Università degli Studi di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di laurea in Fisica

Tesi di laurea triennale

Modellizzazione di vetri in silice con il potenziale di Tersoff

Sviluppo codice Python e Dinamica Molecolare della transizione α-quartz → amorfo

Relatore
prof. Mauro Ferrario

Candidato
Andrea PINTUS

ANNO ACCADEMICO 2019-2020

Indice

Introduzione

In questo progetto si studiano le transizioni di fase, attraverso una simulazione di dinamica molecolare, di un campione di silicio e ossigeno. Si è scelto di utilizzare come modello il potenziale di Tersoff, il quale descrive l'interazione tra le particelle attraverso equazioni che considerano più di due corpi contemporaneamente, a differenza dei potenziali di coppia basati sul modello di Lennard-Jones. Il potenziale di Tersoff viene infatti chiamato potenziale *many-body*, ed è, per questo motivo, relativamente più complicato dei potenziali di coppia, ma ha un raggio di interazione molto ridotto, che permette, nelle sue implementazioni, di considerare gruppi di pochi atomi per volta, alleggerendo il peso della simulazione. L'obiettivo del progetto è lo studio della fase vetrosa del composto SiO_2, partendo da una configurazione cristallina di α-quarzo. Per eseguire ciò bisogna prima fondere il cristallo, e, nel nostro caso, raggiungere temperature oltre i 3000°K, per poi effettuare un quenching, cioè un raffreddamento rapido del campione, in modo da non dare il tempo agli atomi di riposizionarsi nella configurazione cristallina, ottenendo così un amorfo. Ci si aspetta quindi di ottenere un solido in cui gli atomi assumono posizioni disordinate. Per verificare che il campione, nella fase finale, sia amorfo, si confrontano le funzioni di distribuzione radiale nelle tre fasi del campione. Esse ci danno abbastanza informazioni per capire se effettivamente si è passati da una configurazione cristallina ad una fase vetrosa.

La simulazione di dinamica molecolare viene eseguita scrivendo e poi utilizzando un codice, in linguaggio Python, facendo attenzione ad avere un giusto equilibrio tra performance e precisione. Per verificare l'attendibilità dei risultati del codice, essi vengono confrontati con quelli ottenuti con LAMMPS, un codice scritto in C/C++, usando le funzioni di distribuzione radiale come parametro di confronto.

Capitolo 1

Dinamica Molecolare

1.1 Approssimazione classica della Dinamica Molecolare

Prima di entrare nel dettaglio del potenziale di Tersoff è bene discutere del ruolo della dinamica molecolare e della sua utilità [1]. Si parla di dinamica molecolare nel momento in cui si esegue una simulazione su un set di atomi interagenti in cui la loro evoluzione temporale è data dall'integrazione delle equazioni del moto. In dinamica molecolare si sfrutta la seconda legge di Newton:

$$\vec{F}_i = m_i \vec{a}_i \tag{1.1}$$

dove per ogni atomo i di un set di N atomi, $\vec{F}_i$ è il vettore forza che agisce sull'atomo, m_i è la massa e $\vec{a}_i$ è l'accelerazione che ne deriva. La dinamica molecolare è un metodo deterministico, infatti dato un set di N coordinate spaziali e di N momenti, lo sviluppo temporale del modello è unicamente determinato. Il ruolo della simulazione è quindi quello di calcolare per ogni timestep prefissato le coordinate e i momenti del set di atomi nel modello. In altre parole si genera la traiettoria del sistema nello spazio delle fasi di dimensione $6N$.

Ci si potrebbe chiedere come l'integrazione di leggi del moto date dalla meccanica classica siano attendibili se applicate a scale atomiche, dove è risaputo che un contributo è dato anche dalle leggi della meccanica quantistica. Per verificare la validità della dinamica degli atomi bisogna considerare la lunghezza d'onda termica di de Broglie definita come

$$\Lambda = \sqrt{\frac{2\pi\hbar^2}{Mk_BT}} \tag{1.2}$$

dove M è la massa atomica e T è la temperatura. L'approssimazione classica è accettabile quando $\Lambda << a$, dove a è la distanza tra primi vicini. Nella dinamica molecolare si lavora di norma a temperature sufficientemente alte da consentire di trascurare gli effetti quantistici sul moto degli atomi. In questo progetto si lavora su atomi di silicio e ossigeno a temperature relativamente elevate, quindi l'approssimazione classica è sufficiente per descrivere lo sviluppo temporale del nostro modello.

1.2 Modello del sistema

Lo step critico di una simulazione è la scelta del modello da applicare al sistema in questione, che, nel caso della Dinamica Molecolare, sta nel decidere la superficie di energia potenziale che descrive le interazioni tra le particelle. L'energia potenziale è una funzione delle posizioni dei nuclei atomici del sistema: $V(\vec{r}_1, \vec{r}_2, .., \vec{r}_N)$. Questa funzione è invariante sotto trasformazioni di traslazione e rotazione, e spesso sfrutta le posizioni relative degli atomi e non quelle assolute. Dopo la scelta del potenziale, le equazioni delle forze che agiscono sui singoli atomi del sistema sono unicamente ricavate da

$$\vec{F}_i = -\nabla_i V(\vec{r}_1, \vec{r}_2, .., \vec{r}_N) \ , \qquad i = 1, 2, ..., N \tag{1.3}$$

con $\nabla_i = \dfrac{\partial}{\partial \vec{r}_i}$.

La scelta più semplice, ma non sempre la più efficacie, è quella di scrivere l'energia potenziale come somma di interazioni di coppia, cioè da una sommatoria di termini che dipendono solo dalla distanza reciproca tra due atomi.

$$V(\vec{r}_1\vec{r}_2, .., \vec{r}_N) = \frac{1}{2} \sum_i \sum_{i \neq j} \phi\left(|\vec{r}_i - \vec{r}_j|\right) \tag{1.4}$$

$$= \sum_i \sum_{j>i} \phi\left(|\vec{r}_i - \vec{r}_j|\right) \tag{1.5}$$

In entrambi i casi le sommatorie sono costruite in modo da contare una sola volta il contributo dell'interazione tra gli atomi i e j, in un caso moltiplicando per $\frac{1}{2}$ ogni termine e nell'altro prendendo $j > i$. Un esempio di potenziale di coppia è il potenziale Lennard-Jones, dove il termine $\phi\left(|\vec{r}_i - \vec{r}_j|\right) = \phi(r)$ è dato da

$$\phi_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{1.6}$$

Si nota che se $r = \sigma$ il potenziale assume valore nullo, e per distanze maggiori di questa il potenziale è negativo, mentre altrove è positivo.

