistema, stabilito a posteriori studiando l'andamento di energia cinetica e potenziale nel tempo. Proprio a tale scopo vengono calcolate le energie anche in questa fase.

Nella fase statistica, invece, oltre a `passo_verlet()` e `calcola_energie_2()`, vengono richiamate ciclicamente le funzioni `aggiorna_gr()` e `aggiorna_Cv()`, adibite rispettivamente all'aggiornamento degli *array* `gr` e `Cv`, i cui elementi sono in origine inizializzati a zero. Come accennato, le funzioni $g(r)$ e $C_{\mathbf{v}}(t)$ sono valutate numericamente in forma discretizzata.
In particolare, per il calcolo della funzione di distribuzione di coppia, l'intervallo spaziale $\left[0, \tilde{L}/2\right]$ è suddiviso in `N_bin = 500` *bin* di ampiezza $\mathrm{d}\tilde{r}$; la funzione `aggiorna_gr()` calcola la distanza tra ciascuna coppia di particelle, individua il *bin* corrispondente ed incrementa conseguentemente il corrispondente elemento dell'*array* `gr`. Terminata la simulazione, ciascun elemento `gr[i]` deve essere normalizzato, dividendolo per il numero di iterazioni del ciclo costituente la fase

statistica e per la costante di normalizzazione

$$k_i = \frac{4}{3}\pi \left\{ [(i+1)\mathrm{d}\tilde{r}]^3 - (i\mathrm{d}\tilde{r})^3 \right\} \tilde{\rho} N \, .$$

La funzione di correlazione velocità-velocità è invece calcolata sull'intervallo temporale $\left[0,\ (N_v - 1)\delta\tilde{t}\right]$ (con N_v pari a `N_v`) per tempi $\tilde{t} = 0,\ \delta\tilde{t},\ 2\,\delta\tilde{t},\ ...,\ (N_v - 1)\delta\tilde{t}$. La funzione `aggiorna_Cv()`, dato l'*array* `v` pieno, per ogni $i = 0,\ 1,\ ...,\ N_v - 1$, incrementa l'elemento `Cv[i]` della quantità $\sum_{j=1}^{N} \tilde{\mathbf{v}}_j\left(\tilde{t}\right) \cdot \tilde{\mathbf{v}}_j\left(\tilde{t} - i\delta\tilde{t}\right)$, dove $\tilde{t}$ è l'istante di tempo attuale. Terminata la simulazione, ciascun elemento `Cv[i]` deve essere diviso per il numero di *step* dell'algoritmo Velocity-Verlet considerati per il calcolo, per il numero N di particelle e per `Cv[0]`, in modo da ottenere la funzione di correlazione delle velocità normalizzata. Prima di dividere per `Cv[0]`, gli elementi di `Cv` vengono utilizzati per ottenere una stima del coefficiente di autodiffusione, calcolando numericamente l'integrale (6) mediante la regola di Simpson cubica, implementata dalla funzione `integrale(double*, double, int)`. L'integrale, da effettuarsi in teoria sull'intervallo d'integrazione $\left[0,\ \tilde{t}\right]$ con $\tilde{t} \to +\infty$, viene calcolato al variare di $\tilde{t}$, in modo da valutare la velocità con cui converge al valore limite D_a per $\tilde{t} \to +\infty$.
Inoltre, al fine di valutare i valori medi delle energie e del viriale, ad ogni iterazione del ciclo che costituisce quella che è stata chiamata fase statistica, vengono accumulate in apposite variabili le somme e le somme dei quadrati di energia cinetica, energia potenziale, energia totale e viriale; terminato il ciclo, tali quantità vengono utilizzate per ricavare i valori medi delle corrispondenti osservabili e le relative deviazioni standard. La misura del viriale si utilizza in particolare per ottenere una stima della pressione del sistema (con relativa incertezza), sfruttando la formula (10).

La funzione `passo_verlet()` implementante un passo dell'algoritmo Velocity-Verlet è responsabile anche dell'applicazione delle condizioni periodiche al contorno di Born-Von Karman riferite nella sezione 1. A tale scopo, a ciascuna componente di ogni nuova posizione calcolata viene applicata la seguente funzione:

```
1 double pbc(double x){
2     return x - L * rint(x / L);
3 }
```

dove `rint(double)` restituisce l'intero che meglio approssima l'argomento. La funzione `pbc(double)` viene utilizzata anche nel calcolo di forze, energia potenziale e viriale. In tali calcoli, infatti, fissata una particella, vengono considerate le interazioni con tutte e sole le particelle che distano da quella in esame per meno di $\tilde{L}/2$, assumendo trascurabili le interazioni rimanenti. A tale scopo, ad ogni componente delle distanze calcolate considerando le particelle nella scatola di simulazione, viene applicata la funzione `pbc(double)` e vengono poi comunque escluse le distanze che risultano ancora superiori ad $\tilde{L}/2$. La funzione `pbc(double)` è utilizzata in modo analogo anche in `aggiorna_gr()`.

Alcuni accorgimenti significativi che si sono adottati per rendere più efficiente il codice sono la sovrascrittura di ogni variabile contenente quantità di cui non è necessario tenere memoria; la particolare gestione dell'*array* 3D `v`, utilizzato, limitatamente alla prima dimensione, come *array* circolare; il calcolo di forze, energia potenziale e viriale mediante la sola funzione `calcola_forze_Epot_viriale()`, rendendosi necessarie per i tre calcoli in buona parte le medesime operazioni.
Inoltre, al fine di monitorare l'andamento della simulazione, ogni 100 passi dell'algoritmo Velocity-Verlet, vengono stampati a video i valori istantanei delle energie, che, con maggiore frequenza, vengono salvati anche in un file di testo. Al termine della simulazione, vengono stampati a video i valori medi delle osservabili considerate e la stima ottenuta del coefficiente di autodiffusione; vengono salvate su file anche la funzione di distribuzione di coppia e la funzione di correlazione delle velocità. I dati salvati vengono successivamente elaborati mediante l'ambiente MATLAB, con cui vengono tracciati alcuni grafici significativi.

Nell'esecuzione della simulazione, si è scelto di considerare una fase statistica di $N_s = 10^6$ *step* temporali, ciascuno di durata $\delta t = 0.01$ ps; per δt più grandi, infatti, si è constatato come la simulazione non rispecchi la reale dinamica del sistema, tendendo l'energia cinetica stimata a diventare arbitrariamente grande con il passare del tempo, anziché oscillare attorno ad un valore medio. Per la fase di equilibratura si è invece deciso a posteriori di considerare $N_e = 7000$ passi, dopo aver analizzato l'andamento di energia cinetica e potenziale nel tempo ottenuto con una prima esecuzione del codice. Si è fissato inoltre `N_v = 500`; il programma si è in seguito rieseguito per `N_v = 10000` (e $N_s = 10^5$ per ridurre il tempo di esecuzione) allo scopo di ottenere una misura di $C_v(t)$ a tempi più grandi, ricavando così una migliore stima del coefficiente di autodiffusione.

