Corso di Fisica Computazionale

- Quinta Esercitazione -

SIMULAZIONE MONTE CARLO DI UN GAS DI ARGON

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

1 Simulazione Monte Carlo **1**

1.1 Posizioni iniziali 3

1.2 Algoritmo di Metropolis 3

1.3 Autocorrelazioni 6

1.4 Stima della pressione 8

1.5 Stima della capacità termica a volume costante 9

2 Descrizione del codice **10**

3 Discussione dei risultati e confronto con la teoria **14**

A Grafici delle autocorrelazioni **19**

B Capacità termica **21**

C Codice in linguaggio C **22**

Abstract

In questa esercitazione viene implementata una simulazione Monte Carlo basata sull'algoritmo di Metropolis, al fine di stimare la pressione e la capacità termica a volume costante di un gas di Argon nell'*ensemble* canonico. Le due quantità sono calcolate al variare della temperatura e della densità del gas, con l'obiettivo in particolare di ottenere una stima qualitativa della temperatura critica 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. In particolare, vengono confrontate le isoterme ricavate numericamente con l'andamento atteso nel limite di basse densità per cui vale l'approssimazione di gas ideale.

1 Simulazione Monte Carlo

Un gas classico di N particelle di massa m, in equilibrio con un termostato alla temperatura T fissata, è descritto da un'Hamiltoniana del tipo

$$H(\mathbf{r}_1,\, ...,\, \mathbf{r}_N,\, \mathbf{p}_1,\, ...,\, \mathbf{p}_N) = \sum_{i=1}^{N} \frac{p_i^2}{2m} + V(\mathbf{r}_1,\, \mathbf{r}_2,\, ...,\, \mathbf{r}_N)\,,$$

dove $\mathbf{r}_1$, $\mathbf{r}_2$, ..., $\mathbf{r}_N$ e $\mathbf{p}_1$, $\mathbf{p}_2$, ..., $\mathbf{p}_N$ sono rispettivamente le posizioni e i momenti delle N particelle. Nella nostra trattazione, le interazioni tra le particelle vengono modellizzate mediante 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. In questo modo l'energia potenziale del sistema diventa

$$V(\mathbf{r}_1,\, \mathbf{r}_2,\, ...,\, \mathbf{r}_N) = \sum_{i \neq j} v(r_{ij})\,,$$

dove $r_{ij} \equiv |\mathbf{r}_i - \mathbf{r}_j|$.

Lavorando nell'*ensemble* canonico, la distribuzione delle velocità delle particelle costituenti il gas è data dalla nota distribuzione di Maxwell-Boltzmann; la densità di probabilità di avere una data configurazione $(\mathbf{r}_1,\, ...,\, \mathbf{r}_N)$ delle posizioni delle stesse è invece data dalla distribuzione di Boltzmann:

$$p(\mathbf{r}_1,\, ...,\, \mathbf{r}_N) = \frac{e^{-\beta V(\mathbf{r}_1,\, ...,\, \mathbf{r}_N)}}{\int e^{-\beta V(\mathbf{r}_1,\, ...,\, \mathbf{r}_N)}\, \mathrm{d}\mathbf{r}_1 \ldots \mathrm{d}\mathbf{r}_N}\,, \tag{1}$$

dove $\beta \equiv (k_B T)^{-1}$. L'obiettivo di questa esercitazione è campionare configurazioni con tale densità di probabilità, al fine di calcolare i valori medi sull'*ensemble* della

pressione P e della capacità termica a volume costante C_V per valori di temperatura e di densità fissati. Per raggiungere lo scopo si implementa una simulazione Monte Carlo basata sull'algoritmo di Metropolis, che verrà discusso dettagliatamente nella sezione 1.2.

In generale, la giustificazione dei metodi Monte Carlo è data dal teorema del limite centrale, secondo il quale, date M variabili aleatorie X_1, X_2, ..., X_M indipendenti ed identicamente distribuite con densità di probabilità $P(X_i)$ e un'arbitraria funzione $F(X_i)$, la variabile stocastica

$$S_M(F) \equiv \frac{1}{M} \sum_{i=1}^{M} F(X_i)$$

è distribuita con densità di probabilità $P_M(S_M)$ tale che

$$\lim_{M \to \infty} P_M(S_M) = \frac{1}{\sqrt{2\pi\sigma_M^2(F)}} \exp\left(- \frac{(S_M - \langle F \rangle)^2}{2\sigma_M^2(F)} \right),$$

dove si è posto

$$\langle F \rangle \equiv \int P(X)F(X)\mathrm{d}X\,, \qquad \langle F^2 \rangle \equiv \int P(X)F^2(X)\mathrm{d}X$$

e

$$\sigma_M^2(F) \equiv \frac{1}{M}\left(\langle F^2 \rangle - \langle F \rangle^2\right).$$

Di conseguenza, si possono stimare il valore medio e la deviazione standard di $F(X)$ come segue:

$$\langle F \rangle \simeq S_M(F)\,, \qquad \sigma_M \simeq \sqrt{\frac{1}{M-1}\left[S_M(F^2) - S_M^2(F)\right]}\,.$$

Grazie a tale risultato, se siamo in grado di campionare la densità di probabilità $P(X)$, possiamo dunque ottenere una stima del valore medio di $F(X)$ e del suo errore statistico, che decresce con il numero M di campioni considerati come $\frac{1}{\sqrt{M}}$, indipendentemente dalla dimensionalità del problema.

Nell'implementazione della simulazione, al fine di evitare complicazioni dovute alla presenza di pareti, consideriamo il gas come 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 immagina essere periodicamente ripetuta nello spazio. La scatola di simulazione viene 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 dal bordo opposto della scatola, nel punto individuato dalla direzione dello spostamento della prima.

Volendo descrivere un gas di Argon, nella simulazione si fissano le seguenti scelte per i parametri descriventi il sistema:

$$N = 125, \qquad m = 39.948\,\text{uma},$$
$$\varepsilon = 0.01\,\text{eV}, \qquad \sigma = 3.405 \cdot 10^{-10}\,\text{m},$$

dove m è la massa di un atomo di Argon. La simulazione viene quindi eseguita per diverse temperature T e densità ρ.

1.1 Posizioni iniziali

Per poter dare inizio alla simulazione, è necessario innanzitutto scegliere le posizioni iniziali delle N particelle. Per evitare problemi dovuti alla divergenza del potenziale per distanze $r \to 0$, si sceglie di posizionare le particelle su un reticolo cubico. In particolare, la scatola di simulazione 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*).

1.2 Algoritmo di Metropolis

Al fine di campionare configurazioni $R_i \equiv (\mathbf{r}_1^i,\, ...,\, \mathbf{r}_N^i)$ distribuite secondo la distribuzione di Boltzmann (1), si utilizza il cosiddetto *algoritmo di Metropolis-Hastings*, conosciuto anche semplicemente come *algoritmo di Metropolis*. Per la sua formalizzazione, è necessario introdurre una successione di variabili stocastiche nota come *catena di Markov*.

Assumiamo che all'inizio il sistema si trovi in una configurazione R_0 campionata da un'arbitraria distribuzione P_0. Assumiamo inoltre che, data la configurazione

R_i campionata dalla distribuzione P_i al passo i-esimo, la densità di probabilità di trovare il sistema al passo successivo in una configurazione R_{i+1} sia data da

$$P_{i+1}(R_{i+1}) = \int T_i(R_{i+1} \leftarrow R_i) P_i(R_i) \mathrm{d}R_i \,,$$

dove $T_i(R_{i+1} \leftarrow R_i)$ è detta matrice di transizione. Per comodità, introduciamo l'operatore integrale $\hat{T}_i$ tale che

$$P_{i+1}(R_{i+1}) = \hat{T}_i P_i(R_i) \,.$$

Assumiamo che $\hat{T}_i$ non dipenda dall'indice i, ovvero che valga $\hat{T}_i \equiv \hat{T}$ per ogni i. In questo modo, otteniamo una cosiddetta catena di Markov stazionaria. Sotto queste ipotesi, vale

$$P_i(R_i) = \hat{T} P_{i-1}(R_{i-1}) = \hat{T}^k P_0(R_0) \,,$$

ovvero tutti gli elementi della catena sono determinati solamente da $P_0(R_0)$ e $\hat{T}$. Si vede facilmente che, se esiste una densità di probabilità limite $P_\infty(R)$ verso cui converge la catena, allora questa deve essere un autostato dell'operatore $\hat{T}$ con autovalore 1:

$$\hat{T} P_\infty(R) = P_\infty(R) \,.$$

Si può dimostrare che una condizione sufficiente affinché il limite $P_\infty(R)$ esista e sia unico è che la catena di Markov sia tale che ogni configurazione permessa possa essere raggiunta da qualsiasi altra attraverso una sequenza finita di transizioni e non ci sia la possibilità di cadere in sequenze cicliche delle medesime configurazioni. Dunque, sotto tali ipotesi, se costruiamo un operatore di transizione $\hat{T}$ che ha come autostato una data distribuzione $P_\infty(R)$, la ripetuta applicazione di $\hat{T}$ a partire da un'arbitraria configurazione iniziale R_0 ci permette di produrre una catena i cui elementi, nel limite $i \to \infty$, sono distribuiti secondo $P_\infty(R)$.

Una volta raggiunta la distribuzione di probabilità limite, affinché il sistema preservi il suo stato di equilibrio, deve essere soddisfatta la seguente condizione:

$$\int T(R_{i+1} \leftarrow R_i) P_\infty(R_i) \mathrm{d}R_i = \int T(R_i \leftarrow R_{i+1}) P_\infty(R_{i+1}) \mathrm{d}R_{i+1} \,.$$

Condizione sufficiente affinché tale uguaglianza sia verificata è che valga la cosiddetta condizione di bilancio dettagliato, ovvero l'uguaglianza locale degli integrandi:

$$T(R_{i+1} \leftarrow R_i) P_\infty(R_i) = T(R_i \leftarrow R_{i+1}) P_\infty(R_{i+1}) \,,$$

da cui si ottiene

$$\frac{T(R_{i+1} \leftarrow R_i)}{T(R_i \leftarrow R_{i+1})} = \frac{P_\infty(R_{i+1})}{P_\infty(R_i)} \,. \tag{2}$$

Siamo liberi di scrivere la matrice di transizione $T(R_{i+1} \leftarrow R_i)$ come

$$T(R_{i+1} \leftarrow R_i) \equiv W(R_{i+1} \leftarrow R_i) A(R_{i+1} \leftarrow R_i) \,,$$

dove $W(R_{i+1} \leftarrow R_i)$ è una densità di probabilità arbitraria che siamo in grado di campionare e $A(R_{i+1} \leftarrow R_i)$ è da determinare. Così facendo, la condizione (2) diventa

$$\frac{A(R_{i+1} \leftarrow R_i)}{A(R_i \leftarrow R_{i+1})} = \frac{P_\infty(R_{i+1})W(R_i \leftarrow R_{i+1})}{P_\infty(R_i)W(R_{i+1} \leftarrow R_i)} .$$

Una possibile scelta di $A(R_{i+1} \leftarrow R_i)$ che soddisfa quest'ultima condizione è

$$A(R_{i+1} \leftarrow R_i) = \min\left(\frac{P_\infty(R_{i+1})W(R_i \leftarrow R_{i+1})}{P_\infty(R_i)W(R_{i+1} \leftarrow R_i)}, 1\right) .$$