Si riporta l'andamento qualitativo del potenziale di Lennard-Jones:

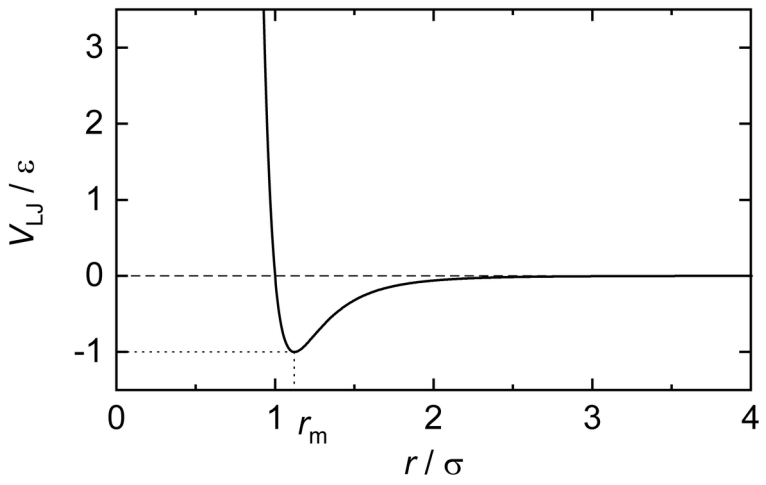

Figura 1.1: Potenziale di Lennard-Jones

Dall'equazione del potenziale possiamo quindi ricavare l'espressione della forza che agisce sull'atomo i nella posizione $\vec{r}_i$ interagente con un atomo j nella posizione $\vec{r}_j$:

$$\vec{F}_i = -\nabla_i \left\{ 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \right\} = 24\epsilon \left[2\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \frac{\vec{r}_{ij}}{r^2} \tag{1.7}$$

con $\vec{r}_{ij} = \vec{r}_j - \vec{r}_i$ e $r = |\vec{r}_{ij}|$.

Ponendo l'equazione della forza uguale a zero si trova il minimo dell'espressione del potenziale, che corrisponde ad una distanza di $r_{min} = \sqrt[6]{2}\sigma$, il quale sostituito nel potenziale di Lennard-Jones, restituisce il valore $\phi_{LJ}(r_{min}) = -\epsilon$. Per valori di $r < r_{min}$ la forza sarà repulsiva, mentre per valori maggiori sarà attrattiva.

Per l'implementazione in una simulazione di dinamica molecolare si utilizzano delle tecniche per ridurre il costo computazionale senza perdere troppa attendibilità dei dati. Una di queste consiste nel tagliare la distanza di interazione tra gli atomi ad una distanza fissata r_{cut}, in modo da considerare solamente gli atomi all'interno della sfera di raggio r_{cut}, e trascurare gli altri. Quindi sarà

$$u(r) = \begin{cases} \phi_{LJ}(r) - \phi_{LJ}(r_{cut}) & r \leq r_{cut} \\ 0 & r > r_{cut} \end{cases} \tag{1.8}$$

In pratica si traspone il potenziale di $\phi_{LJ}(r_{cut})$, per $r \leq r_{cut}$ e si considera nullo il potenziale per distanze maggiori di r_{cut}. Si assicura in questo modo la continuità del potenziale.

1.3 Metodi di integrazione

Dopo aver trovato l'espressione delle forze di interazione, bisogna effettuare l'integrazione delle equazioni del moto. Esistono diversi metodi di integrazione, uno dei più usati è il metodo di Verlet. L'idea di base per ricavare questo metodo è quella di scrivere due espansioni di Taylor

delle posizioni, una avanti e una indietro nel tempo, fermandosi al secondo ordine, cioè:

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 + o(\Delta t^3) \tag{1.9}$$

$$\vec{r}(t-\Delta t) = \vec{r}(t) - \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 + o(\Delta t^3) \tag{1.10}$$

dove $\vec{r}$ è il vettore posizione, $\vec{v}$ è la velocità e $\vec{a}$ è l'accelerazione. Sommando e sottraendo le due equazioni si ottengono le equazioni di Verlet:

$$\vec{r}(t+\Delta t) = 2\vec{r}(t) - \vec{r}(t-\Delta t) + \vec{a}(t)\Delta t^2 + o(\Delta t^4) \tag{1.11}$$

$$\vec{v}(t) = \frac{\vec{r}(t+\Delta t) - \vec{r}(t-\Delta t)}{2\Delta t} \tag{1.12}$$

Visto che stiamo utilizzando le leggi di Newton, l'accelerazione è determinata da esse:

$$\vec{a}(t) = \frac{\vec{F}}{m} = -\frac{\nabla V(\vec{r}(t))}{m} \tag{1.13}$$

Per avanzare le velocità contemporaneamente alle posizioni invece si modifica l'algoritmo di Verlet con due passaggi: si utilizza l'accelerazione al tempo t per incrementare la velocità per un timestep che è metà di Δt, poi si calcola l'accelerazione a $t + \Delta t$ e con essa si incrementa nuovamente la velocità. Successivamente si riassumono gli step per l'implementazione dello schema "velocity" di Verlet

$$\vec{v}(t+\frac{\Delta t}{2}) = \vec{v}(t) + \vec{a}(t)\frac{\Delta t}{2} \tag{1.14}$$

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t+\frac{\Delta t}{2})\Delta t \tag{1.15}$$

$$\vec{a}(t+\Delta t) = -\frac{1}{m}\nabla V(\vec{r}(t+\Delta t)) \tag{1.16}$$

$$\vec{v}(t+\Delta t) = \vec{v}(t+\frac{\Delta t}{2}) + \vec{a}(t+\Delta t)\frac{\Delta t}{2} \tag{1.17}$$

Il metodo di Verlet è facile da implementare, preciso all'ordine Δt^2, invariante per time reversal e richiede una sola valutazione per timestep dell'energia potenziale e delle forze. Per questi motivi è universalmente usato in Dinamica Molecolare.

Capitolo 2

Potenziale di Tersoff

Il potenziale di Tersoff [2], a differenza dei potenziali di coppia, come il potenziale di Lennard-Jones, considera l'azione combinata di più atomi su un altro. Mentre in un potenziale di coppia si sommano le interazioni tra coppie di atomi per ricavare l'azione totale che essi esercitano tra di loro, in un potenziale a più corpi, come quello di Tersoff, si sommano le interazioni tra coppie e terne di atomi, quindi per ogni atomo i ci sarà una sommatoria con indice j sugli n atomi che interagiscono, e per ogni j una sommatoria con indice k su altri $n-1$ atomi. Nei prossimi paragrafi si spiega nel particolare come si ricavano queste interazioni e quali coefficienti si utilizzano per ricavare energia potenziale e forza applicata su ogni atomo.

All'interno delle varie equazioni compariranno dei coefficienti che o dipendono dall'elemento del singolo atomo, nel nostro caso silicio o ossigeno, e quindi avremo due valori differenti, o dipendono dalla coppia di atomi considerata, e di conseguenza saranno tre valori distinti (Si-Si, Si-O, O-O). Il potenziale di Tersoff prende ognuno di questi atomi e, con i parametri appropriati, permette di valutare energia e forza che agisce su uno di essi limitando i calcoli ad una sfera centrata nell'atomo di interesse. Il raggio di questa sfera dipende da quale elemento stiamo considerando e nel nostro caso vale 2.0Å per l'ossigeno e 2.8Å per il silicio. In questo modo il costo computazionale del programma, rispetto ad un potenziale di Lennard-Jones, si riduce di molto perché all'interno della sfera di raggio $2-3$Å si trovano in pratica solo i primi vicini, che, nella configurazione di α-quarzo per SiO_2, sono due per l'ossigeno e quattro per il silicio. Tuttavia il calcolo di ogni termine non è una semplice interazione di coppia, ma considera l'interazione di terne (ovviamente se l'atomo ha più di un primo vicino). Per ogni terna, fissato un atomo, si considerano le distanze tra le due coppie e l'angolo tra esse.