Il codice completo, compreso lo script MATLAB, è riportato in Appendice A.1.

3 Discussione dei risultati e confronto con la teoria

Come riferito nella sezione precedente, la durata della fase di equilibratura è stata scelta a posteriori. Il grafico seguente (Fig. 1) evidenzia la motivazione della scelta di far iniziare la fase statistica dopo 7000 *step* dall'inizio della simulazione.

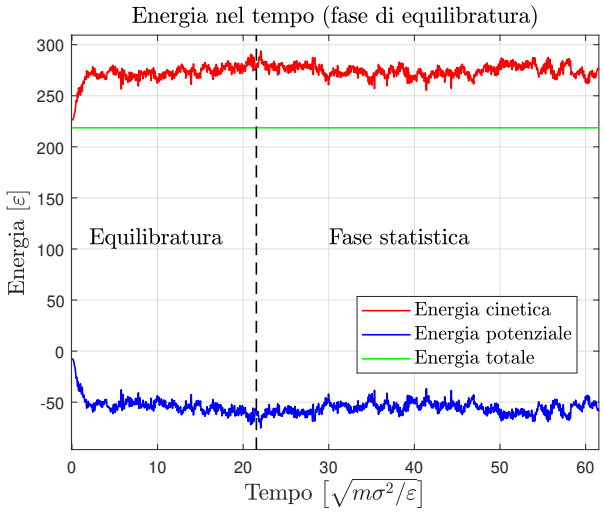

Figura 1: Grafico dei primi valori di energia cinetica, potenziale e totale del sistema in funzione del tempo, con transizione tra la fase di equilibratura e quella statistica.

Si osserva infatti come l'energia cinetica, nella fase di equilibratura considerata, cresca mediamente in modo monotono per assestarsi attorno a un valor medio costante nella successiva fase statistica. Un discorso analogo vale per l'energia potenziale, che presenta un andamento speculare rispetto a quello dell'energia cinetica.
Per quanto la fase di equilibratura possa essere interessante, la fase importante per la descrizione del sistema fisico è però quella statistica. Nel grafico seguente, a sinistra (Fig. 2), sono rappresentate energia cinetica, potenziale e totale in funzione del tempo per tutta la durata della fase statistica.

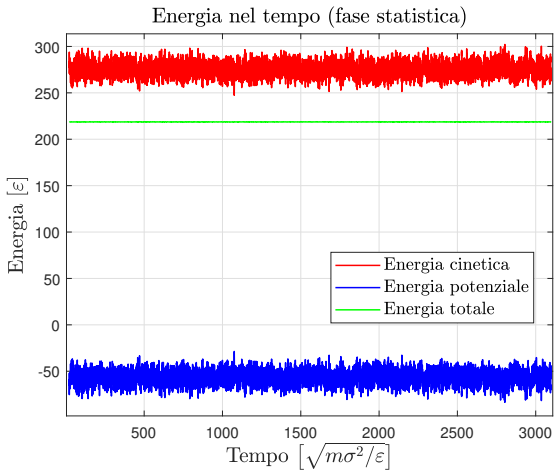

Figura 2: Energia potenziale, cinetica e totale in funzione del tempo per tutti i passi temporali della fase statistica.

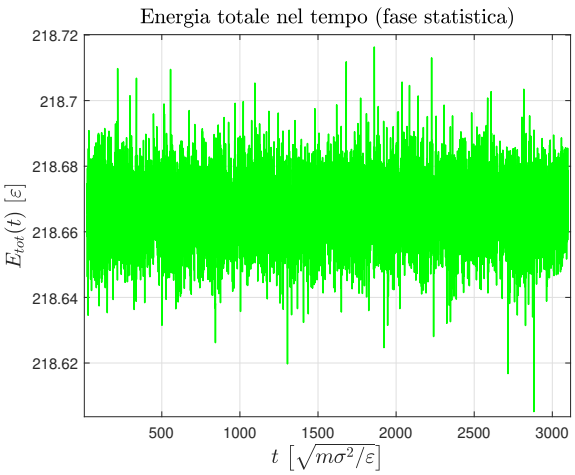

Figura 3: *Zoom* sulla sola energia totale E_{tot} in funzione del tempo t per l'intera fase statistica.

Un'osservazione degna di nota è che, sebbene energia cinetica e potenziale abbiano variazioni intorno al valor medio molto vistose (fino a $25\,\varepsilon$), l'energia totale presenta fluttuazioni relativamente piccole. Questo fatto si vede meglio nella figura di destra (Fig. 2), dove è rappresentata la sola energia totale. D'altronde non c'è da stupirsi: che l'energia totale sia approssimativamente costante, a meno di piccole fluttuazioni, è quanto ci aspettiamo lavorando nell'*ensemble* microcanonico.
Per completezza di seguito riportiamo i valori medi misurati per le energie con la relativa incertezza statistica (Tab. 1).

Tabella 1: Valori medi per energia potenziale, cinetica e totale, calcolati mediando sui 10^6 passi della sola fase statistica. Gli errori sono di origine statistica, dati dalla deviazione standard dei valori medi.

	$E\ [\varepsilon]$	$\Delta E\ [\varepsilon]$
Energia potenziale	-57.207	0.006
Energia cinetica	275.874	0.006
Energia totale	218.66728	0.00001

Il fatto di avere un'energia totale positiva, grazie all'energia cinetica circa 5 volte maggiore di quella potenziale (in valore assoluto), ci fa intuire che il sistema non sia molto legato, cosa che ci aspettiamo parlando di un gas nobile.
Un'ulteriore conferma si ha osservando la funzione di autocorrelazione delle velocità (Fig. 4), che, per il tempo $t = 500\,\delta t$, presenta un valore $C_{\mathbf{v}} \approx 0.5$. Ciò significa che, anche dopo intervalli di tempo relativamente grandi, la velocità di una particella risulta significativamente correlata con quella che aveva prima: dai dati si evince che una particella dopo 5 ps non ha mediamente subito urti sufficienti a perdere completamente memoria della sua velocità iniziale.