Quest'ultima è la *probabilità di accettazione* della configurazione R_{i+1} campionata a partire da R_i secondo la densità di probabilità fissata $W(R_{i+1} \leftarrow R_i)$. Nel caso particolare in cui valga $W(R_{i+1} \leftarrow R_i) = W(R_i \leftarrow R_{i+1})$, si ottiene semplicemente

$$A(R_{i+1} \leftarrow R_i) = \min\left(\frac{P_\infty(R_{i+1})}{P_\infty(R_i)}, 1\right) . \tag{3}$$

Nel nostro caso, la distribuzione di probabilità limite $P_\infty(R)$ che vogliamo campionare è data dalla distribuzione di Boltzmann (1). Di conseguenza, la probabilità di accettazione diventa

$$A(R_{i+1} \leftarrow R_i) = \min\left(\frac{e^{-\beta V(R_{i+1})}}{e^{-\beta V(R_i)}}, 1\right) = \min\left(e^{-\beta(V(R_{i+1})-V(R_i))}, 1\right) . \tag{4}$$

A questo punto abbiamo tutti gli ingredienti per descrivere l'algoritmo di Metropolis. Si inizia scegliendo una configurazione iniziale R_0 delle N particelle. Ad ogni passo, data la configurazione $R_i \equiv R$, si genera una configurazione R' secondo la densità di probabilità $W(R' \leftarrow R)$, che, per semplicità, possiamo scegliere uniforme:

$$W(R' \leftarrow R) \equiv \begin{cases} \frac{1}{\Delta^{3N}} & \text{se } |r'_{k,\alpha} - r_{k,\alpha}| < \frac{\Delta}{2} \quad \forall\, k = 1,\, 2,\, ...,\, N,\ \alpha = x,\, y,\, z \\ 0 & \text{altrimenti} \end{cases}$$

dove Δ è un parametro fissato. Notiamo che, con questa definizione, vale $W(R' \leftarrow R) = W(R \leftarrow R')$. La generazione della configurazione R' può essere implementata mediante l'estrazione di numeri casuali $\xi_{k,\alpha}$, distribuiti uniformemente in $[0, 1)$:

$$r'_{k,\alpha} = r_{k,\alpha} + \Delta\left(\xi_{k,\alpha} - \frac{1}{2}\right) \quad \forall\, k = 1,\, 2,\, ...,\, N,\ \alpha = x,\, y,\, z.$$

La configurazione R' così generata deve essere accettata con probabilità

$$A(R' \leftarrow R) = \min\left(e^{-\beta(V(R')-V(R))}, 1\right) .$$

Pertanto, per stabilire la nuova configurazione R_{i+1}, una volta calcolata la probabilità di accettazione, estraiamo un nuovo numero casuale ζ, anch'esso distribuito uniformemente in $[0, 1)$. Se $\zeta < A(R' \leftarrow R)$ allora si pone $R_{i+1} = R'$, altrimenti si ha $R_{i+1} = R_i$.

Se vogliamo calcolare il valore medio e il corrispondente errore statistico di un'osservabile $O(\mathbf{r}_1, ..., \mathbf{r}_N)$, come ad esempio l'energia potenziale o il viriale, ad ogni passo dell'algoritmo, dobbiamo valutare O sulla configurazione R_i. Dopo M iterazioni, con M sufficientemente grande da ridurre a piacere gli errori statistici, si può stimare il valore medio di O mediante la quantità

$$\langle O \rangle \simeq \frac{1}{M} \sum_{i=1}^{M} O(R_i) \,,$$

affetta da un errore statistico pari a

$$\varDelta O \simeq \sqrt{\frac{1}{M-1} \left(\frac{1}{M} \sum_{i=1}^{M} O^2(R_i) - \langle O \rangle^2 \right)} \,.$$

Quest'ultima stima non è in realtà corretta, in quanto un'ipotesi cruciale del teorema del limite centrale è l'indipendenza delle variabili aleatorie campionate. Nel caso dell'algoritmo di Metropolis, basato su catene di Markov di variabili stocastiche, tale ipotesi viene a mancare. Di conseguenza, per una stima corretta degli errori statistici, è necessario tenere conto delle autocorrelazioni sussistenti tra le configurazioni campionate, come verrà discusso nella prossima sezione.
Inoltre, come verrà spiegato nella sezione 2, nel calcolo del valore medio di una data osservabile, è necessario considerare solamente le configurazioni generate dopo un'iniziale fase di equilibratura, necessaria a far perdere memoria al sistema delle condizioni iniziali e alla convergenza della distribuzione di probabilità campionata alla distribuzione di Boltzmann.

1.3 Autocorrelazioni

Come accennato nella sezione precedente, nel caso della successione di variabili stocastiche prodotta dall'algoritmo di Metropolis, venendo meno l'ipotesi di indipendenza, la tesi del teorema del limite centrale deve essere corretta tenendo conto delle autocorrelazioni sussistenti tra le variabili aleatorie campionate.
A tale scopo, consideriamo M configurazioni R_1, R_2, ..., R_M, campionate mediante l'algoritmo di Metropolis con l'obiettivo di valutare valore di aspettazione e varianza di una data osservabile O. A causa delle correlazioni, il valore medio di O dovrà essere calcolato come

$$\langle O \rangle = \frac{1}{M} \sum_{i=1}^{M} \int \tilde{P}(R_1, ..., R_N) O(R_i) \mathrm{d}R_1 ... \mathrm{d}R_N \,,$$

dove $\tilde{P}(R_1, ..., R_N)$ è la densità di probabilità congiunta delle M configurazioni, che non può essere banalmente fattorizzata nel prodotto delle rispettive densità di probabilità marginali. Essendo però, all'equilibrio, le densità di probabilità marginali

delle singole configurazioni tutte uguali a P_∞, risulta

$$\langle O \rangle = \frac{1}{M} \sum_{i=1}^{M} \int P_\infty(R_i) O(R_i) \mathrm{d}R_i = \int P_\infty(R) O(R) \mathrm{d}R \,.$$

Dunque, la stima del valore medio non va corretta. Ciò che invece varia a causa delle autocorrelazioni sussistenti tra passi successivi dell'algoritmo di Metropolis è l'errore statistico. Per stimarlo opportunamente è utile introdurre le funzioni di autocorrelazione.

Si definisce funzione di autocorrelazione dell'osservabile O la seguente quantità dipendente dal numero l di passi dell'algoritmo di Metropolis:

$$C_O(l) \equiv \frac{\langle O(R_i) O(R_{i+l}) \rangle - \langle O(R_i) \rangle^2}{\sigma_O^2} \,, \tag{5}$$

dove si è posto

$$\sigma_O^2 \equiv \left\langle O^2(R_i) \right\rangle - \langle O(R_i) \rangle^2$$

e le medie sono calcolate sugli *step* dell'algoritmo. Tipicamente, le funzioni di autocorrelazione così definite decadono esponenzialmente all'aumentare del numero di passi dell'algoritmo di Metropolis considerati:

$$C_O(l) \sim e^{-l/\tau} \,, \tag{6}$$

dove τ è detta *lunghezza di autocorrelazione.*

Per quanto visto, tenendo conto delle autocorrelazioni, la varianza del valore medio di O può essere scritta come

$$(\Delta O)^2 = \left\langle \frac{1}{M^2} \sum_{i=1}^{M} O(R_i) \sum_{j=1}^{M} O(R_j) \right\rangle - \langle O \rangle^2 \,. \tag{7}$$

Il primo termine può essere facilmente riscritto come

$$\left\langle \frac{1}{M^2} \sum_{i=1}^{M} O(R_i) \sum_{j=1}^{M} O(R_j) \right\rangle = \frac{1}{M^2} \sum_{i=1}^{M} \sum_{l=0}^{M-1} \langle O(R_i) O(R_{i+l}) \rangle \,. \tag{8}$$

Inoltre, per la definizione (5), vale

$$\langle O(R_i) O(R_{i+l}) \rangle = C_O(l) \sigma_O^2 + \langle O \rangle^2 \,. \tag{9}$$

Sostituendo le relazioni (8) e (9) nella (7), si ottiene

$$(\Delta O)^2 = \frac{1}{M} \sum_{l=0}^{M-1} C_O(l) \sigma_O^2 \,. \tag{10}$$

Approssimando la sommatoria con un integrale, per la (6), si può scrivere

$$\sum_{l=0}^{M-1} C_O(l) \approx \int_0^{+\infty} e^{-l/\tau} \mathrm{d}l = \tau\,.$$

Sostituendo quest'ultima relazione nella (10), si ottiene infine

$$\mathit{\Delta} O \approx \sqrt{\frac{\tau}{M}\sigma_O^2}\,. \tag{11}$$

Dunque, nel caso di catene di variabili stocastiche correlate con lunghezza di autocorrelazione τ, l'errore statistico sul valore medio cresce come $\sqrt{\tau}$. Nel caso di variabili indipendenti, invece, si ha

$$C_O(l) = \begin{cases} 1 & \text{se } l = 0 \\ 0 & \text{se } l \neq 0 \end{cases}$$

Di conseguenza, $\sum_{l=0}^{M-1} C_O(l) = 1$ e si ritrova il risultato del teorema del limite centrale.

1.4 Stima della pressione

Una quantità importante facilmente stimabile mediante una simulazione Monte Carlo 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 sono calcolate sullo spazio delle fasi. 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

$$P = \rho k_B T + \frac{1}{3V} \left\langle \sum_{i=1}^{N} \mathbf{F}_i^{int} \cdot \mathbf{r}_i \right\rangle . \tag{12}$$

La forza interna è facilmente calcolabile conoscendo la forma del potenziale d'interazione tra le particelle $v(r)$ (nel nostro caso, il potenziale di Lennard-Jones):

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

dove si è posto $\mathbf{r}_{ij} \equiv \mathbf{r}_i - \mathbf{r}_j$.
Ora, per eliminare nella (12) la dipendenza dalla posizione del centro di massa del sistema, è sufficiente sfruttare il terzo principio della dinamica per arrivare al risultato finale

$$\boxed{P = \rho k_B T - \frac{1}{3V} \left\langle \sum_{i=1}^{N} \sum_{j<i} \left. \frac{\partial v(r)}{\partial r} \right|_{r_{ij}} r_{ij} \right\rangle ,} \tag{13}$$

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).