2.1 Condizioni al contorno Born - von Karman

Utilizzando un cristallo finito come campione per la nostra simulazione, si impiegano le cosiddette condizioni al contorno periodiche di Born - von Karman. Queste condizioni vengono ampiamente utilizzate nella fisica dello stato solido, e risultano in una approssimazione di un cristallo infinito. In sostanza si considera il cristallo come una ripetizione infinita nelle tre direzioni del campione considerato. Considerando un bordo del campione quindi, l'interazione è, oltre a quella ricavata dagli atomi interni, data dalle repliche degli atomi che stanno nei pressi della faccia opposta del campione.

Si ha inoltre una migliore approssimazione delle proprietà di bulk più il campione iniziale ha dimensioni maggiori. Bisogna quindi trovare un giusto compromesso tra costo computazionale della simulazione e attendibilità anche nella scelta delle dimensioni del campione.

2.2 Energia Potenziale e Forza con il potenziale di Tersoff

2.2.1 Energia Potenziale

Come è stato detto in precedenza, nel calcolo di energia potenziale e forze per il potenziale di Tersoff [3], bisogna considerare in generale più di una coppia di atomi per volta. Nel caso l'atomo in questione ne abbia solo un altro all'interno della sfera centrata nell'atomo di raggio S_{ij}, che è un parametro del potenziale di Tersoff analogo al cutoff di Lennard-Jones, allora l'interazione è semplicemente di coppia. Si consideri un cristallo composto da N atomi, l'energia potenziale totale del cristallo sarà

$$V = \sum_{i=1}^{N} V_i = \frac{1}{2} \sum_{i;\, j \neq i} V_{ij} \tag{2.1}$$

dove V_i è l'energia di ogni singolo atomo e V_{ij} é il potenziale calcolato per ogni coppia di atomi.

Nel caso del potenziale di Tersoff il contributo V_{ij} è dato dalla seguente equazione:

$$V_{ij} = f_C(r_{ij}) \left[f_R(r_{ij}) + b_{ij} f_A(r_{ij}) \right] \tag{2.2}$$

dove f_C, f_R e f_A sono i termini che dipendono dalla distanza tra gli atomi della coppia ij, mentre in b_{ij} compaiono i termini a più corpi. Le equazioni dei termini di coppia sono:

$$f_A(r_{ij}) = -B_{ij} e^{-\mu_{ij} r_{ij}} \tag{2.3}$$

$$f_R(r_{ij}) = A_{ij} e^{-\lambda_{ij} r_{ij}} \tag{2.4}$$

$$f_C(r_{ij}) = \begin{cases} 1 & r_{ij} < R_{ij} \\ \frac{1}{2} + \frac{1}{2} cos(\pi \frac{r_{ij} - R_{ij}}{S_{ij} - R_{ij}}) & R_{ij} < r_{ij} < S_{ij} \\ 0 & r_{ij} > S_{ij} \end{cases} \tag{2.5}$$

Si fa notare che S_{ij}, il quale ha le dimensioni di una lunghezza, è il raggio della sfera entro la quale consideriamo le interazioni, in quanto, per valori di raggio maggiori di questa lunghezza, il termine nel potenziale di Tersoff $f_C(r_{ij})$, che moltiplica tutti gli altri termini, va a zero. L'espressione di b_{ij} invece è leggermente più complicata, in quanto include le interazioni tra più corpi. La dipendenza da un terzo atomo compare nel contributo $\zeta\left(r_{ij}\right)$, nel quale vi è una sommatoria sugli atomi k (con $k \neq i, j$) all'interno della sfera di raggio S_{ij} considerata. Nell'equazione di b_{ij} troviamo

- χ_{ij} costante che dipende dalla coppia di elementi ij
- n_i e β_i costanti che dipendono dal solo elemento i
- ζ_{ij} in cui compare la sommatoria sugli altri atomi k

Se si guarda il termine ζ_{ij}, all'interno della sommatoria si ritrova f_C, che stavolta dipende dalla distanza tra la coppia ik. Il termine g invece dipende dall'angolo θ_{ijk} compreso tra il vettore che congiunge i e j e quello che congiunge i e k.

$$b_{ij} = \chi_{ij} (1 + \beta_i^{n_i} \zeta_{ij}^{n_i})^{-\frac{1}{2n_i}} \tag{2.6}$$

$$\zeta_{ij} = \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \tag{2.7}$$

Andiamo ora a vedere come è strutturato il termine $g(\theta_{ijk})$. Nell'equazione si notano diversi parametri, quali c_i, d_i e h_i, costanti che dipendono solo dall'elemento i considerato, e il coseno dell'angolo θ_{ijk}, il quale viene scritto come rapporto tra il prodotto scalare dei vettori tra cui è compreso e il prodotto dei moduli.

$$g(\theta_{ijk}) = 1 + \frac{c_i^2}{d_i^2} - \frac{c_i^2}{d_i^2 + (h_i - cos(\theta_{ijk}))^2} \tag{2.8}$$

$$cos(\theta_{ijk}) = \frac{\vec{r}_{ij} \cdot \vec{r}_{ik}}{r_{ik} r_{ij}} \tag{2.9}$$

Date queste equazioni si possiedono tutte le informazioni necessarie per calcolare le forze che agiscono su ogni atomo

2.2.2 Forza

Per il calcolo delle forze si parte dalla definizione

$$\begin{aligned}
\vec{F}_i = -\frac{\partial V}{\partial \vec{r}_i} &= -\sum_j \frac{\partial V_j}{\partial \vec{r}_i} \\
&= -\sum_{j \neq i} \left(\frac{\partial V_j}{\partial \vec{r}_i} \right) - \frac{\partial V_i}{\partial \vec{r}_i} \\
&= -\sum_{j \neq i} \left(\sum_{k \neq j} \frac{\partial V_j}{\partial \vec{r}_{jk}} \frac{\partial \vec{r}_{jk}}{\partial \vec{r}_i} + \frac{\partial V_i}{\partial \vec{r}_{ij}} \frac{\partial \vec{r}_{ij}}{\partial \vec{r}_i} \right) \\
&= \sum_{j \neq i} \left(\frac{\partial V_i}{\partial \vec{r}_{ij}} - \frac{\partial V_j}{\partial \vec{r}_{ji}} \right)
\end{aligned} \tag{2.10}$$