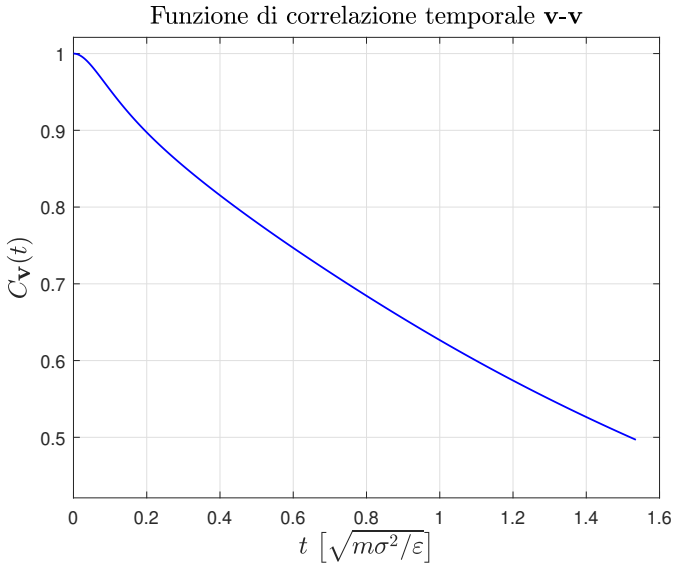

Figura 4: Funzione di autocorrelazione delle velocità (5), in funzione del tempo t, per `N_v = 500`.

Si apprezza meglio l'andamento della funzione di autocorrelazione delle velocità quando essa è calcolata per 10^4 *step* temporali (Fig. 5): la $C_{\mathbf{v}}(t)$ raggiunge lo

0 senza mai diventare negativa, quindi non presenta fenomeni di anticorrelazione tipici dei cristalli.

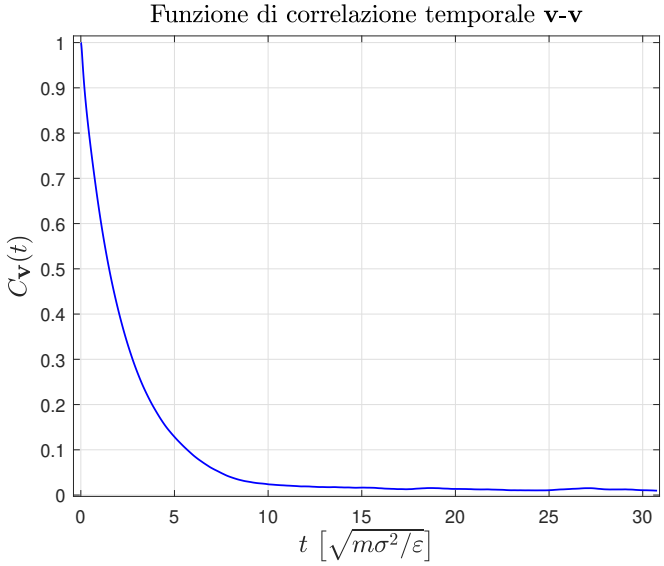

Figura 5: Funzione di autocorrelazione delle velocità (5), in funzione del tempo t, per `N_v = 10000`.

Come accennato nella sezione precedente, il calcolo della funzione di autocorrelazione per un numero così elevato di punti si è rivelato necessario al fine di dare una miglior stima del coefficiente di autodiffusione dello Xenon per la densità e la temperatura considerate. Dal grafico che segue (Fig. 6), è chiaro infatti come per $t \approx 1.5\sqrt{m\sigma^2/\varepsilon}$ il valore di D_a non sia affatto quello a cui converge l'integrale (6).

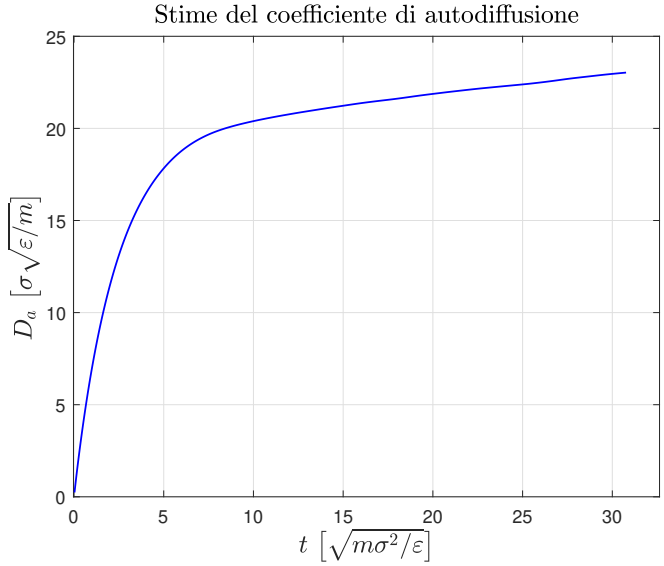

Figura 6: Stime del coefficiente di autodiffusione D_a in funzione del tempo t, estremo dell'intervallo d'integrazione $[0,\, t]$, ricavate sfruttando la relazione (6).

Il valore finale, ottenuto integrando su tutti gli $N_v = 10^4$ punti, è $D_a \approx 23.03\ \sigma\sqrt{\varepsilon/m} \approx 0.035\,\mathrm{mm}^2/\mathrm{s}$.

Di seguito (Fig. 7) è riportata la funzione di distribuzione di coppia $g(r)$, calcolata numericamente seguendo l'idea illustrata nella sezione 1.4. Come ci si aspetta dalla forma del potenziale di Lennard-Jones, per $r < \sigma$ la funzione è pressoché nulla: mediamente non ci sono particelle a queste distanze le une dalle altre a causa del termine repulsivo del potenziale.

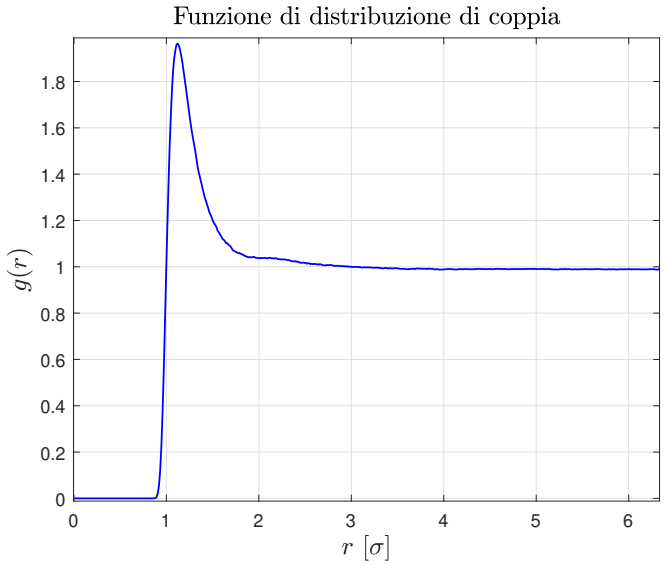

Figura 7: Funzione di distribuzione di coppia $g(r)$ calcolata fino a $r \approx 6.2\,\sigma$.