1.5 Stima della capacità termica a volume costante

Un'altra quantità interessante che si può stimare mediante una simulazione Monte Carlo è la capacità termica a volume costante, definita come

$$C_V(T) \equiv \frac{\partial U}{\partial T} , \tag{14}$$

dove $U(T)$ è l'energia interna del sistema. Quest'ultima si può scrivere come

$$U(T) = \langle E_{pot} \rangle + \langle E_{kin} \rangle ,$$

dove E_{pot} è l'energia potenziale ed E_{kin} è l'energia cinetica. Lavorando nell'*ensemble* canonico, per il teorema di equipartizione, vale

$$\langle E_{kin} \rangle = \frac{3}{2} N k_B T .$$

Inoltre, si ha

$$\langle E_{pot} \rangle = \frac{\int V(\mathbf{r}_1, \dots, \mathbf{r}_N) e^{-\beta V(\mathbf{r}_1, \dots, \mathbf{r}_N)} \mathrm{d}\mathbf{r}_1 \dots \mathrm{d}\mathbf{r}_N}{\int e^{-\beta V(\mathbf{r}_1, \dots, \mathbf{r}_N)} \mathrm{d}\mathbf{r}_1 \dots \mathrm{d}\mathbf{r}_N} .$$

Sostituendo tali relazioni nella definizione (14) e svolgendo la derivata rispetto a T ricordando che $\beta \equiv (k_B T)^{-1}$, si ottiene infine

$$\boxed{C_V(T) = \frac{\langle E_{pot}^2 \rangle - \langle E_{pot} \rangle^2}{k_B T^2} + \frac{3}{2} N k_B \,.} \tag{15}$$

2 Descrizione del codice

Al fine di simulare il comportamento del gas di Argon nell'*ensemble* canonico, si è realizzato un programma in linguaggio C che implementa l'algoritmo di Metropolis e il calcolo della pressione P e della capacità termica a volume costante C_V del gas. Per l'implementazione si è ritenuto opportuno considerare unità adimensionali così definite:

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

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

Il programma, nella funzione principale `main()`, per ciascuno degli `N_T` valori di temperatura nell'*array* `const double temperature[N_T]` e ciascuno degli `N_d` valori di densità nell'*array* `const double densita[N_d]`, esegue, in sequenza, le seguenti operazioni:

1. generazione delle posizioni iniziali delle particelle nella scatola di simulazione mediante la funzione `genera_r()`, che le dispone su un reticolo cubico come riferito nella sezione 1.1;

2. calcolo dell'energia potenziale e del viriale iniziali utilizzando rispettivamente le funzioni `calcola_E(Punto*)` ed `aggiorna_W()`;

3. fase di equilibratura, durante la quale il sistema viene fatto evolvere mediante l'algoritmo di Metropolis fino alla sua termalizzazione, ovvero fino alla perdita di memoria dello stato iniziale e alla convergenza della distribuzione di probabilità campionata alla distribuzione di Boltzmann (1);

4. fase statistica, che prevede l'evoluzione del sistema mediante l'algoritmo di Metropolis e, ad ogni iterazione, il calcolo del viriale e l'aggiornamento delle funzioni di autocorrelazione necessarie alla stima dell'errore su pressione e capacità termica;

5. normalizzazione delle funzioni di autocorrelazione e calcolo della lunghezza di autocorrelazione τ;

6. calcolo della pressione media e della capacità termica a volume costante, con le relative incertezze.

Più in dettaglio, nella fase di equilibratura, viene richiamata ciclicamente la funzione `passo_metropolis()`, che esegue un passo dell'algoritmo di Metropolis, così come descritto nella sezione 1.2. Per ottimizzare la gestione delle posizioni delle `N = 125` particelle nella scatola di simulazione, vengono utilizzati due *array* dichiarati globali, `Punto r1[N]` e `Punto r2[N]`, dove `Punto` è una *struct* definita nel modo seguente:

```
1 typedef struct{
2     double r[3];
3 }Punto;
```

L'*array* `r` rappresenta un punto di coordinate $(x, y, z) = ($ `r[0]`, `r[1]`, `r[2]` $)$. Gli *array* `r1` e `r2`, ad ogni passo dell'algoritmo di Metropolis, contengono uno le nuove posizioni generate, l'altro quelle del passo precedente. `r1` e `r2` sono puntati da due puntatori, `Punto *p_r1` e `Punto *p_r2`, anch'essi dichiarati globali. Quando viene richiamata `passo_metropolis()`, `p_r1` punta all'*array* contenente le posizioni del passo precedente, `p_r2` punta all'*array* in cui vengono salvate le nuove posizioni generate. Se la nuova configurazione proposta viene accettata, allora il contenuto dei due puntatori viene scambiato. In questo modo, si evita di dover copiare $3N$ valori di tipo `double` da un *array* all'altro ogni volta che un nuovo *set* di posizioni viene accettato.
Nella nostra implementazione, la fase di equilibratura, oltre ad essere preposta al raggiungimento della termalizzazione del sistema, ha anche lo scopo di trovare un valore di Δ (memorizzato nella variabile globale `double Delta`) per cui l'*acceptance rate* appartenga all'incirca ad un intervallo fissato $($ `ACC_MIN`, `ACC_MAX` $)$. A tale fine, ogni `STEP_CONTROL = 400` iterazioni del ciclo costituente la fase di equilibratura, viene valutato l'*acceptance rate* parziale `fraz_acc`: se vale `fraz_acc > ACC_MAX`, allora viene sommata a `Delta` una quantità fissata `dDelta`; se invece risulta `fraz_acc < ACC_MIN`, allora `dDelta` viene sottratto a `Delta`. Se in due controlli consecutivi, `Delta` viene incrementato e decrementato, allora la quantità `dDelta` viene dimezzata. Se, dopo un controllo, `Delta` risulta negativo, allora gli viene nuovamente sommato `dDelta` e quest'ultimo viene poi dimezzato.

Anche nella successiva fase statistica viene richiamata ciclicamente la funzione `passo_metropolis()`, ma con `Delta` fissato al valore stabilito al termine della fase di equilibratura. Ad ogni iterazione del ciclo costituente la fase statistica, viene anche aggiornato il valore del viriale (tramite la funzione `aggiorna_W()`), solo se diverso da quello trovato al passo precedente. Inoltre, allo scopo di valutare la pressione media P e la capacità termica a volume costante C_V, ad ogni iterazione, vengono accumulate in apposite variabili (inizializzate a zero) le somme e le somme dei quadrati di energia potenziale, energia potenziale al quadrato e viriale; terminato il ciclo tali quantità vengono utilizzate per ricavare P e C_V, mediante le formule (13) e (15) rispettivamente, e le relative deviazioni standard.

Come discusso nella sezione 1.3, per valutare correttamente l'incertezza delle osservabili calcolate, è necessario disporre di una stima delle rispettive funzioni di

autocorrelazione. A tale scopo, gli ultimi `N_E`, `N_E2` e `N_W` valori calcolati di energia potenziale, energia potenziale al quadrato e viriale vengono rispettivamente memorizzati nei tre *array* `double E[N_E]`, `double E2[N_E2]` e `double W[N_W]`, dichiarati globali. Per ottimizzare l'esecuzione del codice, tali *array* sono utilizzati come degli *array* circolari; a tale scopo, gli indici che individuano l'ultimo valore inserito in ciascun *array* sono memorizzati nelle variabili globali `idx_E`, `idx_E2` e `idx_W`. Quando ad esempio `E` risulta pieno, il valore di energia potenziale più vecchio viene sostituito con il più recente, mantenendo memoria dell'ordine sequenziale delle energie grazie a `idx_E`.
Per calcolare le funzioni di autocorrelazione delle tre grandezze, durante la fase statistica, per ciascuno dei tre *array*, viene richiamata ciclicamente la funzione `aggiorna_C(double*, double*, int, int)`. Quest'ultima riceve in *input* i puntatori a due *array*, `C` ed `O` (con `O` *array* circolare contenente un'osservabile calcolata a passi diversi), la lunghezza `lun` dei due, e l'indice `idx_now` che individua il valore più recente memorizzato in `O`. La funzione, per ogni `i` $= 0, 1, ...,$ `lun-1`, incrementa l'elemento `C[i]` della quantità `O[idx_now] * O[idx]`, dove si è posto `idx = (idx_now - i + lun) % lun`[1].
Al termine della fase statistica, affinché `C` contenga la funzione di autocorrelazione di `O` correttamente normalizzata, è necessario compiere alcune operazioni su ciascun elemento `C[i]`: si deve dividerlo per il numero di *step* dell'algoritmo di Metropolis considerati per calcolarlo, sottrargli il quadrato del valor medio di `O` calcolato sui medesimi passi e normalizzarlo dividendolo per per `C[0]`. Tali operazioni sono eseguite dalla funzione `normalizza_C(double*, double, int, int)`, che ha anche lo scopo di calcolare la lunghezza di autocorrelazione τ. Quest'ultima è stimata individuando, per interpolazione lineare, il punto d'intersezione della funzione di autocorrelazione con la retta di equazione $y = e^{-1}$.

Nella funzione `passo_metropolis()`, alle nuove posizioni generate sono applicate le condizioni periodiche al contorno di Born-Von Karman riferite nella sezione 1. A tale scopo, a ciascuna componente di ogni nuova posizione generata 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 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 $\tilde{L}/2$.

[1] Nel linguaggio C, la quantità `a % b`, con `a`, `b` variabili di tipo `int`, è pari al resto della divisione intera `a / b`.

Alcuni accorgimenti significativi che si sono adottati per rendere più efficiente il codice sono: l'uso diffuso di puntatori; la sovrascrittura di ogni variabile contenente quantità di cui non è necessario tenere memoria; la particolare gestione degli *array* globali `E`, `E2` e `W`, utilizzati come *array* circolari; l'evitare di ricalcolare energia e viriale quando questi non cambiano.
Al fine di monitorare l'andamento della simulazione, ogni `STEP_PRINT = 1000` iterazioni del ciclo costituente la fase statistica, vengono stampati a video i valori istantanei dell'energia potenziale e del viriale, e l'ammontare dell'*acceptance rate*. Al termine del ciclo, vengono stampati a video i valori medi di energia potenziale, energia potenziale al quadrato e viriale, le rispettive lunghezze di autocorrelazione, e le stime di pressione e capacità termica a volume costante, seguite dai propri errori statistici. Per ogni coppia di valori di temperatura e densità considerate, vengono salvati su appositi file le tre funzioni di autocorrelazione calcolate e le stime di P e C_V ottenute. 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 $N_s = 5 \cdot 10^5$ passi e una fase di equilibratura di almeno $N_e = 10^4$ *step*, anche tenendo conto delle lunghezze di autocorrelazione trovate in alcune esecuzioni preliminari del codice. Allo scopo di trovare l'intervallo di valori di *acceptance rate* che minimizzano le lunghezze di autocorrelazione (e quindi le incertezze) delle osservabili calcolate, per alcuni valori di temperatura e densità, si è eseguito il codice al variare delle costanti `ACC_MIN` ed `ACC_MAX`, ricavando i grafici riportati nelle figure 1 e 2.

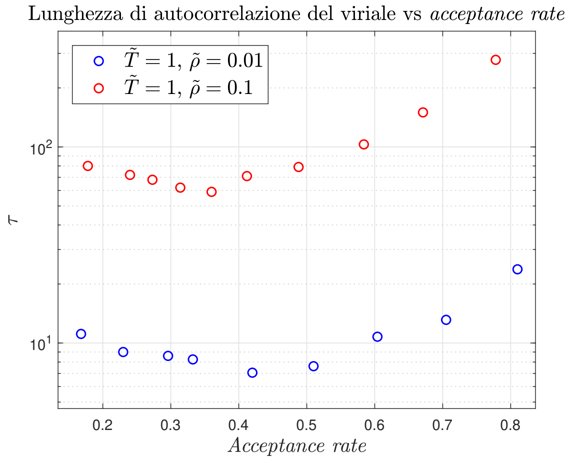

Figura 1: Lunghezza di autocorrelazione τ del viriale in funzione dell'*acceptance rate* per temperatura $\tilde{T} = 1$ e due diversi valori di densità.