Da cui segue che

$$\vec{F}_{ij} = -\vec{F}_{ji} = \frac{\partial V_i}{\partial \vec{r}_{ij}} - \frac{\partial V_j}{\partial \vec{r}_{ji}} \tag{2.11}$$

Consideriamo per il momento il termine $\dfrac{\partial V_i}{\partial \vec{r}_{ij}}$ (l'altro termine è determinato di conseguenza invertendo i e j)

$$\frac{\partial V_i}{\partial \vec{r}_{ij}} = \frac{1}{2} \frac{\partial V_{ij}}{\partial \vec{r}_{ij}} + \frac{1}{2} \sum_{k \neq i,j} \frac{\partial V_{ik}}{\partial \vec{r}_{ij}} \tag{2.12}$$

In questo modo si divide quindi il calcolo in due step: uno in cui si considera la derivata parziale del potenziale V_{ij}, e nell'altro la somma della derivata degli altri termini V_{ik}. Isolando il primo e ricordando che

$$V_{ij} = f_C(r_{ij}) \left[f_R(r_{ij}) + b_{ij} f_A(r_{ij}) \right] \tag{2.13}$$

si ricava la seguente espressione

$$\begin{aligned}\frac{\partial V_{ij}}{\partial \vec{r}_{ij}} =& f_C'(r_{ij})\left[f_R(r_{ij}) + b_{ij} f_A(r_{ij})\right]\frac{\vec{r}_{ij}}{r_{ij}} + \\ &+ f_C(r_{ij})\left[f_R'(r_{ij}) + b_{ij} f_A'(r_{ij})\right]\frac{\vec{r}_{ij}}{r_{ij}} + \\ &+ f_C(r_{ij}) f_A(r_{ij})\frac{\partial b_{ij}}{\partial \vec{r}_{ij}}\end{aligned} \tag{2.14}$$

In cui si è indicato

$$f_C'(r_{ij}) = \frac{\partial f_C(r_{ij})}{\partial r_{ij}} \tag{2.15}$$

$$f_A'(r_{ij}) = \frac{\partial f_A(r_{ij})}{\partial r_{ij}} \tag{2.16}$$

$$f_R'(r_{ij}) = \frac{\partial f_R(r_{ij})}{\partial r_{ij}} \tag{2.17}$$

Dalla (2.6) si ricava che

$$\frac{\partial b_{ij}}{\partial \vec{r}_{ij}} = -\frac{1}{2}\chi_{ij}\beta_i^{n_i}\zeta_{ij}^{n_i-1}\left(1+\beta_i^{n_i}\zeta_{ij}^{n_i}\right)^{-\frac{1}{2n_i}-1}\frac{\partial \zeta_{ij}}{\partial \vec{r}_{ij}} = b_{ij}'\frac{\partial \zeta_{ij}}{\partial \vec{r}_{ij}} \tag{2.18}$$

$$\frac{\partial \zeta_{ij}}{\partial \vec{r}_{ij}} = \sum_{k\neq i,j}\left[\frac{\partial f_C(r_{ik})}{\partial \vec{r}_{ij}} g(\theta_{ijk}) + f_C(r_{ik})\frac{\partial g(\theta_{ijk})}{\partial \vec{r}_{ij}}\right] \tag{2.19}$$

Nella (2.19) il primo addendo della sommatoria è nullo poiché f_C dipende da r_{ik}, e facendo la derivata parziale rispetto a $\vec{r}_{ij}$ si ottiene un termine nullo essendo $k \neq j$, mentre in $g(\theta_{ijk})$ compare il coseno di θ_{ijk}, il quale dipende esplicitamente da $\vec{r}_{ij}$. Rimane quindi

$$\frac{\partial \zeta_{ij}}{\partial \vec{r}_{ij}} = \sum_{k\neq i,j} f_C(r_{ik})\frac{\partial g(\theta_{ijk})}{\partial \vec{r}_{ij}} \tag{2.20}$$

Dalla (2.8)

$$\frac{\partial g(\theta_{ijk})}{\partial \vec{r}_{ij}} = -\frac{2c_i^2(h_i - \cos(\theta_{ijK})}{\left[d_i^2 + (h_i - \cos(\theta_{ijk}))^2\right]^2}\frac{\partial \cos(\theta_{ijk})}{\partial \vec{r}_{ij}} = g'(\theta_{ijk})\frac{\partial \cos(\theta_{ijk})}{\partial \vec{r}_{ij}} \tag{2.21}$$

Dove nella (2.21) si è scritto $g'(\theta_{ijk}) = \frac{\partial g(\theta_{ijk})}{\partial \cos(\theta_{ijk})}$, mentre la derivata parziale del coseno è facilmente ricavata dalla (2.9)

$$\frac{\partial \cos(\theta_{ijk})}{\partial \vec{r}_{ij}} = \frac{1}{r_{ij}}\left[\frac{\vec{r}_{ik}}{r_{ik}} - \cos(\theta_{ijk})\frac{\vec{r}_{ij}}{r_{ij}}\right] \tag{2.22}$$

Quindi mettendo insieme tutte le equazioni ricavate si ottiene:

$$\begin{aligned}\frac{\partial V_{ij}}{\partial \vec{r}_{ij}} =& f_C'(r_{ij})\left[f_R(r_{ij}) + b_{ij} f_A(r_{ij})\right]\frac{\vec{r}_{ij}}{r_{ij}} + \\ &+ f_C(r_{ij})\left[f_R'(r_{ij}) + b_{ij} f_A'(r_{ij})\right]\frac{\vec{r}_{ij}}{r_{ij}} + \\ &+ \sum_{k\neq i,j} f_C(r_{ij}) f_A(r_{ij}) b_{ij}' f_C(r_{ik}) g'(\theta_{ijk})\frac{\partial \cos(\theta_{ijk})}{\partial \vec{r}_{ij}}\end{aligned} \tag{2.23}$$

Consideriamo ora il secondo termine della forza: $\sum_{k\neq i,j}\frac{\partial V_{ik}}{\partial \vec{r}_{ij}}$. Se consideriamo le derivate parziali di $f_C(r_{ik})$, $f_A(r_{ik})$ e $f_R(r_{ik})$ rispetto a $\vec{r}_{ij}$, essi sono nulli perché dipendono da r_{ij}. Rimane quindi un solo termine che è

$$\sum_{k\neq i,j}\frac{\partial V_{ik}}{\partial \vec{r}_{ij}} = \sum_{k\neq i,j} f_C(r_{ik}) f_A(r_{ik})\frac{\partial b_{ik}}{\partial \vec{r}_{ij}} \tag{2.24}$$

poiché, come si vedrà, avrà dei termini non nulli. Infatti isolando la derivata parziale si ottiene

$$\frac{\partial b_{ik}}{\partial \vec{r}_{ij}} = b_{ik}'\frac{\partial \zeta_{ik}}{\partial \vec{r}_{ij}} \tag{2.25}$$

$$= \sum_{l\neq i,k}\left[b_{ik}' f_C'(r_{il}) g(\theta_{ikl})\frac{\vec{r}_{ij}}{r_{ij}} + b_{ik}' f_C(r_{il}) g'(\theta_{ikl})\frac{\partial \cos(\theta_{ikl})}{\partial \vec{r}_{ij}}\right] \tag{2.26}$$

dove

$$b_{ik}' = \frac{\partial b_{ik}}{\partial \zeta_{ik}} \tag{2.27}$$

$$f_C'(r_{il}) = \frac{\partial f_C(r_{il})}{\partial r_{ij}} = f_C'(r_{ij})\delta_{jl} \tag{2.28}$$

$$\frac{\partial \cos(\theta_{ikl})}{\partial \vec{r}_{ij}} = \frac{\partial \cos(\theta_{ikj})}{\partial \vec{r}_{ij}}\delta_{jl} \tag{2.29}$$