Il picco, che si ha ad $r \approx 1.12\,\sigma$, indica invece la distanza tra due particelle maggiormente probabile. L'andamento della $g(r)$ rispecchia in parte quello tipico di alcuni gas e alcuni liquidi. Per verificare che si tratti effettivamente di un gas, come supposto inizialmente, è sufficiente utilizzare la pressione stimata come illustrato nella sezione 1.5. Il valore medio, con incertezza di origine statistica, risulta $\tilde{P} = 0.06586(1)$ in unità adimensionali, corrispondente a $P = 3.4458(6)\,\text{MPa} = 34.007(6)\,\text{atm}$. A questo punto, è sufficiente verificare in quale punto del diagramma di stato (Fig. 8) cada il nostro fluido.

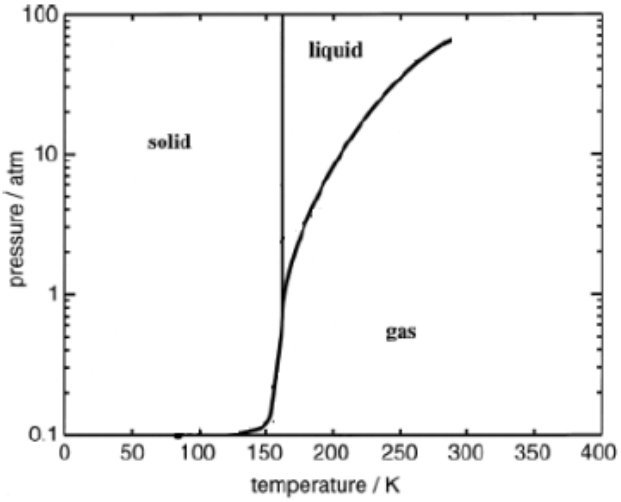

Figura 8: Diagramma di fase dello Xenon, da *Hyperpolarised Xenon in biology*, A. Cherubini e A. Bifone, Progress in NMR Spectroscopy, 2003.

Per ricavare la temperatura utilizziamo l'energia cinetica media, attraverso la relazione

$$E^{kin} = \frac{3}{2} N k_B T \,,$$

dalla quale otteniamo $T = 341.48(1)\,\mathrm{K}$. Risulta chiaro allora che, sebbene quasi a ridosso della zona supercritica, la fase del sistema studiato sia gassosa.

A Appendice

A.1 Codice

Di seguito sono riportati il codice della simulazione e lo *script* utilizzato per tracciare i grafici.

A.1.1 Codice in linguaggio C

```
// DINAMICA MOLECOLARE: fluido di Lennard-Jones

#include <stdio.h>
#include <stdlib.h>
#include <math.h>


// COSTANTI ---------------------------------
#define PI 3.14159265358979323846 // pi greco

// Costante di Boltzmann
#define kB 0.00008617333 // eV/K

// Parametri potenziale di Lennard-Jones
#define epsilon 0.02 // eV
#define sigma 3.94 // A (Angstrom)

// Massa Xenon
#define m 131.293 // u.m.a.

// Parametri simulazione
#define ro_dim 0.001 // Densita' [A^(-3)]
#define T_dim 300.0 // Temperatura [K]

// Step temporale per Velocity-Verlet
#define deltaT_dim 0.01 // ps

// Fattore di conversione per unita' di misura temporali
#define conv_tempo 0.0101805086 // 1 sqrt(u.m.a. * A^2 / eV) =
    (conv_tempo) ps

// Numero di particelle nella scatola di simulazione
#define N 125

#define N_eq 7000 // Numero di step per fase di equilibratura
#define N_stat 1000000 // Numero di step per fase statistica