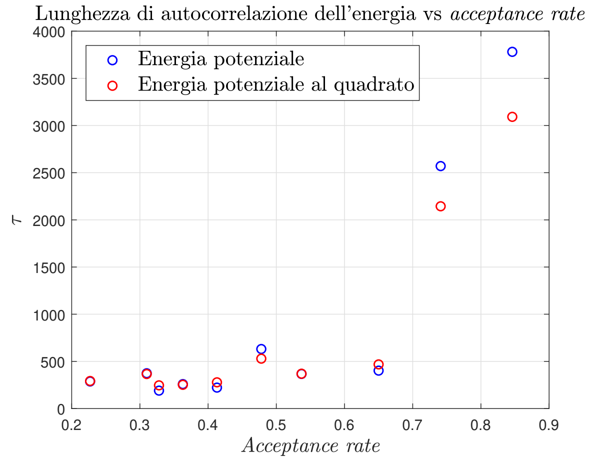

Figura 2: Lunghezza di autocorrelazione τ dell'energia potenziale e del suo quadrato in funzione dell'*acceptance rate* per temperatura $\tilde{T} = 1$ e densità $\tilde{\rho} = 0.01$.

Dai due grafici emerge come i valori ottimali di *acceptance rate* siano da ricercarsi nell'intervallo (0.3, 0.4) circa. Nella simulazione, si è dunque scelto di imporre un valore di *acceptance rate* appartenente proprio a tale intervallo.

Il codice completo è riportato in Appendice C.

3 Discussione dei risultati e confronto con la teoria

Nel grafico seguente (Fig.3) è rappresentata l'isoterma ottenuta numericamente per la temperatura più bassa considerata, ovvero $\widetilde{T} = 0.7$, utilizzando 24 valori di densità differenti.

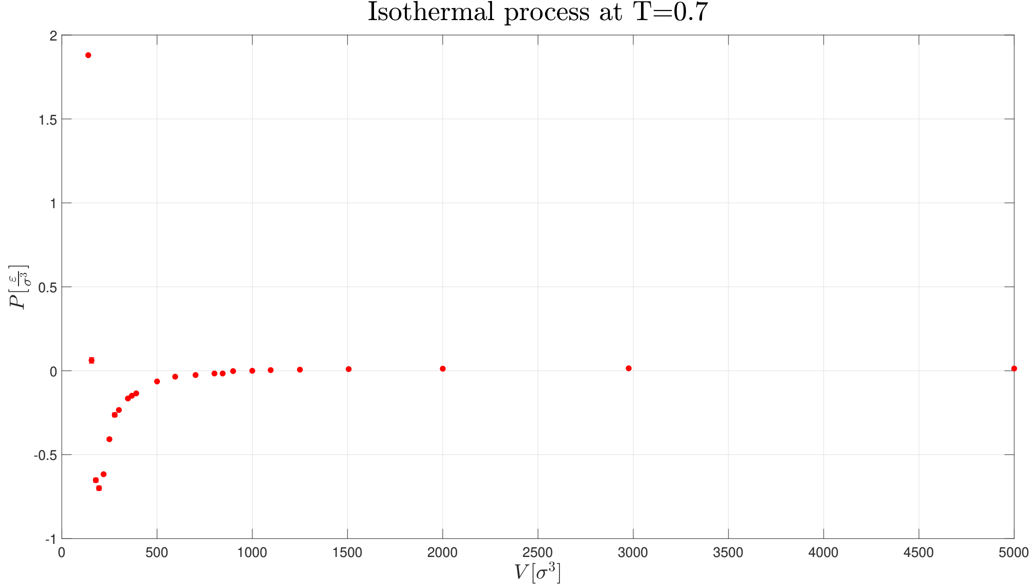

Figura 3: Grafico P-V dell'isoterma per $\widetilde{T} = 0.7$.

L'errore sui punti graficati è stato propagato partendo da quello sulla stima del viriale. Quest'ultimo è stato valutato considerando l'errore statistico corretto secondo la relazione (11), che tiene conto delle autocorrelazioni introdotte dall'algoritmo di Metropolis. In appendice A sono riportati alcuni grafici esemplificativi della funzione di autocorrelazione del viriale (Fig. 11 e 12).

Tornando al grafico in Fig.3, si osserva che, per la temperatura $\widetilde{T} = 0.7$, minore di quella critica, l'andamento della curva non è monotono decrescente e addirittura la pressione risulta negativa in un certo intervallo di volumi. Il motivo di questi comportamenti è riconducibile alla finitezza del sistema simulato. Il numero di particelle nella scatola di simulazione infatti non è sufficientemente grande per simulare efficacemente la transizione di fase che avverrebbe in realtà. La situazione ideale si avrebbe nel caso di una simulazione di almeno un numero di Avogadro di molecole, anche se già con decine di migliaia di particelle si ottengono buoni risultati. Per $N = 125$, il sistema esplora invece stati non fisici dello spazio delle fasi, che non sarebbero in realtà permessi, dovendo la pressione rimanere costante per l'intervallo di densità, dipendente dall'isoterma considerata, a cui avviene la transizione di fase. Quanto detto non crea però problemi a basse densità per nessuna temperatura considerata (Fig.4). In questo regime, si nota addirittura un buon accordo tra i punti calcolati numericamente e il modello di gas ideale, come ci aspetteremmo d'altronde.

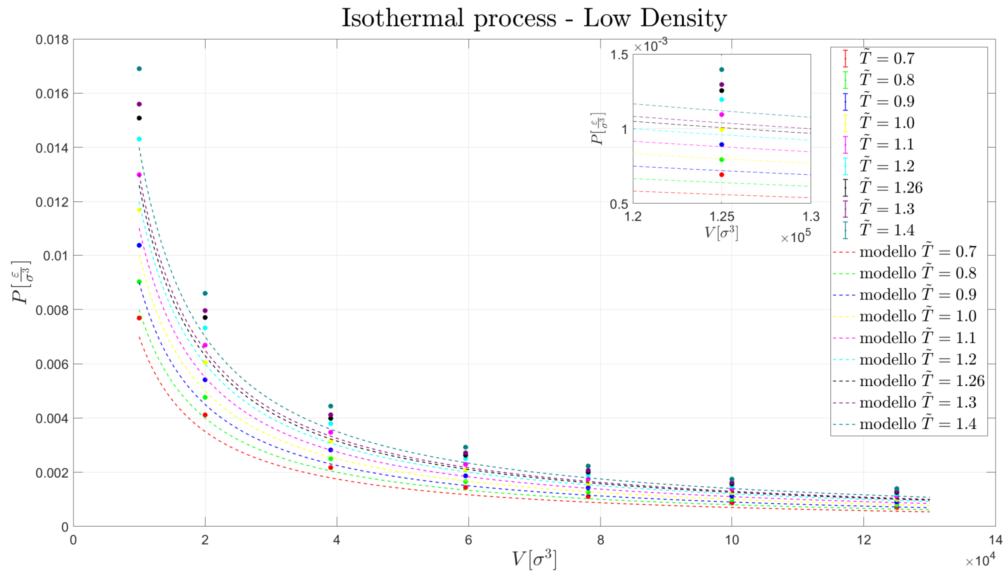

Figura 4: Confronto tra le isoterme stimate numericamente e il modello del gas ideale.

Osserviamo inoltre che l'iperbole dell'isoterma prevista dal modello del gas perfetto risulta tanto più in accordo con quanto ottenuto dalla simulazione tanto più le temperature sono basse. Anche in questo caso è quello che ci si aspetta, pensando al termine correttivo del volume presente nell'equazione di Van der Waals.
Per densità maggiori però, anche se non eccessivamente elevate, nemmeno a basse temperature c'è accordo con il modello (Fig.5).

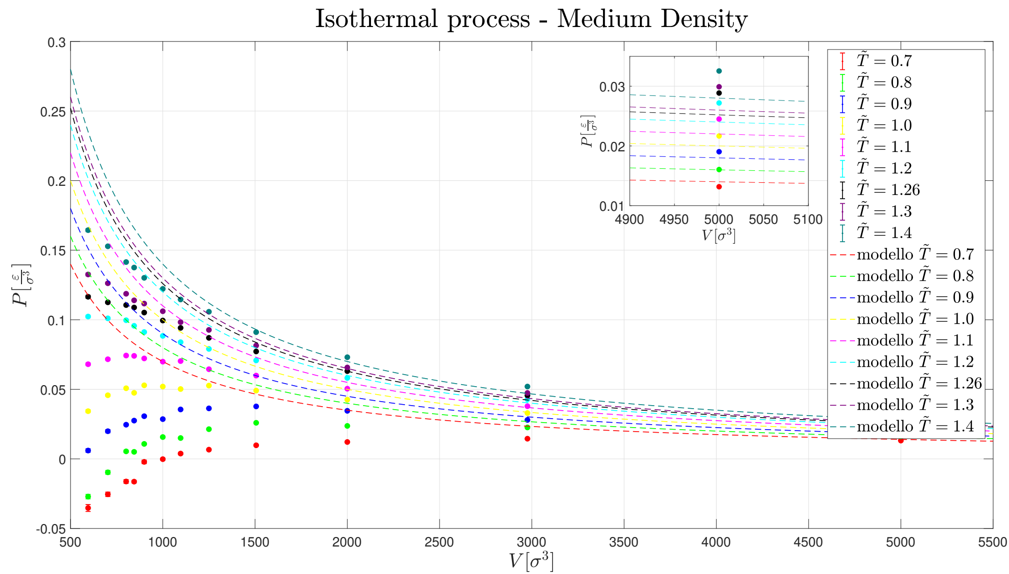

Figura 5: Isoterme e modello nella regione critica.

In questa regione di densità iniziano a manifestarsi i comportamenti descritti sopra, ma è comunque possibile fornire una stima della temperatura critica T_c. Risulta infatti chiara graficamente la presenza di un punto di flesso nelle isoterme che inizia a comparire per temperature adimensionali comprese tra 1.26 e 1.3. Questo intervallo

corrisponde a $T_c \in [150\,\mathrm{K},\ 156\,\mathrm{K}]$ in unità del Sistema Internazionale. Concludiamo che la stima ottenuta è in accordo con il valore tabulato pari a $T_c = 150.8\,\mathrm{K}$. In aggiunta, anche la pressione critica, tabulata a $P_c = 4898\,\mathrm{kPa}$ ovvero $\widetilde{P}_c = 0.116$, risulta entro le due isoterme sopracitate in corrispondenza del cambio di pendenza. Ovviamente il modello di gas ideale risulta sempre meno accurato per alte densità (Fig.6).

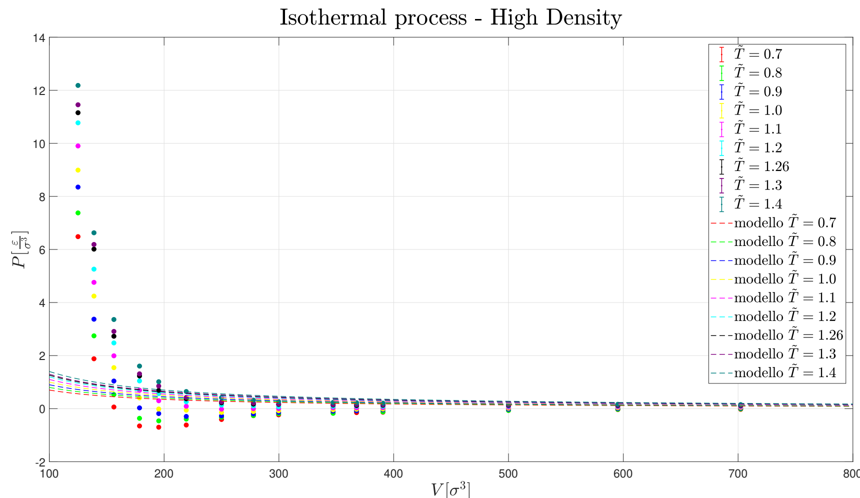

Figura 6: Isoterme e modello ad alte densità.