Le delta mi condensano la sommatoria ad un unico termine

$$\sum_{l\neq i,k}\left[f_C'(r_{il}) g(\theta_{ikl})\frac{\vec{r}_{ij}}{r_{ij}} + f_C(r_{il}) g'(\theta_{ikl})\frac{\partial \cos(\theta_{ikl})}{\partial \vec{r}_{ij}}\right] = \sum_{k\neq i,j}\left[f_C'(r_{ij}) g(\theta_{ikj})\frac{\vec{r}_{ij}}{r_{ij}} + f_C(r_{ij}) g'(\theta_{ikj})\frac{\partial \cos(\theta_{ikj})}{\partial \vec{r}_{ij}}\right] \tag{2.30}$$

Inoltre $\cos(\theta_{ijk}) = \cos(\theta_{ikj})$, e, di conseguenza, $g(\theta_{ijk}) = g(\theta_{ikj})$. Quindi riscrivendo la (2.24) espandendo $\frac{\partial b_{ik}}{\partial \vec{r}_{ij}}$ si ottiene

$$\sum_{k\neq i,j}\frac{\partial V_{ik}}{\partial \vec{r}_{ij}} = \sum_{k\neq i,j} f_C(r_{ik}) f_A(r_{ik}) b_{ik}'\left[f_C'(r_{ij}) g(\theta_{ijk})\frac{\vec{r}_{ij}}{r_{ij}} + f_C(r_{ij}) g'(\theta_{ijk})\frac{\partial \cos(\theta_{ijk})}{\partial \vec{r}_{ij}}\right] \tag{2.31}$$

In conclusione si ottiene l'equazione con i termini estesi per la (2.10)

$$\begin{aligned}\frac{\partial V_i}{\partial \vec{r}_{ij}} =& \frac{1}{2} f_C'(r_{ij}) \left[f_R(r_{ij}) + b_{ij} f_A(r_{ij}) \right] \frac{\vec{r}_{ij}}{r_{ij}} \\ &+ \frac{1}{2} f_C(r_{ij}) \left[f_R'(r_{ij}) + b_{ij} f_A'(r_{ij}) \right] \frac{\vec{r}_{ij}}{r_{ij}} \\ &+ \frac{1}{2} \sum_{k \neq i,j} f_C(r_{ik}) f_C'(r_{ij}) f_A(r_{ik}) b_{ik}' g(\theta_{ijk}) \frac{\vec{r}_{ij}}{r_{ij}} \\ &+ \frac{1}{2} \sum_{k \neq i,j} f_C(r_{ik}) f_C(r_{ij}) g'(\theta_{ijk}) \frac{\partial \cos(\theta_{ijk})}{\partial \vec{r}_{ij}} \left[f_A(r_{ij}) b_{ij}' + f_A(r_{ik}) b_{ik}' \right]\end{aligned} \tag{2.32}$$

2.3 Coefficienti e unità di misura

Per calcolare energia potenziale e forze usando il potenziale di Tersoff è necessario considerare i coefficienti per la specifica coppia di elementi del composto, nel nostro caso silicio e ossigeno. Si riporta successivamente la tabella dei coefficienti utilizzati nel calcolo dei termini che compaiono nelle espressioni di energia e forza. I coefficienti sono stati presi da Munetoh *et al.* [4]

	O	Si	Units
A	1.88255e3	1.8308e3	eV
B	2.18787e2	4.7118e2	eV
lambda	41.7108	24.799	1/nm
mu	23.5692	17.322	1/nm
beta	1.1632e-7	1.1e-6	
n	1.04968	7.8734e-1	
c	6.46921e4	1.0039e5	
d	4.11127	1.6217e1	
h	-8.45922e-1	-5.9825e-1	
R	.17	.25	nm
S	.20	.28	nm
X(Si-O)	1.17945		

Alcuni coefficienti (A, B, λ, μ, R, S) dipendono dalla coppia di atomi (O-O, O-Si, Si-Si), gli altri invece dal singolo atomo (O, Si). Sullo stesso articolo si trovano le equazioni per ricavare la terna di valori dei coefficienti che dipendono dalla coppia di atomi

$$A_{ij} = \sqrt{A_i A_j} \tag{2.33}$$

$$\lambda_{ij} = \frac{\lambda_i + \lambda_j}{2} \tag{2.34}$$

$$R_{ij} = \sqrt{R_i R_j} \tag{2.35}$$

$$B_{ij} = \sqrt{B_i B_j} \tag{2.36}$$

$$\mu_{ij} = \frac{\mu_i + \mu_j}{2} \tag{2.37}$$

$$S_{ij} = \sqrt{S_i S_j} \tag{2.38}$$

dove i e j sono gli elementi interessati nella coppia. I coefficienti che dipendono invece dal singolo atomo sono unicamente ricavati dalla tabella. Essi inoltre sono tutti adimensionali, mentre gli altri, quelli che dipendono dalla coppia, hanno le diverse dimensioni specificate.

Capitolo 3

Programma Python

3.1 Costruzione del cristallo

Per prima cosa si decide che tipo di campione utilizzare per costruire la condizione iniziale. Nel nostro caso si è scelto un cristallo di α-quarzo composto da silicio e ossigeno. Le informazioni che servono per generare le posizioni di un cristallo sono i vettori di una cella di base (o unitaria o non) e il numero di celle che si vogliono riprodurre nelle tre direzioni. Per fare ciò è stata costruita una funzione che prende come input un file di testo basato sullo standard *".xyz"* in cui viene scritto, in ordine, il numero di atomi di base ($n_{vettori}$), le dimensioni della cella di base da ripetere, e poi $n_{vettori}$ righe in cui si scrivono il tipo di atomo e le coordinate del vettore posizione dell'atomo all'interno della cella di base. Si riporta successivamente il file di input della funzione [5],

```
18
4.9134, 8.51025844, 5.4052
1,       1.1544,      6.5108,      3.6035
1,       1.1544,      1.9995,      1.8017
1,       3.6111,      2.2556,      3.6035
1,       3.6111,      6.2546,      1.8017
1,       0.1479,      4.2551,      0.0000
1,       2.6046,      0.0000,      0.0000
0,       1.6760,      7.8890,      4.2452
0,       1.6760,      0.6212,      1.1600
0,       4.1327,      3.6339,      4.2452
0,       2.1567,      6.0172,      2.4435
0,       1.0807,      3.1143,      0.6418
0,       4.1327,      4.8764,      1.1600
0,       1.0807,      5.3959,     -0.6418
0,       2.1567,      2.4931,      2.9617
0,       3.5374,      7.3695,      0.6418
0,       4.6134,      6.7482,      2.9617
0,       4.6134,      1.7620,      2.4435
0,       3.5374,      1.1408,     -0.6418
```