```

```
37 /* Numero di passi precedenti per cui viene mantenuta in
      memoria la velocita' delle particelle (per il calcolo della
       funzione di correlazione delle velocita' C_v(t)) */
38 #define N_v 500
39
40 /* Numero di bin dell'istogramma per il calcolo della funzione
       di distribuzione di coppia g(r) */
41 #define N_bin 500
42
43 /* Numero di passi tra una stampa e video e l'altra dei valori
       istantanei dell'energia */
44 #define STEP_PRINT 100
45
46 /* Numero di passi tra una stampa su file e l'altra dei valori
       istantanei dell'energia */
47 #define STEP_SAVE 10
48
49
50 // PROTOTIPI DELLE FUNZIONI ---------------------------------
51 void genera_cond_ini(); // Genera le condizioni iniziali
52 void genera_r(); // Genera le posizioni iniziali delle
      particelle
53 void genera_v(); // Genera le velocita' iniziali delle
      particelle
54 void primo_passo_verlet(); // Esegue il primo passo dell'
      algoritmo Velocity-Verlet
55 void passo_verlet(); // Esegue un passo dell'algoritmo
      Velocity-Verlet
56 void calcola_forze(); // Aggiorna le forze sulle particelle
57 void calcola_forze_Epot_viriale(); // Aggiorna le forze, l'
      energia potenziale e il viriale
58 double pbc(double); // Applica le condizioni periodiche al
      bordo
59 double der_lj(double); // Calcola la derivata del potenziale
      di Lennard-Jones
60 double lj(double); // Calcola il potenziale di Lennard-Jones
61 void calcola_Ek(); // Aggiorna l'energia cinetica
62 void calcola_Epot(); // Aggiorna l'energia potenziale
63 void calcola_energie(); // Aggiorna energia cinetica,
      potenziale e totale
64 void calcola_energie_2(); // Aggiorna energia cinetica e
      totale
65 //void calcola_viriale(); // Aggiorna il viriale
66 void inizializza_gr(); // Inizializza i vettori "gr" e "norm"
67 void aggiorna_gr(); // Aggiorna il vettore "gr" per il calcolo
       della g(r)
```

```
68 void inizializza_Cv(); // Inizializza il vettore "Cv"
69 void aggiorna_Cv(); // Aggiorna il vettore "Cv" per il calcolo
       della C_v(t)
70 double integrale(double*, double, int); // Integrazione con
      regola di Simpson cubica
71
72
73 // VARIABILI GLOBALI --------------------------------
74 double n; // N^(1/3)
75 double ro; // Densita' adimensionale
76 double T; // Temperatura adimensionale
77 double deltaT; // Step temporale adimensionale
78 double L, L2; // Lato e meta' lato della scatola di
      simulazione
79 double deltaR; // Larghezza bin istogramma per g(r)
80
81 /* Posizioni attuali delle particelle:
82 r[i][j] con i = 0, 1, ..., N (numero di particelle);
83 j = 0, 1, 2 corrispondenti alle coordinate x, y, z */
84 double r[N][3];
85
86 double v[N_v][N][3]; // Velocita' delle particelle
87
88 /* Indice della prima dimensione di "v" corrispondente alle
      velocita' al tempo attuale */
89 int idx_v = 0;
90
91 double F[N][3]; // Forze sulle particelle
92
93 double tempo; // Istante di tempo attuale
94
95 /* "E[0]" Energia cinetica attuale
96    "E[1]" Energia potenziale attuale
97    "E[2]" Energia totale attuale */
98 double E[3];
99
100 double viriale; // Viriale attuale
101
102 /* Funzione di distribuzione di coppia g(r), se ciascun
      elemento e' diviso per il numero di passi di Velocity-
      Verlet considerati per il calcolo, e per le costanti di
      normalizzazione */
103 double gr[N_bin];
104
105 double norm[N_bin]; // Costanti di normalizzazione per la g(r)
106
```

```
107 /* Funzione di correlazione delle velocita' C_v(t), se ciascun
       elemento e' diviso per il numero di passi di Velocity-
       Verlet considerati per il calcolo, per il numero di
       particelle, e per "Cv[0]" */
108 double Cv[N_v];
109
110
111 // MAIN --------------------------------
112 int main(){
113     FILE *file;
114
115     // File per il salvataggio delle energie
116     file = fopen("C:\\Users\\Admin\\Documents\\MATLAB\\
           esercizio4_energie.txt", "w");
117
118     int i;
119     short j;
120     double dist, time;
121     double E_sum[3] = {0, 0, 0}; // Somma delle energie (
           cinetica [0], potenziale [1], totale [2]) a tempi
           diversi
122     double E2_sum[3] = {0, 0, 0}; // Somma dei quadrati delle
           energie a tempi diversi
123     double E_mean[3], E_std[3], E_var[3];
124     double viriale_sum = 0; // Somma del viriale a tempi
           diversi
125     double viriale2_sum = 0; // Somma del quadrato del viriale
           a tempi diversi
126     double viriale_mean, press_mean, press_std;
127     double D;
128
129     n = pow(N, 1./3.);
130     ro = ro_dim * sigma * sigma * sigma;
131     T = T_dim * kB / epsilon;
132     deltaT = deltaT_dim / conv_tempo * sqrt(epsilon / m) /
           sigma;
133     L = pow(N / ro, 1./3.);
134     L2 = L * 0.5;
135     deltaR = L2 / N_bin;
136
137     inizializza_Cv();
138     inizializza_gr();
139
140     genera_cond_ini();
141     calcola_energie();
142
```

```
143     // Stampo energie iniziali su file
144     fprintf(file, "%f %f %f %f\n", tempo, E[0], E[1], E[2]);
145 
146     printf("DINAMICA MOLECOLARE: fluido di Lennard-Jones\n\n")
            ;
147     printf("Energia cinetica iniziale: %f\n", E[0]);
148     printf("Energia potenziale iniziale: %f\n", E[1]);
149     printf("Energia totale iniziale: %f\n\n", E[2]);
150     printf("Inizio della fase di equilibratura\n\n");
151     //printf("Step\tE_k\t\tE_pot\t\tE_tot\n");
152 
153     primo_passo_verlet();
154 
155     // Fase di equilibratura
156     for(i = 1; i <= N_eq; i++){
157         passo_verlet();
158         calcola_energie_2();
159 
160         // Stampo su file
161         if(i % STEP_SAVE == 0)
162             fprintf(file, "%f %f %f %f\n", tempo, E[0], E[1],
                    E[2]);
163 
164         // Stampo a video
165         /*if(i % STEP_PRINT == 0)
166             printf("%d\t%f\t%f\t%f\n", i, E[0], E[1], E[2]);*/
167 
168         // Barra di caricamento
169         if(i % STEP_PRINT == 0)
170             printf("%c", 177);
171     }
172 
173     printf("\n\nFine della fase di equilibratura\n\n");
174     printf("Step\tE_k\t\tE_pot\t\tE_tot\n");
175 
176     // Fase statistica
177     for(i = 1; i <= N_stat; i++){
178         passo_verlet();
179 
180         /* Calcolo osservabili
181         (energia potenziale e viriale sono gia' stati
                calcolati in "passo_verlet()") */
182         calcola_energie_2();
183         aggiorna_gr();
184         if(i >= N_v)
185             aggiorna_Cv();
```

```
186
187         // Accumulo valori delle energie e del viriale
188         for(j = 0; j < 3; j++){
189             E_sum[j] += E[j];
190             E2_sum[j] += E[j] * E[j];
191         }
192         viriale_sum += viriale;
193         viriale2_sum += viriale * viriale;
194
195         // Stampo su file
196         if(i % STEP_SAVE == 0)
197             fprintf(file, "%f %f %f %f\n", tempo, E[0], E[1],
                    E[2]);
198
199         // Stampo a video
200         if(i % STEP_PRINT == 0)
201             printf("%d\t%f\t%f\t%f\n", i, E[0], E[1], E[2]);
202     }
203
204     fclose(file);
205
206     // File per il salvataggio della g(r)
207     file = fopen("C:\\Users\\Admin\\Documents\\MATLAB\\
           esercizio4_gr.txt", "w");
208
209     // Normalizzazione e salvataggio della g(r)
210     for(i = 0; i < N_bin; i++){
211         gr[i] /= norm[i] * N_stat;
212         dist = (i + 0.5) * deltaR;
213         fprintf(file, "%f %f\n", dist, gr[i]);
214     }
215
216     fclose(file);
217
218     // File per il salvataggio delle stime del coeff. di
           autodiffusione
219     file = fopen("C:\\Users\\Admin\\Documents\\MATLAB\\
           esercizio4_D.txt", "w");
220
221     // Normalizzazione della C_v(t) (senza dividere per "Cv
           [0]")
222     for(i = 0; i < N_v; i++)
223         Cv[i] /= N * (N_stat - N_v + 1);
224
225     // Calcolo e salvataggio di stime del coeff. di
           autodiffusione
```

```
226     for(i = 10; i <= N_v; i += 10){