Infatti, oltre al comportamento non fisico, notiamo che la pressione del gas simulato diverge già per valori di $\widetilde{V}$ ben maggiori di quelli previsti.

In Fig.7 è riportato il grafico della capacità termica a volume costante C_V calcolata per l'isoterma a $\widetilde{T} = 0.7$ per 12 densità differenti. In appendice B si trova il grafico della capacità termica in funzione del volume per tutti i valori di temperatura considerati.

Come si nota da un rapido confronto con la Fig.3, gli errori relativi su questi punti sono molto maggiori rispetto a quelli ottenuti per la pressione.

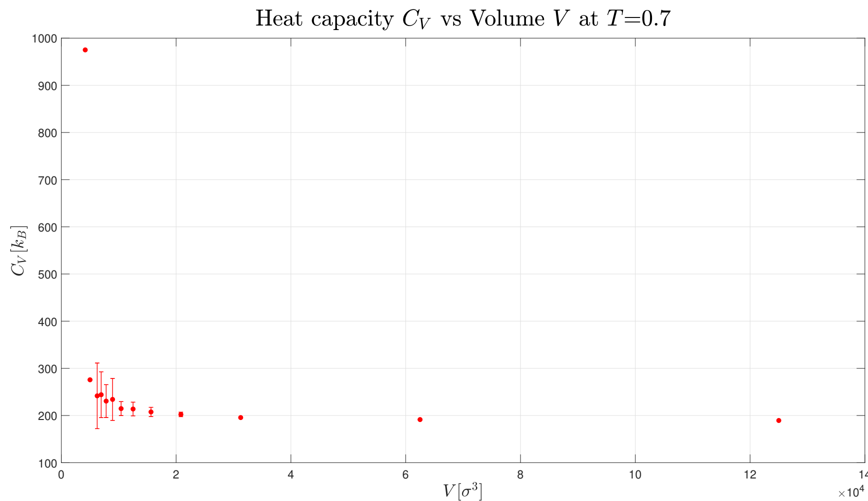

Figura 7: Capacità termica a volume costante C_V in funzione del volume V per $\widetilde{T} = 0.7$.

Per i due punti a volume più basso non è riportato l'ammontare dell'incertezza, in quanto, con il programma utilizzato, non è possibile una stima corretta della stessa. Infatti per densità troppo alte le autocorrelazioni dell'energia potenziale e del suo quadrato smettono di avere un comportamento monotono decrescente e tantomeno esponenziale. Questo è ancora una volta dovuto alla dimensione finita della scatola di simulazione. Inoltre, come ci si aspetta, diminuendo il volume le autocorrelazioni aumentano e quindi, anche nel caso in cui si abbia una buona curva, il numero di passi necessario per stimare la lunghezza di autocorrelazione τ diventa così grande da allungare esageratamente il tempo di esecuzione del programma. Un esempio di ciò è riportato nella figura seguente (Fig.8), ricordando che per stimare correttamente τ dovremmo trovare il numero di passi per cui $C_{E^2} \approx e^{-1} \approx 0.367$.

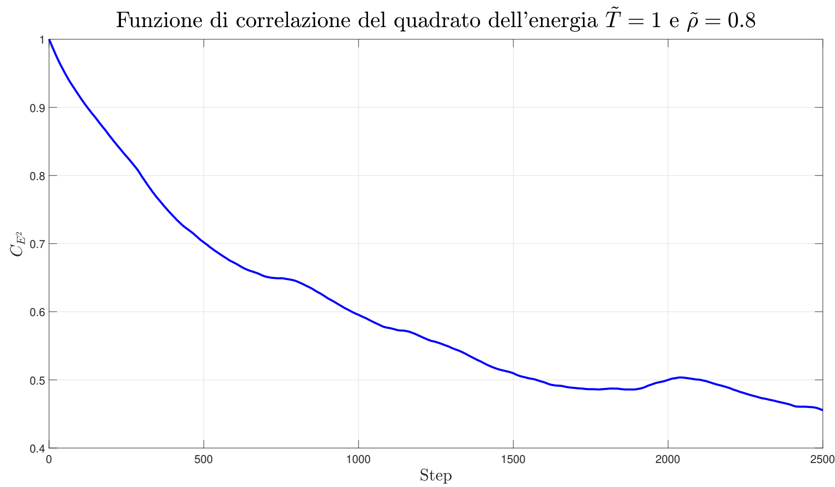

Figura 8: Funzione di autocorrelazione del quadrato dell'energia potenziale per $\widetilde{T} = 1$ e $\widetilde{\rho} = 0.8$, per 2500 passi.

Per i suddetti motivi, uniti al fatto che l'incertezza relativa su C_V sarebbe maggiore del 50%, il calcolo della capacità termica a volume costante è stato effettuato fino a densità adimensionali pari a $\widetilde{\rho} = 0.03$ (Fig.9).

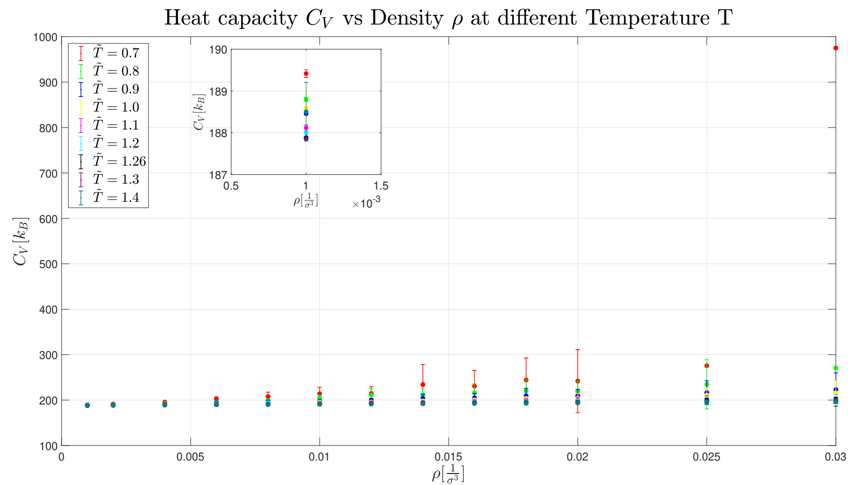

Figura 9: Capacità termica a volume costante C_V in funzione della densità ρ a diverse temperature.

Grafici esemplificativi delle funzioni di autocorrelazione dell'energia potenziale e del suo quadrato a basse densità si trovano ancora in appendice A (Fig. 13 e 14).
Il grafico che risulterebbe più significativo per fornire un'ulteriore conferma del punto critico dell'Argon sarebbe quello della capacità termica a volume costante in funzione della temperatura. Di seguito ne è riportato uno per tutte le densità considerate, fino a $\widetilde{\rho} = 0.03$ (Fig.10).

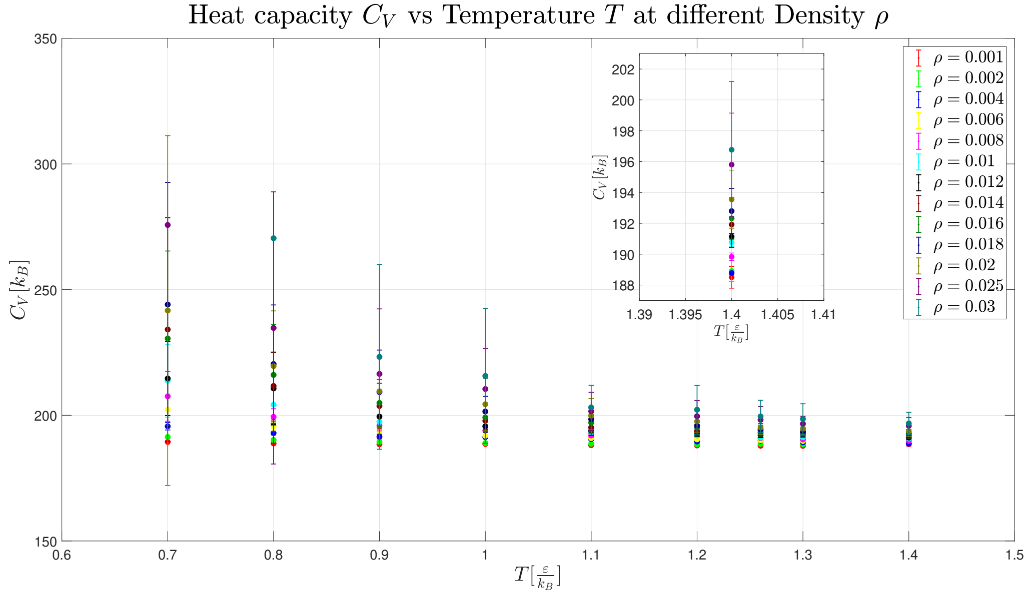

Figura 10: Capacità termica a volume costante C_V in funzione della temperatura T a diverse densità.

Nel nostro caso il grafico però non porta alcuna informazione per quanto riguarda la temperatura critica del gas, data l'assenza di un picco, qualsiasi sia la densità considerata. Ma è giusto che sia così: infatti la densità critica tabulata per l'Argon è pari a $\rho_c = 0.5377\,\mathrm{kg/dm^3}$, che, in unità adimensionali, corrisponde a $\widetilde{\rho_c} = 0.32$. Giungiamo alla conclusione che i volumi da noi considerati sono troppo grandi per riuscire ad ottenere una stima della temperatura critica attraverso l'andamento della capacità termica. La strada più percorribile allora rimane quella del calcolo della pressione considerata in precedenza.