dove si identificano con il tipo 0 gli atomi di ossigeno e con il tipo 1 quelli di silicio. Il file di testo viene letto nella funzione `read_vec_file` che prende in input il nome del file e memorizza tutti i dati necessari. Questa tuttavia non è la cella di base unitaria, ma per comodità si è presa la cella di base a geometria a parallelepipedo, ripetibile periodicamente per la configurazione α-quarzo, con facce ortogonali e volume doppio della cella cristallografica. Basterà quindi replicare questa cella nelle tre direzioni per costruire la super-cella con il campione iniziale. La funzione per la lettura dei parametri di input del cristallo è riportata di seguito

```
  def read_vec_file(self, filename='alphaquartz.csv'):
      from numpy import zeros, loadtxt, int32
      self.nvec       = loadtxt(filename, delimiter=',', max_rows=1, dtype=int32)
      self.xvector    = zeros(self.nvec)
      self.yvector    = zeros(self.nvec)
      self.zvector    = zeros(self.nvec)
      self.spv        = zeros(self.nvec)
      self.ax,self.by,self.cz = loadtxt(filename, delimiter=',', skiprows=1, max_rows=1)
      self.spv[:],self.xvector[:],self.yvector[:],self.zvector[:]= loadtxt(filename, delimiter=',',
    skiprows=2, unpack=True)
```

Il numero di ripetizione della cella di base viene invece scelto inserendo tre interi in tre rispettive variabili all'interno del programma, ognuna per ogni direzione nel nostro sistema di riferimento. Queste variabili sono definite nel file principale `run_test.py` come `mx`, `my` e `mz`. Si riporta successivamente la funzione che costruisce la super-cella del cristallo, dati i vettori del file in input di componenti `xvector`, `yvector`, `zvector`, il numero di vettori nella cella singola `self.nvec` e le variabili che contengono il numero di celle, qua chiamate `lx`, `ly` e `lz`.

```
j   = 0
for nx in arange(lx) :
   for ny in arange(ly) :
      for nz in arange(lz) :
          for vec in arange(self.nvec) :
              self.x[j] = xi + ax*nx + rrx[j] + self.xvector[vec]
              self.y[j] = yi + ay*ny + rry[j] + self.yvector[vec]
              self.z[j] = zi + az*nz + rrz[j] + self.zvector[vec]
              self.sp[j]= self.spv[vec]
              if self.sp[j] == 0:
                  self.m[j] = self.ma
                  self.Na += 1
              else:
                  self.m[j] = self.mb
                  self.Nb += 1
              j +=1
```

Per ogni atomo j si memorizzano quindi posizioni, specie (silicio o ossigeno) e massa in unità di massa atomica, indicando con `self.ma` la massa dell'Ossigeno e con `self.mb` la massa del Silicio. Si memorizzano anche due contatori (`self.Na`, `self.Nb`) che contengono rispettivamente il numero di atomi per ossigeno e silicio.

3.2 Ottimizzazione per le celle

Per minimizzare il numero di coppie di atomi prelevato in ogni ciclo, si divide la super-cella con il campione in celle identiche di dimensione minime, in modo che, prendendo la cella in cui si trova l'atomo i, e le altre 26 celle vicine, intorno a questa, la sfera si raggio S_{ij}, qualunque sia l'atomo che considero, sia tutta contenuta dentro a questa regione. Si fa quindi in modo che le celle costruite abbiano il lato minore il più vicino possibile, e, comunque, maggiore del raggio della sfera. In questo modo basta considerare, per ogni atomo della cella in questione, gli atomi contenuti in 27 celle, i quali sono in numero estremamente ridotto essendo $S_{ij} < 3$Å indipendentemente dalla coppia di atomi considerata.

Ad esempio considerando una super-cella, per semplicità cubica, di lato 15Å con $N = 250$ atomi e prendendo come raggio della sfera $S_{ij} = 3$Å si ottengono $5 \times 5 \times 5 = 125$ celle con una media di $n = 250/125 = 2$ atomi per cella. Quindi per calcolare l'energia e la forza che

agisce su un atomo di una cella mi basta considerare per ognuno di essi in media $27 * 2 - 1 = 53$ coppie, il che riduce di molto il costo computazionale del programma. Si riporta un'immagine per una maggiore comprensione del metodo, che, per semplicità grafica, è stata proiettata in due dimensioni. Viene evidenziata in rosso la cella di interesse, in viola il raggio S_{ij} della sfera, e in verde le quattro circonferenze agli estremi della cella.

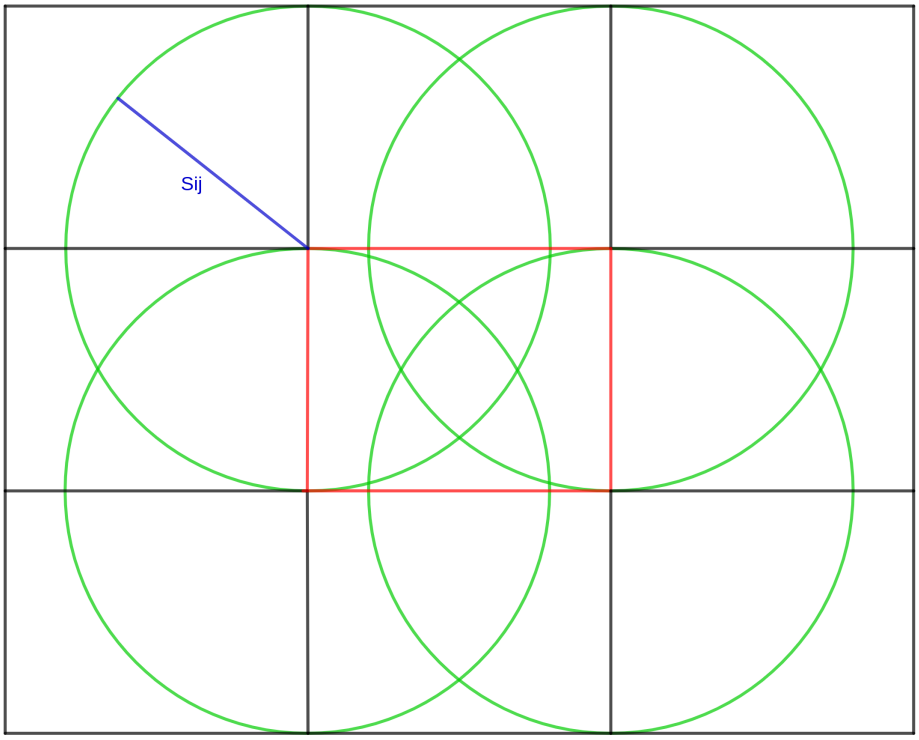

Figura 3.1: Configurazione delle celle

Come si può notare, qualunque circonferenza di raggio S_{ij} centrata in un punto della cella rossa è completamente contenuta all'interno delle nove celle, e quindi considerando solo queste è possibile comprendere le interazioni tra gli atomi della sezione rossa con tutti quelli che interagiscono. Una cella di dimensioni minori comporterebbe delle circonferenze che eccedono l'area ottenuta dalle nove celle; una cella maggiore invece porterebbe ad un costo computazionale maggiore e non necessario. Tuttavia il nostro sistema ha delle dimensioni ben definite, e quindi dividendo ogni lato del campione per il rispettivo lato della cella nelle tre coordinate, devo ottenere necessariamente un intero, in modo che le celle abbiano tutte dimensioni identiche. Si otterranno quindi dei lati per le celle leggermente maggiori del raggio della sfera, ma divisori interi dei lati della super-cella con il campione.