227         D = 2 * integrale(&Cv[0], deltaT, i);
228         time = (i - 1) * deltaT;
229         fprintf(file, "%f %f\n", time, D);
230     }
231 
232     printf("\n\nCoefficiente di autodiffusione: %f\n", D);
233 
234     fclose(file);
235 
236     // File per il salvataggio della C_v(t)
237     file = fopen("C:\\Users\\Admin\\Documents\\MATLAB\\
            esercizio4_Cv.txt", "w");
238 
239     // Divisione per "Cv[0]" e salvataggio della C_v(t)
240     fprintf(file, "%f %f\n", 0., 1.);
241     for(i = 1; i < N_v; i++){
242         Cv[i] /= Cv[0];
243         time = i * deltaT;
244         fprintf(file, "%f %f\n", time, Cv[i]);
245     }
246 
247     fclose(file);
248 
249     // Calcolo medie temporali e incertezze delle energie
250     for(j = 0; j < 3; j++){
251         E_mean[j] = E_sum[j] / N_stat; // Media temporale
252         E_var[j] = E2_sum[j] / N_stat - E_mean[j] * E_mean[j];
                // Varianza della singola misura
253         E_std[j] = sqrt(E_var[j] / N_stat); // Deviazione
                standard della media
254     }
255 
256     // Calcolo media temporale e deviazione standard della
            pressione
257     viriale_mean = viriale_sum / N_stat;
258     press_mean = ro * T + viriale_mean / (3 * L * L * L);
259     press_std = sqrt((viriale2_sum / N_stat - viriale_mean *
            viriale_mean) / N_stat) / (3 * L * L * L);
260 
261     // Stampo a video medie temporali delle osservabili
262     printf("Energia cinetica media: %f +/- %f (varianza: %f)\n
            ", E_mean[0], E_std[0], E_var[0]);
263     printf("Energia potenziale media: %f +/- %f (varianza: %f)
            \n", E_mean[1], E_std[1], E_var[1]);
```

```
264     printf("Energia totale media: %f +/- %f (varianza: %f)\n",
            E_mean[2], E_std[2], E_var[2]);
265     printf("Pressione media: %f +/- %f\n", press_mean,
            press_std);
266
267     return 0;
268 }
269
270
271 // FUNZIONI ---------------------------------
272
273 // Genera le condizioni iniziali
274 void genera_cond_ini(){
275     genera_r();
276     genera_v();
277     tempo = 0;
278 }
279
280
281 /* Posiziona le particelle nella scatola di simulazione su un
        reticolo */
282 void genera_r(){
283     double dL = L / n, L0 = -L * 0.5;
284     int x, y, z, i = 0;
285
286     for(x = 0; x < n; x++){
287         for(y = 0; y < n; y++){
288             for(z = 0; z < n; z++){
289                 r[i][0] = L0 + dL * x;
290                 r[i][1] = L0 + dL * y;
291                 r[i][2] = L0 + dL * z;
292                 i++;
293             }
294         }
295     }
296 }
297
298
299 /* Genera le velocita' iniziali delle particelle, utilizzando
        le trasformazioni di Box-Muller */
300 void genera_v(){
301     int i;
302     double sig = sqrt(T); // k_B = 1, m = 1 in unita'
            adimensionali
303     double pi2 = PI * 2;
304     double x1, x2, x3, x4;
```

```
305
306     for(i = 0; i < N; i++){
307         x1 = (double)rand() / RAND_MAX;
308         x2 = (double)rand() / RAND_MAX;
309         x3 = (double)rand() / RAND_MAX;
310         x4 = (double)rand() / RAND_MAX;
311         v[idx_v][i][0] = sig * sqrt(-2 * log(1 - x1)) * cos(
                pi2 * x2);
312         v[idx_v][i][1] = sig * sqrt(-2 * log(1 - x2)) * sin(
                pi2 * x1);
313         v[idx_v][i][2] = sig * sqrt(-2 * log(1 - x3)) * sin(
                pi2 * x4);
314     }
315 }
316
317
318 /* Esegue il primo passo dell'algoritmo Velocity-Verlet
319 (la forza totale su ciascuna perticelle e' nulla inizialmente)
320 e calcola le forze dopo gli spostamenti iniziali */
321 void primo_passo_verlet(){
322     int i;
323     short j;
324
325     for(i = 0; i < N; i++){
326         for(j = 0; j < 3; j++){
327             r[i][j] = pbc(r[i][j] + v[idx_v][i][j] * deltaT);
328         }
329     }
330
331     calcola_forze();
332
333     tempo = deltaT;
334 }
335
336
337 /* Esegue un passo dell'algoritmo Velocity-Verlet,
338 e aggiorna forze, energia potenziale e viriale
339 (richiamando "calcola_forze_Epot_viriale()") */
340 void passo_verlet(){
341     int i;
342     short j;
343     double deltaT2 = deltaT * 0.5;
344     double deltaT22 = deltaT2 * deltaT;
345
346     int idx_v_prec = idx_v;
```

```
347     idx_v = (idx_v + 1) % N_v; // Aggiorno indice delle
            velocita'
348
349     for(i = 0; i < N; i++){
350         for(j = 0; j < 3; j++){
351             r[i][j] = pbc(r[i][j] + v[idx_v_prec][i][j] *
                    deltaT + deltaT22 * F[i][j]);
352             v[idx_v][i][j] = v[idx_v_prec][i][j] + deltaT2 * F
                    [i][j];
353         }
354     }
355
356     calcola_forze_Epot_viriale();
357
358     for(i = 0; i < N; i++){
359         for(j = 0; j < 3; j++)
360             v[idx_v][i][j] += deltaT2 * F[i][j];
361     }
362
363     tempo += deltaT;
364 }
365
366
367 // Aggiorna le forze sulle particelle
368 void calcola_forze(){
369     int i, j;
370     short k;
371     double dr[3];
372     double f, dr_mod, f_mod;
373
374     // Inizializzazione a zero degli elementi di "F"
375     for(i = 0; i < N; i++){
376         for(k = 0; k < 3; k++)
377             F[i][k] = 0;
378     }
379
380     for(i = 1; i < N; i++){
381         for(j = 0; j < i; j++){
382             // Calcolo distanza tra particelle i-esima e j-
                    esima
383             for(k = 0; k < 3; k++)
384                 dr[k] = pbc(r[i][k] - r[j][k]);
385             dr_mod = sqrt(dr[0] * dr[0] + dr[1] * dr[1] + dr
                    [2] * dr[2]);
386
```

```
387             if(dr_mod < L2){ // Considero solo le distanze
                    minori di L/2
388                 // Calcolo le forze
389                 f_mod = -der_lj(dr_mod) / dr_mod;
390                 for(k = 0; k < 3; k++){
391                     f = f_mod * dr[k];
392                     F[i][k] += f;
393                     F[j][k] -= f; // F_{ji} = -F_{ij}
394                 }
395             }
396         }
397     }
398 }
399
400
401 // Aggiorna le forze, l'energia potenziale e il viriale
402 void calcola_forze_Epot_viriale(){
403     int i, j;
404     short k;
405     double dr[3];
406     double f, dr2, dr_mod, f_mod, der_pot;
407
408     // Inizializzazione a zero di "F", "E[1]" e "viriale"
409     for(i = 0; i < N; i++){
410         for(k = 0; k < 3; k++)
411             F[i][k] = 0;
412     }
413     E[1] = 0;
414     viriale = 0;
415
416     for(i = 1; i < N; i++){
417         for(j = 0; j < i; j++){
418             // Calcolo distanza tra particelle i-esima e j-
                    esima
419             for(k = 0; k < 3; k++)
420                 dr[k] = pbc(r[i][k] - r[j][k]);
421             dr2 = dr[0] * dr[0] + dr[1] * dr[1] + dr[2] * dr
                    [2];
422             dr_mod = sqrt(dr2);
423
424             if(dr_mod < L2){ // Considero solo le distanze
                    minori di L/2
425                 der_pot = der_lj(dr_mod);
426                 f_mod = -der_pot / dr_mod;
427                 for(k = 0; k < 3; k++){
428                     f = f_mod * dr[k];
```

```
                    F[i][k] += f;
                    F[j][k] -= f; // F_{ji} = -F_{ij}
                }

                E[1] += lj(dr2);

                viriale -= der_pot * dr_mod;
            }
        }
    }
}


// Applica le condizioni periodiche al bordo
double pbc(double x){
    return x - L * rint(x / L);
}


/* Calcola la derivata del potenziale di Lennard-Jones, data
   la distanza "x" */
double der_lj(double x){
    return 24 * (- 2 / pow(x, 13) + 1 / pow(x, 7));
}


/* Calcola il potenziale di Lennard-Jones, dato il modulo
   quadro della distanza "x2" */
double lj(double x2){
    double v1 = x2 * x2 * x2;
    double v2 = v1 * v1;
    return 4 * (1 / v2 - 1 / v1);
}


// Aggiorna l'energia cinetica
void calcola_Ek(){
    int i;
    E[0] = 0;
    for(i = 0; i < N; i++)
        E[0] += (v[idx_v][i][0] * v[idx_v][i][0] + v[idx_v][i
           ][1] * v[idx_v][i][1] + v[idx_v][i][2] * v[idx_v][i
           ][2]) * 0.5;
}