A Grafici delle autocorrelazioni

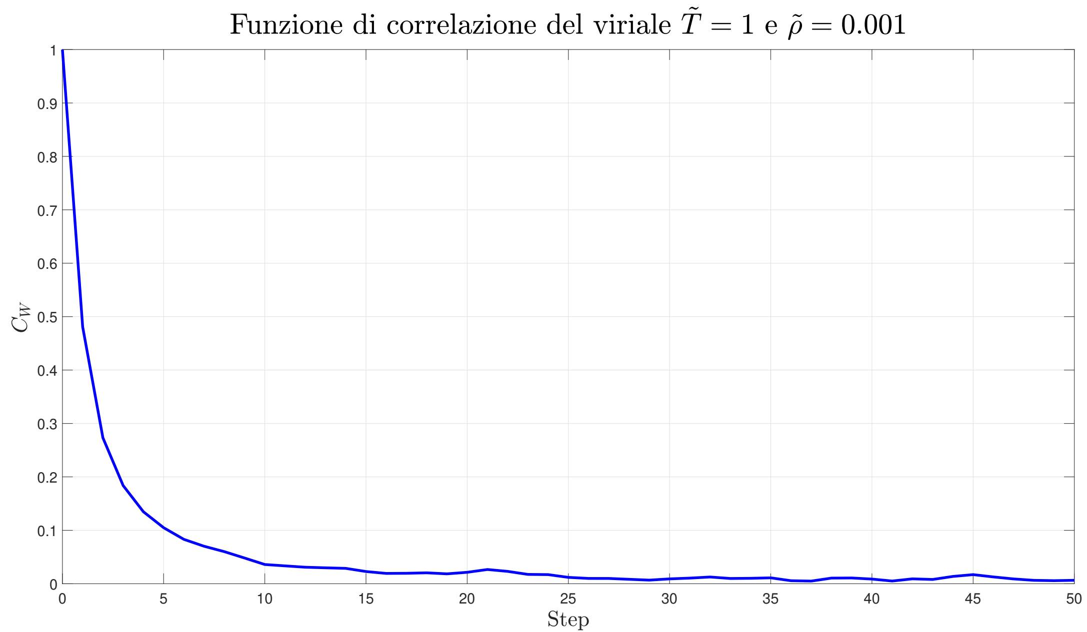

Figura 11: Funzione di autocorrelazione del viriale per $\widetilde{T} = 1$ e $\widetilde{\rho} = 0.001$, per 50 passi.

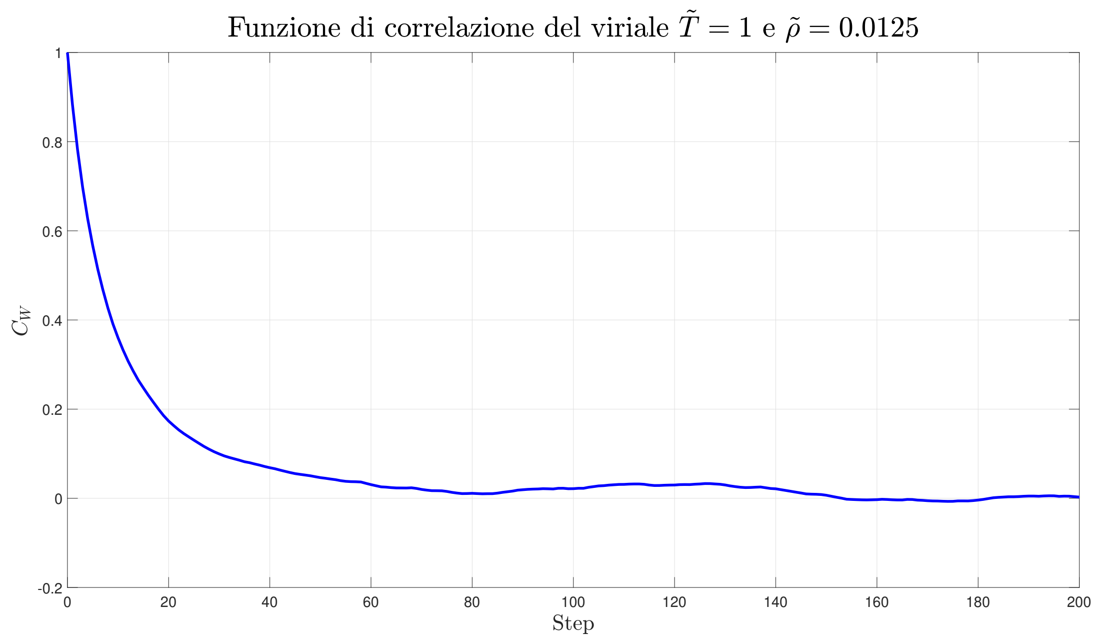

Figura 12: Funzione di autocorrelazione del viriale per $\widetilde{T} = 1$ e $\widetilde{\rho} = 0.0125$, per 200 passi.

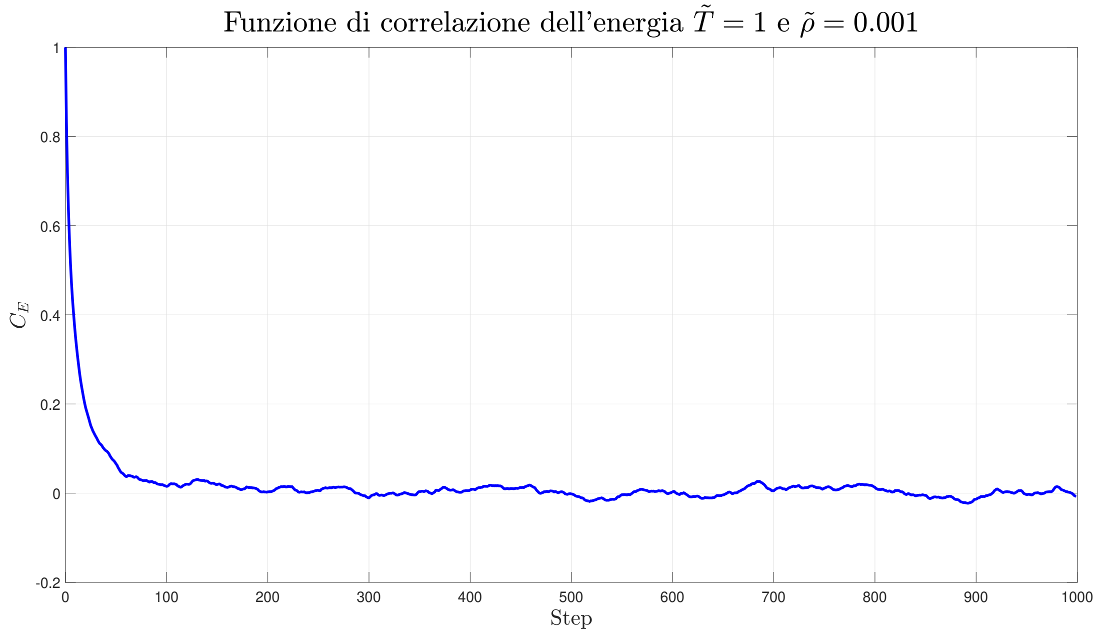

Figura 13: Funzione di autocorrelazione dell'energia potenziale per $\widetilde{T} = 1$ e $\widetilde{\rho} = 0.001$, per 1000 passi.

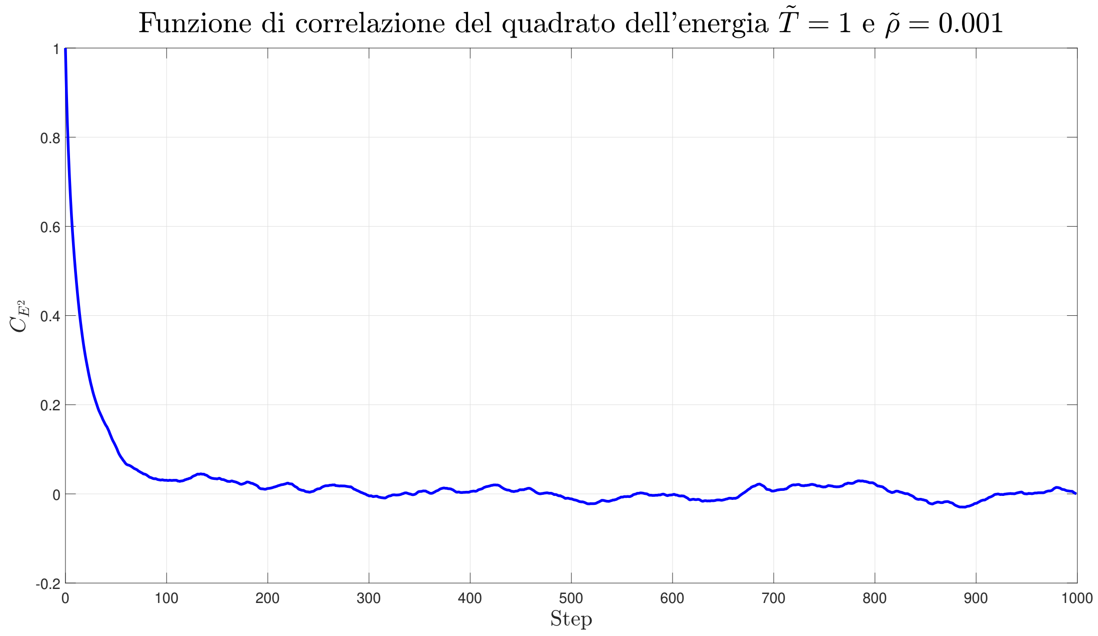

Figura 14: Funzione di autocorrelazione del quadrato dell'energia potenziale per $\widetilde{T} = 1$ e $\widetilde{\rho} = 0.001$, per 1000 passi.

B Capacità termica

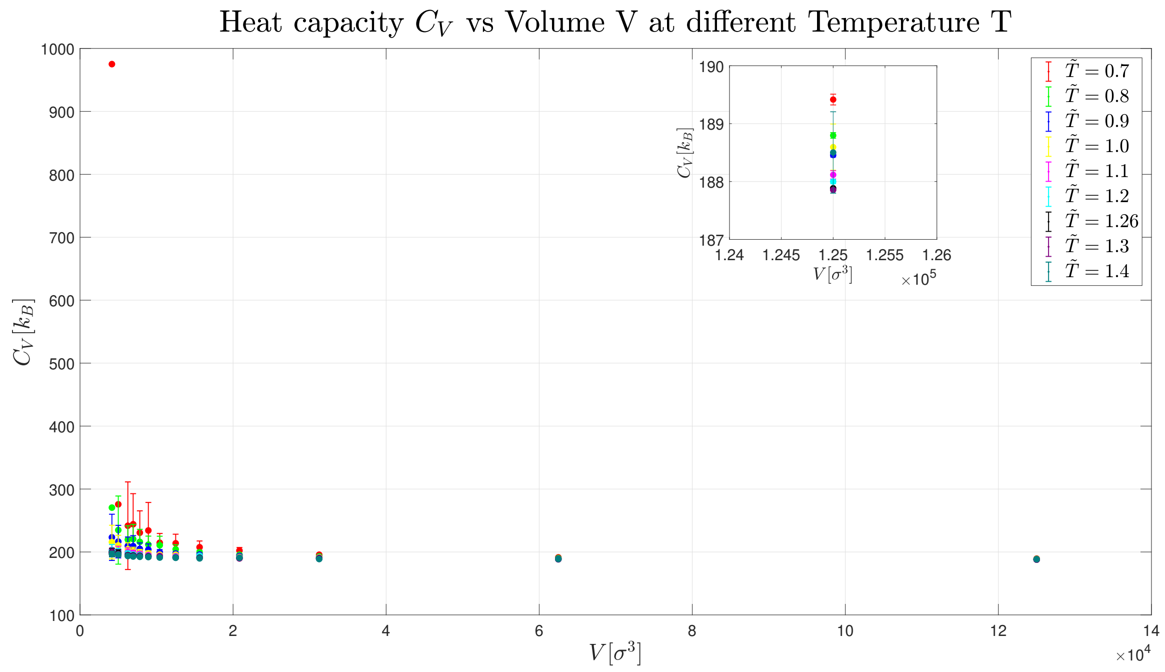

Figura 15: Capacità termica a volume costante C_V in funzione del volume V per diverse temperature.