Prima di procedere a calcolare energie e forze è necessario un riordinamento delle celle, in modo da semplificare la scrittura successiva del codice. Questo viene fatto creando un array (`indc`) di dimensione pari al numero di atomi, nel quale si memorizza a quale cella appartiene l'atomo i-esimo, mediante le seguenti istruzioni:

```
for i in range(N):
    vcx=int(mx*(rx[i]+0.5)) # indice nella coordinata x della cella a cui appartiene l'atomo i-esimo
    vcy=int(my*(ry[i]+0.5)) # indice nella coordinata y della cella a cui appartiene l'atomo i-esimo
    vcz=int(mz*(rz[i]+0.5)) # indice nella coordinata z della cella a cui appartiene l'atomo i-esimo
    c = mz*(my*vcx+vcy)+vcz # indice della cella
    indc[i]=c               # vettore in cui si memorizza a quale cella appartiene l'atomo
    np[c] += 1              # numero di particelle nella cella c
```

dove `rx`, `ry` e `rz` sono vettori contenenti le coordinate normalizzate a uno di ogni atomo. Si crea poi un array `indp` di dimensione pari al numero di celle, in cui si memorizza il numero degli atomi nella cella sommato al numero di atomi nella cella precedente, cioè

$$indp[i+1] - indp[i] = np[i] \tag{3.1}$$

che viene semplicemente implementato nel seguente modo:

```
indp[0] = 0.
for c in range(0,ncells) :
    indp[c+1] = indp[c] + np[c]
```

Successivamente si creano i vettori che contengono le posizioni effettive delle celle moltiplicando semplicemente i vettori normalizzati per il lato del campione nella rispettiva coordinata.

Fatto questo gli atomi nella cella c sono quelli tra $indp[c]$ e $indp[c+1]$, che saranno infatti gli estremi del ciclo per il calcolo dell'energia e delle forze. Questa assegnazione degli atomi andrà fatta per ogni timestep, poiché essi muovendosi potrebbero andare andare a trovarsi in una cella limitrofa alla precedente. L'ordinamento delle celle è stato arbitrariamente scelto, in un sistema di riferimento xyz, attribuendo ad ogni cella il valore $c = m_z\left(m_y vcx + vcy\right) + vcz$, dove m_y e m_z sono rispettivamente il numero di celle nelle direzioni y e z, mentre vcx, vcy e vcz sono interi che indicano le coordinate della cella nelle tre direzioni del sistema di riferimento. Considerando quindi l'esempio precedente in cui si hanno $N = 125$ celle, cinque in ogni direzione, l'indice della cella alla posizione (2,3,1) sarà $c = 5\left(5 \times 2 + 3\right) + 1 = 66$. Questa scelta implica quindi che spostandoci lungo z, c aumenterà di una unità per ogni cella percorsa; lungo y c incrementerà di 5 mentre lungo x di 25.

3.3 Calcolo di energia potenziale e forze

Dopo aver costruito la suddivisione in celle ci si pone come obiettivo di scrivere un codice efficiente che calcoli i valori dell'energia potenziale e delle forze per ogni atomo nel campione. Si scelgono anche le unità di misura per essere sicuri di non incorrere in overflow e underflow. In questo caso si usa come unità di tempo $10^{-14}s = 10fs$, di lunghezza 1Å e di massa l'unità di massa atomica. Per la temperatura invece si usa il corrispettivo dell'energia termica in eV, quindi per esempio $T = 300$°K corrispondono a circa 0,03 eV in energia termica, se moltiplicata per la costante di Boltzmann.

Con la costruzione del cristallo effettuata precedentemente risulta semplice implementare i cicli sulle celle: dopo aver individuato l'atomo i si esegue un ciclo sulle celle limitrofe e sulla stessa cella dell'i-esimo atomo, facendo semplicemente un ciclo `for` che aggiunga o sottragga una unità all'indice della cella centrale. Ad esempio se l'atomo i si trova nella cella di indici (2, 4, 3), le celle saranno univocamente trovate: (1, 3, 2), (1, 3, 3), (1, 3, 4), (1, 4, 2), (1, 4, 3), (1, 4, 4), ... Per fare ciò si costruiscono 3 vettori `vcx1`, `vcy1` e `vcz1` ognuno di dimensione 27, dove si memorizzano tutte le combinazioni con ripetizione di (-1,0,1). Questo viene fatto attraverso il triplo ciclo riportato successivamente

```
k = 0
for i in range(-1,2):
    for j in range(-1,2):
        for l in range(-1,2):
            vcx1[k] = i
            vcy1[k] = j
            vcz1[k] = l
            k += 1
```

Poi individuata la cella di coordinate `vcx`, `vcy` e `vcz` e l'i-esimo atomo nella cella, si va a fare un ciclo di 27 iterazioni (con indice `k`) per individuare le celle limitrofe a quella considerata, semplicemente sommando le coordinate `vcx + vcx1[k]`. Dopo aver individuato la cella vicina si applicano le condizioni al contorno di Born - von Karman, poiché essa potrebbe essere esterna al campione, cioè, potrebbe avere coordinate pari a -1 o maggiori del numero di celle in quella direzione. Si riporta la condizione al contorno per la sola coordinata x, le altre due direzioni sono analoghe.

```
for k in range(27) :
    wcx=vcx + vcx1[k]

    # Periodic boundary conditions
    shiftx = 0.
    if (wcx == -1) :
        shiftx =-Lx
        wcx = mx-1
    elif (wcx==mx) :
        shiftx = Lx
        wcx = 0
```

La cella dopo le condizioni al contorno avrà indice $c_1 = m_z(m_y wcx + wcy) + wcz$, e il ciclo successivo sarà effettuato sugli atomi della cella c_1.