```

```
471 // Aggiorna l'energia potenziale
472 void calcola_Epot(){
473     int i, j;
474     short k;
475     double dr[3];
476     double dr2, L22 = L2 * L2;
477 
478     E[1] = 0;
479 
480     for(i = 1; i < N; i++){
481         for(j = 0; j < i; j++){
482             // Calcolo distanza tra particelle i-esima e j-
                esima
483             for(k = 0; k < 3; k++)
484                 dr[k] = pbc(r[i][k] - r[j][k]);
485             dr2 = dr[0] * dr[0] + dr[1] * dr[1] + dr[2] * dr
                [2];
486 
487             if(dr2 < L22) // Considero solo le distanze minori
                di L/2
488                 E[1] += lj(dr2);
489         }
490     }
491 }
492 
493 
494 // Aggiorna energia cinetica, potenziale e totale
495 void calcola_energie(){
496     calcola_Ek();
497     calcola_Epot();
498     E[2] = E[0] + E[1];
499 }
500 
501 
502 /* Aggiorna energia cinetica e totale
503 (l'energia potenziale deve essere precedentemente aggiornata)
    */
504 void calcola_energie_2(){
505     calcola_Ek();
506     E[2] = E[0] + E[1];
507 }
508 
509 
510 // Aggiorna il viriale
511 /*void calcola_viriale(){
512     int i, j;
```

```
513     short k;
514     double dr[3], dr_mod;
515 
516     viriale = 0;
517 
518     for(i = 1; i < N; i++){
519         for(j = 0; j < i; j++){
520             // Calcolo distanza tra particelle i-esima e j-
                esima
521             for(k = 0; k < 3; k++)
522                 dr[k] = pbc(r[i][k] - r[j][k]);
523             dr_mod = sqrt(dr[0] * dr[0] + dr[1] * dr[1] + dr
                [2] * dr[2]);
524 
525             if(dr_mod < L2) // Considero solo le distanze
                minori di L/2
526                 viriale -= der_lj(dr_mod) * dr_mod;
527         }
528     }
529 }*/
530 
531 
532 // Inizializza i vettori "gr" e "norm"
533 void inizializza_gr(){
534     int i;
535     for(i = 0; i < N_bin; i++){
536         norm[i] = 4. / 3. * PI * (pow((i + 1) * deltaR, 3) -
              pow(i * deltaR, 3)) * ro * N;
537         gr[i] = 0;
538     }
539 }
540 
541 
542 // Aggiorna il vettore "gr" per il calcolo della g(r)
543 void aggiorna_gr(){
544     int i, j, idx;
545     double dr[3], dr_mod;
546     short k;
547 
548     for(i = 1; i < N; i++){
549         for(j = 0; j < i; j++){
550             // Calcolo distanza tra particelle i-esima e j-
                esima
551             for(k = 0; k < 3; k++)
552                 dr[k] = pbc(r[i][k] - r[j][k]);
```

```
553             dr_mod = sqrt(dr[0] * dr[0] + dr[1] * dr[1] + dr
                    [2] * dr[2]);
554
555             // Calcolo indice di "gr" corrispondente alla
                    distanza trovata
556             idx = (int)(dr_mod / deltaR);
557
558             // Aggiorno "gr"
559             if(idx < N_bin)
560                 gr[idx] += 2;
561         }
562     }
563 }
564
565
566 // Inizializza il vettore "Cv"
567 void inizializza_Cv(){
568     int i;
569     for(i = 0; i < N_v; i++)
570         Cv[i] = 0;
571 }
572
573
574 // Aggiorna il vettore "Cv" per il calcolo della C_v(t)
575 void aggiorna_Cv(){
576     int i, j, idx;
577
578     for(i = 0; i < N_v; i++){
579         idx = (idx_v - i + N_v) % N_v; // Aggiorno "idx" all'
                indice delle velocita' al tempo "tempo - i * deltaT
                "
580         for(j = 0; j < N; j++){
581             Cv[i] += v[idx_v][j][0] * v[idx][j][0] + v[idx_v][
                    j][1] * v[idx][j][1] + v[idx_v][j][2] * v[idx][
                    j][2];
582         }
583     }
584 }
585
586
587 /* Integrazione con regola di Simpson cubica
588 "*f" e' un puntatore al primo elemento di un vettore
    contenente la funzione da integrare valutata in "numPunti"
    punti consecutivi distanziati uno dall'altro di un
    intervallo "h". Viene restituito l'integrale della funzione
     sull'intervallo coperto dal vettore il cui primo elemento
```

```
    e' puntato da "*f", stimato mediante la regola di Simpson
    cubica. */
double integrale(double *f, double h, int numPunti){
    double integ = (*f) + (*(f+numPunti-1));
    int i;

    for(i = 1; i < numPunti-1; i++){
        f++;
        if(i%2 == 0){
            integ += (*f) * 2;
        }
        else{
            integ += (*f) * 4;
        }
    }

    return integ * h / 3;
}
```