C Codice in linguaggio C

```c
// MONTE CARLO: fluido di Lennard-Jones

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


// COSTANTI ---------------------------------

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

#define N_eq 10000 // Numero minimo di step della fase di
    equilibratura
#define N_stat 500000 // Numero di step della fase statistica

/* Numero di passi precedenti per cui vengono mantenuti in
    memoria viriale ("N_W"), energia potenziale ("N_E") ed
    energia potenziale al quadrato ("N_E2") (per il calcolo
    delle funzioni di autocorrelazione) */
#define N_W 5000
#define N_E 5000
#define N_E2 5000

// Numero di passi tra i controlli dell'acceptance rate
#define STEP_CONTROL 400

#define ACC_MAX 0.42 // Acceptance rate massimo
#define ACC_MIN 0.36 // Acceptance rate minimo

// Numero di passi tra una stampa e video e l'altra dei valori
     istantanei delle osservabili
#define STEP_PRINT 1000

#define N_T 9 // Numero di elementi dell'array "temperature"
#define N_d 31 // Numero di elementi dell'array "densita"

/* Valori di temperatura e densita' (in unita' adimensionali)
    per cui viene effettuata la simulazione */
const double temperature[N_T] = {0.7, 0.8, 0.9, 1, 1.1, 1.2,
    1.26, 1.3, 1.4};
const double densita[N_d] = {0.001, 0.00125, 0.0016, 0.0021,
    0.0032, 0.00625, 0.0125, 0.025, 0.042, 0.0625, 0.083, 0.1,
    0.114, 0.125, 0.139, 0.148, 0.156, 0.178, 0.21, 0.25, 0.32,
```

```
    0.34, 0.36, 0.417, 0.45, 0.5, 0.57, 0.64, 0.7, 0.8, 0.9};
36
37 // Prima parte del nome dei file salvati
38 const char nomeFile[] = "C:\\Users\\Admin\\Documents\\MATLAB\\
   esercizio5_";
39 const char estens[] = ".txt"; // Estensione dei file
40
41
42 // DICHIARAZIONE STRUTTURA ---------------------------------
43 typedef struct{
44     /* "r" contiene le coordinate di un punto:
45     0 -> x, 1 -> y, 2 -> z */
46     double r[3];
47 }Punto;
48
49
50 // PROTOTIPI DELLE FUNZIONI ---------------------------------
51 void genera_r(); // Genera le posizioni iniziali delle
   particelle
52 double pbc(double); // Applica le condizioni periodiche al
   bordo
53 double der_lj(double); // Calcola la derivata del potenziale
   di Lennard-Jones
54 double lj(double); // Calcola il potenziale di Lennard-Jones
55 double calcola_E(Punto*); // Calcola l'energia potenziale
56 void aggiorna_W(); // Aggiorna il viriale
57 void inizializza_C(double*, int); // Inizializza l'array di
   una funzione di autocorrelazione
58 void aggiorna_C(double*, double*, int, int); // Aggiorna l'
   array per il calcolo di una funzione di autocorrelazione
59 double normalizza_C(double*, double, int, int); // Normalizza
   una funzione di autocorrelazione e ne calcola il tau
60 short passo_metropolis(); // Esegue un passo dell'algoritmo di
    Metropolis
61
62
63 // VARIABILI GLOBALI ---------------------------------
64 double ro; // Densita' adimensionale
65 double T; // Temperatura adimensionale
66 double L, L2; // Lato e meta' lato della scatola di
   simulazione (adimensionali)
67 double V; // Volume della scatola di simulazione (
   adimensionale)
68
69 // Array contenenti le posizioni delle N particelle
70 Punto r1[N], r2[N];
```

```
71
72 // Puntatori agli array "r1" e "r2"
73 Punto *p_r1, *p_r2;
74
75 /* Array circolari contenenti gli ultimi "N_W", "N_E", "N_E2"
      valori calcolati per viriale, energia potenziale, energia
      poteziale al quadrato, rispettivamente */
76 double W[N_W], E[N_E], E2[N_E2];
77
78 /* Indici degli array "W", "E", "E2" in cui sono salvati gli
      ultimi valori calcolati delle rispettive osservabili */
79 int idx_W, idx_E, idx_E2;
80
81 double Delta; // Delta per generazione nuove posizioni
82
83
84 // MAIN ---------------------------------
85 int main(){
86     FILE *file;
87     FILE *file_isoterme;
88     FILE *file_CV;
89     char str[60];
90
91     /* Funzioni di autocorrelazione di viriale ("CW"), energia
          potenziale ("CE"), energia potenziale al quadrato ("
        CE2") */
92     double CW[N_W], CE[N_E], CE2[N_E2];
93
94     int i, j, k;
95     double E_sum; // Somma dell'energia potenziale a passi
        diversi
96     double E2_sum; // Somma dell'energia potenziale al
        quadrato a passi diversi
97     double E4_sum; // Somma dell'energia potenziale alla
        quarta a passi diversi
98     double W_sum; // Somma del viriale a passi diversi
99     double W2_sum; // Somma del quadrato del viriale a passi
        diversi
100    double tau_W, tau_E, tau_E2;
101    double E_mean, E_std;
102    double E2_mean, E2_std;
103    double W_mean, W_std;
104    double P, P_std;
105    double CV, CV_std;
106
107    int acc, acc_tot;
```

```
108     double fraz_acc;
109     double dDelta;
110     int idx;
111     short test[2];
112     int num_eq;
113
114     printf("Fluido di Lennard-Jones: simulazione Monte Carlo\n
           \n");
115
116     // Ciclo sulle temperature nell'array "temperature"
117     for(j = 0; j < N_T; j++){
118         T = temperature[j];
119
120         // Apro file per salvataggio pressioni e capacita'
               termiche
121         sprintf(str, "%s%s%c%s", nomeFile, "ro-P_", 'a'+j,
               estens);
122         file_isoterme = fopen(str, "w");
123         sprintf(str, "%s%s%c%s", nomeFile, "CV_", 'a'+j,
               estens);
124         file_CV = fopen(str, "w");
125
126         // Ciclo sulle densita' nell'array "densita"
127         for(k = 0; k < N_d; k++){
128             ro = densita[k];
129
130             // Calcolo lato, mezzo lato e volume della scatola
                   di simulazione
131             L = pow(N / ro, 1./3.);
132             L2 = L * 0.5;
133             V = L * L * L;
134
135             // Inizializzazione degli array per le funzioni di
                   autocorrelazione
136             inizializza_C(CE, N_E);
137             inizializza_C(CE2, N_E2);
138             inizializza_C(CW, N_W);
139
140             // Assegnazione array a puntatori
141             p_r1 = r1;
142             p_r2 = r2;
143
144             // Generazione posizioni iniziali delle particelle
145             genera_r();
146
147             // Inizializzazione indici
```

```
148        idx_E = 0;
149        idx_E2 = 0;
150        idx_W = -1;
151
152        // Calcolo energia potenziale e viriale iniziali
153        E[idx_E] = calcola_E(p_r1);
154        E2[idx_E2] = E[idx_E] * E[idx_E];
155        aggiorna_W();
156
157        // Stampo a video info
158        printf("Temperatura: %f\n", T);
159        printf("Densita': %f\n", ro);
160        printf("Lato: %f\n", L);
161        printf("Energia potenziale iniziale: %f\n", E[
              idx_E]);
162        printf("Viriale iniziale: %f\n\n", W[idx_W]);
163        printf("Inizio della fase di equilibratura\n\n");
164        printf("Step\tE\t\tDelta\t\tAcceptance rate
              parziale\n");
165
166        // Inizializzazione di "Delta" e "dDelta"
167        Delta = L * 0.01;
168        dDelta = Delta * 0.5;
169
170        acc = 0;
171        acc_tot = 0;
172        test[0] = 2;
173        num_eq = N_eq;
174        fraz_acc = 1;
175
176        // Fase di equilibratura
177        for(i = 1; i <= num_eq; i++){
178            // Passo dell'algoritmo di Metropolis
179            if(passo_metropolis()){
180                acc++;
181                acc_tot++;
182            }
183
184            // Correzione di Delta in base all'acceptance
                  rate parziale stimato
185            if(i % STEP_CONTROL == 0){
186                fraz_acc = (double)acc / STEP_CONTROL;
187
188                if(fraz_acc > ACC_MAX){
189                    Delta += dDelta;
190                    test[1] = 1;
```

```
191                 }
192                 else if(fraz_acc < ACC_MIN){
193                     Delta -= dDelta;
194                     test[1] = 0;
195                 }
196 
197                 if(Delta <= 0){
198                     Delta += dDelta;
199                     dDelta *= 0.5;
200                 }
201                 else if(test[0] + test[1] == 1)
202                     dDelta *= 0.5;
203 
204                 test[0] = test[1];
205                 acc = 0;
206             }
207 
208             // Stampo a video
209             if(i % STEP_PRINT == 0){
210                 printf("%d\t%f", i, E[idx_E]);
211                 printf("\t%f\t%f\n", Delta, fraz_acc);
212             }
213 
214             if(i == num_eq && (fraz_acc > ACC_MAX ||
                   fraz_acc < ACC_MIN))
215                 num_eq++;
216         }
217 
218         printf("\nFine della fase di equilibratura\n");
219         printf("Acceptance rate: %f\n", (double)acc_tot /
                (double)num_eq);
220         printf("Delta: %f\n\n", Delta);
221         printf("Premi invio per continuare");
222         getchar();
223         printf("\n\nFase statistica\n\n");
224         printf("Step\tE\t\tW\t\tAcceptance rate\n");
225 
226         E_sum = 0;
227         E2_sum = 0;
228         E4_sum = 0;
229         W_sum = 0;
230         W2_sum = 0;
231 
232         acc_tot = 0;
233 
234         // Fase statistica
```

```
235         for(i = 1; i <= N_stat; i++){
236             // Passo dell'algoritmo di Metropolis
237             if(passo_metropolis()){
238                 aggiorna_W();
239                 acc_tot++;
240             }
241             else{
242                 idx = idx_W;
243                 idx_W = (idx_W + 1) % N_W;
244                 W[idx_W] = W[idx];
245             }
246 
247             // Aggiornamento funzioni di autocorrelazione
248             if(i >= N_E)
249                 aggiorna_C(CE, E, N_E, idx_E);
250             if(i >= N_E2)
251                 aggiorna_C(CE2, E2, N_E2, idx_E2);
252             if(i >= N_W)
253                 aggiorna_C(CW, W, N_W, idx_W);
254 
255             // Accumulo valori dell'energia potenziale e
                   del viriale
256             E_sum += E[idx_E];
257             E2_sum += E2[idx_E2];
258             E4_sum += E2[idx_E2] * E2[idx_E2];
259             W_sum += W[idx_W];
260             W2_sum += W[idx_W] * W[idx_W];
261 
262             // Stampo a video
263             if(i % STEP_PRINT == 0){
264                 printf("%d\t%f\t%f", i, E[idx_E], W[idx_W
                       ]);
265                 printf("\t%f\n", (double)acc_tot / (double
                       )i);
266             }
267         }
268 
269         printf("\nFine della fase statistica\n");
270         printf("Acceptance rate: %f\n\n", (double)acc_tot
               / N_stat);
271 
272         // Calcolo medie delle osservabili
273         E_mean = E_sum / N_stat;
274         E2_mean = E2_sum / N_stat;
275         W_mean = W_sum / N_stat;
276 
```

```
277             /* Normalizzazione e salvataggio della funzione di
                    autocorrelazione dell'energia potenziale, e
                   calcolo della sua lunghezza di autocorrelazione
                    tau */
278             sprintf(str, "%s%s%c%d%s", nomeFile, "CE_", 'a'+j,
                    k+1, estens);
279             file = fopen(str, "w");
280             tau_E = normalizza_C(CE, E_mean, N_E, N_stat - N_E
                    + 1);
281             for(i = 0; i < N_E; i++)
282                 fprintf(file, "%d %f\n", i, CE[i]);
283             fclose(file);
284
285             /* Normalizzazione e salvataggio della funzione di
                    autocorrelazione dell'energia potenziale al
                   quadrato, e calcolo della sua lunghezza di
                   autocorrelazione tau */
286             sprintf(str, "%s%s%c%d%s", nomeFile, "CE2_", 'a'+j
                   , k+1, estens);
287             file = fopen(str, "w");
288             tau_E2 = normalizza_C(CE2, E2_mean, N_E2, N_stat -
                    N_E2 + 1);
289             for(i = 0; i < N_E2; i++)
290                 fprintf(file, "%d %f\n", i, CE2[i]);
291             fclose(file);
292
293             /* Normalizzazione e salvataggio della funzione di
                    autocorrelazione del viriale, e calcolo della
                   sua lunghezza di autocorrelazione tau */
294             sprintf(str, "%s%s%c%d%s", nomeFile, "CW_", 'a'+j,
                    k+1, estens);
295             file = fopen(str, "w");
296             tau_W = normalizza_C(CW, W_mean, N_W, N_stat - N_W
                    + 1);
297             for(i = 0; i < N_W; i++)
298                 fprintf(file, "%d %f\n", i, CW[i]);
299             fclose(file);
300
301             // Calcolo deviazioni standard delle osservabili
302             E_std = sqrt((E2_sum / N_stat - E_mean * E_mean) /
                    (N_stat - 1.) * tau_E);
303             E2_std = sqrt((E4_sum / N_stat - E2_mean * E2_mean
                   ) / (N_stat - 1.) * tau_E2);
304             W_std = sqrt((W2_sum / N_stat - W_mean * W_mean) /
                    (N_stat - 1.) * tau_W);
305
```

```
306             // Calcolo pressione e capacita' termica, e le
                   relative incertezze
307             P = W_mean / (3. * V) + ro * T;
308             P_std = W_std / (3. * V);
309             CV = (E2_mean - E_mean * E_mean) / (T * T) + 1.5 *
                    N;
310             CV_std = (E2_std + 2. * E_std) / (T * T);
311
312             // Stampo a video medie delle osservabili
313             printf("Risultati:\n");
314             printf("tau dell'energia potenziale: %f\n", tau_E)
                   ;
315             printf("tau dell'energia potenziale al quadrato: %
                   f\n", tau_E2);
316             printf("tau del viriale: %f\n", tau_W);
317             printf("Energia potenziale media: %f +/- %f\n",
                   E_mean, E_std);
318             printf("Energia potenziale al quadrato media: %f
                   +/- %f\n", E2_mean, E2_std);
319             printf("Capacita' termica: %f +/- %f\n", CV,
                   CV_std);
320             printf("Viriale medio: %f +/- %f\n", W_mean, W_std
                   );
321             printf("Pressione media: %1.10f +/- %1.10f\n\n", P
                   , P_std);
322
323             // Salvataggio su file della pressione e della
                   capacita' termica
324             fprintf(file_isoterme, "%1.2f %1.5f %2.10f %1.10f\
                   n", T, ro, P, P_std);
325             fprintf(file_CV, "%1.2f %1.5f %2.10f %1.10f\n", T,
                    ro, CV, CV_std);
326
327             printf("Premi invio per continuare");
328             getchar();
329             system("cls");
330         }
331
332         fclose(file_isoterme);
333         fclose(file_CV);
334     }
335
336     return 0;
337 }
338
339
```

```
340 // FUNZIONI ---------------------------------
341
342 /* Posiziona le particelle nella scatola di simulazione su un
      reticolo */
343 void genera_r(){
344     double n = pow(N, 1./3.);
345     double dL = L / n;
346     int x, y, z, i = 0;
347
348     for(x = 0; x < n; x++){
349         for(y = 0; y < n; y++){
350             for(z = 0; z < n; z++){
351                 r1[i].r[0] = -L2 + dL * x;
352                 r1[i].r[1] = -L2 + dL * y;
353                 r1[i].r[2] = -L2 + dL * z;
354                 i++;
355             }
356         }
357     }
358 }
359
360
361 /* Esegue un passo dell'algoritmo di Metropolis. Restituisce
      il valore 1 se la nuova configurazione e' stata accettata,
      0 altrimenti. */
362 short passo_metropolis(){
363     short k;
364     int i, idx[2];
365     double xi, E_new, prob;
366     Punto *temp;
367
368     // Genero un nuovo set di posizioni
369     for(i = 0; i < N; i++){
370         for(k = 0; k < 3; k++){
371             xi = (double)rand() / RAND_MAX;
372             p_r2[i].r[k] = pbc(p_r1[i].r[k] + Delta * (xi -
                   0.5));
373         }
374     }
375
376     // Calcolo l'energia potenziale e la probabilita' di
           accettazione
377     E_new = calcola_E(p_r2);
378     prob = exp(- (E_new - E[idx_E]) / T); // k_B = 1 in unita'
              adimensionali
379     xi = (double)rand() / RAND_MAX;
```

```
380
381     idx[0] = idx_E;
382     idx[1] = idx_E2;
383
384     // Aggiorno indici degli array "E", "E2"
385     idx_E = (idx_E + 1) % N_E;
386     idx_E2 = (idx_E2 + 1) % N_E2;
387
388     // Test per decidere se accettare la nuova configurazione
389     if(xi < prob){
390         // Scambia gli array puntati dai puntatori "p_r1" e "
                p_r2"
391         temp = p_r1;
392         p_r1 = p_r2;
393         p_r2 = temp;
394
395         E[idx_E] = E_new;
396         E2[idx_E2] = E_new * E_new;
397
398         return 1;
399     }
400
401     E[idx_E] = E[idx[0]];
402     E2[idx_E2] = E2[idx[1]];
403
404     return 0;
405 }
406
407
408 // Applica le condizioni periodiche al bordo
409 double pbc(double x){
410     return x - L * rint(x / L);
411 }
412
413
414 /* Calcola la derivata del potenziale di Lennard-Jones, data
       la distanza "x" */
415 double der_lj(double x){
416     return 24 * (- 2. / pow(x, 13) + 1. / pow(x, 7));
417 }
418
419
420 /* Calcola il potenziale di Lennard-Jones, dato il modulo
       quadro della distanza "x2" */
421 double lj(double x2){
422     double v1 = x2 * x2 * x2;
```

```
423     double v2 = v1 * v1;
424     return 4 * (1. / v2 - 1. / v1);
425 }
426
427
428 /* Calcola l'energia potenziale associata al set di posizioni
       contenute nell'array puntato da "rr" */
429 double calcola_E(Punto *rr){
430     int i, j;
431     short k;
432     double dr[3];
433     double dr2, L22 = L2 * L2;
434     double E_new = 0;
435
436     for(i = 1; i < N; i++){
437         for(j = 0; j < i; j++){
438             // Calcolo distanza tra particelle i-esima e j-
                   esima
439             for(k = 0; k < 3; k++)
440                 dr[k] = pbc(rr[i].r[k] - rr[j].r[k]);
441             dr2 = dr[0] * dr[0] + dr[1] * dr[1] + dr[2] * dr
                   [2];
442
443             if(dr2 < L22) // Considero solo le distanze minori
                    di L/2
444                 E_new += lj(dr2);
445         }
446     }
447
448     return E_new;
449 }
450
451
452 // Aggiorna il viriale
453 void aggiorna_W(){
454     int i, j;
455     short k;
456     double dr[3];
457     double dr_mod;
458
459     // Aggiorno indice dell'array "W"
460     idx_W = (idx_W + 1) % N_W;
461
462     // Inizializzo a zero "W[idx_W]"
463     W[idx_W] = 0;
464
```

```
465     for(i = 1; i < N; i++){
466         for(j = 0; j < i; j++){
467             // Calcolo distanza tra particelle i-esima e j-
                   esima
468             for(k = 0; k < 3; k++)
469                 dr[k] = pbc(p_r1[i].r[k] - p_r1[j].r[k]);
470             dr_mod = sqrt(dr[0] * dr[0] + dr[1] * dr[1] + dr
                   [2] * dr[2]);
471 
472             if(dr_mod < L2){ // Considero solo le distanze
                   minori di L/2
473                 W[idx_W] -= der_lj(dr_mod) * dr_mod;
474             }
475         }
476     }
477 }
478 
479 
480 /* Inizializza a zero gli elementi dell'array di lunghezza "
       lun" puntato da "C" */
481 void inizializza_C(double *C, int lun){
482     int i;
483     for(i = 0; i < lun; i++)
484         C[i] = 0;
485 }
486 
487 
488 /* Aggiorna l'array puntato da "C" per il calcolo di una
       funzione di autocorrelazione. Il puntatore "O" punta all'
       array contenente l'osservabile a cui si riferisce la
       funzione di autocorrelazione, calcolata a passi diversi. "
       lun" e' la lunghezza degli array puntati da "C" e da "O"; "
       idx_now" e' l'indice che individua l'elemento piu' recente
       dell'array puntato da "O". */
489 void aggiorna_C(double *C, double *O, int lun, int idx_now){
490     int i, idx;
491 
492     for(i = 0; i < lun; i++){
493         idx = (idx_now - i + lun) % lun; // Aggiorno "idx" all
               'indice corrispondente al passo (ultimo passo) - i
494         C[i] += O[idx_now] * O[idx];
495     }
496 }
497 
498 
499 /* Normalizza una funzione di autocorrelazione e calcola la
```

```
   lunghezza di autocorrelazione "tau". "C" e' un puntatore
   all'array contenente la funzione di autocorrelazione, "mean
   " e' il valor medio dell'osservabile a cui si riferisce, "
   lun" e' la lunghezza dell'array puntato da "C", "N_step" e'
    il numero di step dell'algoritmo di Metropolis utilizzati
   per il calcolo della funzione di autocorrelazione. La
   lunghezza di autocorrelazione e' stimata individuando il
   punto d'intersezione tra la funzione di autocorrelazione e
   la retta di equazione y = 1 / e. */
500 double normalizza_C(double *C, double mean, int lun, int
   N_step){
501     int i;
502     double tau = 0;
503     double mean2 = mean * mean;
504     double inv_e = 1. / M_E;
505 
506     C[0] = C[0] / (double)N_step - mean2;
507     for(i = 1; i < lun; i++){
508         C[i] = (C[i] / (double)N_step - mean2) / C[0];
509         if(C[i] <= inv_e && tau == 0){
510             // Interpolazione lineare
511             if(i != 1)
512                 tau = (inv_e - C[i]) / (C[i] - C[i-1]) + i;
513             else
514                 tau = (inv_e - C[i]) / (C[i] - 1) + i; // "C
                    [0] = 1" se normalizzato
515         }
516     }
517     C[0] = 1;
518 
519     return tau;
520 }
```