A.1.2 Script MATLAB per i grafici

```
close all;
format long;


%% Energie
% Parametri (da impostare come nel programma in C)
N_eq = 7000;
STEP_SAVE = 10;
deltaT = 0.003077;

% Per plot
N1 = N_eq / STEP_SAVE + 1;
N2 = 2000;

% Carico dati
dati = importdata('esercizio4_energie.txt');

% Dati fase statistica
t = dati(N1:end, 1);
Ek = dati(N1:end, 2);
Epot = dati(N1:end, 3);
Etot = dati(N1:end, 4);
```

```
23
24 % Grafico energie (fase statistica)
25 figure();
26 grid on;
27 hold on;
28 box on;
29 pl = plot(t, Ek, 'r');
30 set(pl, 'LineWidth', 1);
31 pl = plot(t, Epot, 'b');
32 set(pl, 'LineWidth', 1);
33 pl = plot(t, Etot, 'g');
34 set(pl, 'LineWidth', 1);
35 tt = title('Energia nel tempo (fase statistica)', 'Interpreter
   ', 'latex');
36 xx = xlabel('Tempo $\big[\sqrt{m \sigma^2 / \varepsilon}\big]$
   ', 'Interpreter', 'latex');
37 yy = ylabel('Energia [$\varepsilon$]', 'Interpreter', 'latex')
   ;
38 ll = legend('Energia cinetica', 'Energia potenziale', 'Energia
    totale');
39 set(xx, 'FontSize', 14);
40 set(yy, 'FontSize', 14);
41 set(tt, 'FontSize', 14);
42 set(ll, 'FontSize', 12);
43 set(ll, 'Location', 'SouthWest');
44 set(ll, 'Interpreter', 'latex');
45
46 % Grafico energia totale (fase statistica)
47 figure();
48 grid on;
49 hold on;
50 box on;
51 pl = plot(t, Etot, 'g');
52 set(pl, 'LineWidth', 1);
53 tt = title('Energia totale nel tempo (fase statistica)', '
   Interpreter', 'latex');
54 xx = xlabel('$t$ $\big[\sqrt{m \sigma^2 / \varepsilon}\big]$',
    'Interpreter', 'latex');
55 yy = ylabel('$E_{tot}(t)$ [$\varepsilon$]', 'Interpreter', '
   latex');
56 set(xx, 'FontSize', 14);
57 set(yy, 'FontSize', 14);
58 set(tt, 'FontSize', 14);
59
60 % Dati fase di equilibratura e inizio fase statistica
61 t = dati(1:N2, 1);
```

```
Ek = dati(1:N2, 2);
Epot = dati(1:N2, 3);
Etot = dati(1:N2, 4);

% Grafico energie (fase di equilibratura)
figure();
grid on;
hold on;
box on;
pl = plot(t, Ek, 'r');
set(pl, 'LineWidth', 1);
pl = plot(t, Epot, 'b');
set(pl, 'LineWidth', 1);
pl = plot(t, Etot, 'g');
set(pl, 'LineWidth', 1);
pl = plot([(N_eq + 1) * deltaT, (N_eq + 1) * deltaT], [-100,
    310], 'k--');
set(pl, 'LineWidth', 1);
tt = title('Energia nel tempo (fase di equilibratura)', '
    Interpreter', 'latex');
xx = xlabel('Tempo $\big[\sqrt{m \sigma^2 / \varepsilon}\big]$
    ', 'Interpreter', 'latex');
yy = ylabel('Energia [$\varepsilon$]', 'Interpreter', 'latex')
    ;
ll = legend('Energia cinetica', 'Energia potenziale', 'Energia
     totale');
set(xx, 'FontSize', 14);
set(yy, 'FontSize', 14);
set(tt, 'FontSize', 14);
set(ll, 'FontSize', 12);
set(ll, 'Location', 'SouthWest');
set(ll, 'Interpreter', 'latex');
testo = text(2, 110, 'Equilibratura', 'Interpreter', 'latex');
set(testo, 'FontSize', 14);
testo = text(30, 110, 'Fase statistica', 'Interpreter', 'latex
    ');
set(testo, 'FontSize', 14);


%% Funzione di distribuzione di coppia g(r)
% Carico dati
dati = importdata('esercizio4_gr.txt');
r = dati(:, 1);
g = dati(:, 2);

% Grafico
```

```
figure();
grid on;
hold on;
box on;
pl = plot(r, g, 'b');
set(pl, 'LineWidth', 1);
tt = title('Funzione di distribuzione di coppia', 'Interpreter', 'latex');
xx = xlabel('$r$ [$\sigma$]', 'Interpreter', 'latex');
yy = ylabel('$g(r)$', 'Interpreter', 'latex');
set(xx, 'FontSize', 14);
set(yy, 'FontSize', 14);
set(tt, 'FontSize', 14);


%% Funzione di correlazione temporale velocita'-velocita' C_v(t)
% Carico dati
dati = importdata('esercizio4_Cv.txt');
dt = dati(:, 1);
Cv = dati(:, 2);

% Grafico
figure();
grid on;
hold on;
box on;
pl = plot(dt, Cv, 'b');
set(pl, 'LineWidth', 1);
tt = title('Funzione di correlazione temporale \textbf{v}-\textbf{v}', 'Interpreter', 'latex');
xx = xlabel('$t$ $\big[\sqrt{m \sigma^2 / \varepsilon}\big]$', 'Interpreter', 'latex');
yy = ylabel('$C_{\textbf{v}}(t)$', 'Interpreter', 'latex');
set(xx, 'FontSize', 14);
set(yy, 'FontSize', 14);
set(tt, 'FontSize', 14);
ylim([0, 1.1]);


%% Coefficiente di autodiffusione D_a
% Carico dati
dati = importdata('esercizio4_D.txt');
t = dati(:, 1);
D = dati(:, 2);

```

```
144 % Grafico
145 figure();
146 grid on;
147 hold on;
148 box on;
149 pl = plot(t, D, 'b');
150 set(pl, 'LineWidth', 1);
151 tt = title('Stime del coefficiente di autodiffusione', '
    Interpreter', 'latex');
152 xx = xlabel('$t$ $\big[\sqrt{m \sigma^2 / \varepsilon}\big]$',
     'Interpreter', 'latex');
153 yy = ylabel('$D_a$ $\big[\sigma \sqrt{\varepsilon / m}\big]$',
     'Interpreter', 'latex');
154 set(xx, 'FontSize', 14);
155 set(yy, 'FontSize', 14);
156 set(tt, 'FontSize', 14);
```