All'interno del ciclo sulla cella c_1 si controlla se gli atomi stanno dentro alla sfera di raggio S_{ij} centrata nell'atomo i, e allo stesso tempo si mette anche un controllo che permetta di non

considerare lo stesso atomo i. In questo modo si ha un filtraggio degli atomi, considerando solo quelli interessati dal potenziale di Tersoff, permettendo, nei calcoli successivi, di considerarne il minore numero possibile. Successivamente si calcolano le grandezze che verranno riutilizzate, in modo da non dover ricavare più volte questi valori, e si memorizzano all'interno di vettori di dimensione massima 10, poiché ci aspettiamo che gli atomi interagenti siano di norma quattro nelle fasi del composto SiO_2, con piccole variazioni.

```
q = 0
# ciclo su k
    ...
    for j in range(indp[c1],indp[c1+1]):
        dx[q]   = rcx[i]-(rcx[j] + shiftx)
        dy[q]   = rcy[i]-(rcy[j] + shifty)
        dz[q]   = rcz[i]-(rcz[j] + shiftz)
        r2      = dx[q]*dx[q] + dy[q]*dy[q] + dz[q]*dz[q]
        spq[q]  = s1p[j]
        index_ij= spq[q]+spi
        if r2 <= S2[index_ij] and r2 > 0.1:
            atom[q] = j
            r[q]    = sqrt(r2)
            rr[q]   = 1./r[q]
            dr[q]   = array([dx[q], dy[q], dz[q]])*rr[q]
            fA[q]   = -B[index_ij]*exp(-mu[index_ij]*r[q])
            fR[q]   = A[index_ij]*exp(-lam[index_ij]*r[q])
            dfA[q]  = -mu[index_ij]*fA[q]
            dfR[q]  = -lam[index_ij]*fR[q]
            if r2 > R2[index_ij]:
                a       = pRSr[index_ij]*(r[q]-R[index_ij])
                fc[q]   = 0.5 + 0.5*cos(a)
                dfc[q]  = -0.5*sin(a)*pRSr[index_ij]
            else:
                fc[q]   = 1.
                dfc[q]  = 0.
            q+=1
```

Il questo modo avremo, alla fine del ciclo sulle 27 celle, dei vettori di dimensione q che contengono le grandezze di interesse. La dimensione dei vettori è sempre dieci, ma i cicli successivi saranno fatti su q valori, in modo da prendere solo le grandezze che interessano l'atomo i considerato. Si ha quindi cura di azzerare il valore di q ogniqualvolta si cambi atomo i, poiché cambiano, in generale, gli atomi interagenti. Quindi, individuati i q atomi, i cicli successivi saranno solo su questi, migliorando di molto le prestazioni del programma. Si è inoltre deciso di calcolare, fissati gli atomi i e j, $\frac{\partial V_i}{\partial \vec{r}_{ij}}$ e poi sommare questo contributo sia su $\vec{F}_{ij}$ che su $\vec{F}_{ji}$, con i rispettivi segni, poiché

$$\vec{F}_{ij} = \frac{\partial V_i}{\partial \vec{r}_{ij}} - \frac{\partial V_j}{\partial \vec{r}_{ji}} \tag{3.2}$$

$$\vec{F}_{ji} = \frac{\partial V_j}{\partial \vec{r}_{ji}} - \frac{\partial V_i}{\partial \vec{r}_{ij}} \tag{3.3}$$

in modo da semplificare la scrittura del codice senza incidere sulle performance.

Per calcolare energia e forza agente sull'atomo i, utilizzando le equazioni del potenziale di Tersoff che, come detto in precedenza, è un potenziale *many-body*, si ha la necessità di effettuare un ciclo sui q atomi, indicizzati con $j \neq i$, all'interno della sfera, e per ognuno di essi un ciclo sui $q-1$ atomi $k \neq i, j$. Il calcolo che si fa all'interno del ciclo su k è per i termini che nella (2.32) compaiono all'interno della sommatoria, e per ricavare il valore di ζ_{ij}, contenuto in b_{ij}. Si fa notare

che si riscontra la necessità di effettuare un ciclo per ogni atomo k per il valore di ζ_{ik} che compare in b'_{ik} della (2.32). Il ciclo sarà con indice $l \neq i, k$ sempre su $q - 1$ atomi.

Nel caso si voglia approfondire la routine che calcola energia e forza dell'atomo i, si riporta successivamente la parte di programma che la ricava.

```
#ciclo su k
    gThetai = 1. + c2*dr2
    for next_j1 in range(q):
        next_j  = next_j1 % q
        index_ij= spq[next_j] + spi
        f3  = 0.
        f4  = zeros(3)
        f5  = zeros(3)
        # Calcolo preliminare di zetaij per bij
        zetaij  = 0.

        for next_k1 in range(next_j+1, next_j+q):
            next_k  = next_k1 % q
            # cos(thetaijk)
            rij_Scalar_rik          = dx[next_j]*dx[next_k]+dy[next_j]*dy[next_k]+dz[next_j]*dz[next_k]
            rrij_rrik               = rr[next_j]*rr[next_k]
            cosThetaijk[next_k]     = rij_Scalar_rik * rrij_rrik
            h_cosThetaijk[next_k]   = h-cosThetaijk[next_k]
            gThetaijk_den           = 1./(d2  +  h_cosThetaijk[next_k] * h_cosThetaijk[next_k])
            gThetaijk[next_k]       = gThetai - c2*gThetaijk_den
            zetaij                 += fc[next_k]*gThetaijk[next_k]
            dgThetaijk[next_k]      = -2.*c2*h_cosThetaijk[next_k] * gThetaijk_den*gThetaijk_den
            dCosThetaijk[next_k]    = rr[next_j] * (dr[next_k] - dr[next_j] * cosThetaijk[next_k])

        bZetaijn    = 1.+betan*(zetaij**n)
        bij         = X[index_ij]*(bZetaijn)**n2r
        dbij        = -0.5*X[index_ij]*(bZetaijn**(n2r-1.))*betan*(zetaij**(n-1.))
        for next_k1 in range(next_j+1, next_j+q):
            next_k  = next_k1 % q
            index_ik= spq[next_k] + spi
            # Calcolo di zetaik
            zetaik = 0.
            for next_l1 in range(next_k+1, next_k+q):
                next_l  = next_l1 % q
                rik_Scalar_ril  = dx[next_k]*dx[next_l] + dy[next_k]*dy[next_l] + dz[next_k]*dz[next_l]
                rrik_rril       = rr[next_k]*rr[next_l]
                cosThetaikl     = rik_Scalar_ril * rrik_rril
                h_cosThetaikl   = h-cosThetaikl
                gThetaikl_den   = 1./(d2  +  h_cosThetaikl * h_cosThetaikl)
                gThetaikl       = gThetai - c2*gThetaikl_den
                zetaik         += fc[next_l]*gThetaikl
            bZetaikn    = 1.+betan*(zetaik**n)
            dbik        = -0.5*X[index_ik]*(bZetaikn**(n2r-1.))*betan*(zetaik**(n-1.))
            f34 = fc[next_k] * fA[next_k] * dbik
            f3 += dfc[next_j] * gThetaijk[next_k] * f34
            f4 += fc[next_j] * dgThetaijk[next_k] * dCosThetaijk[next_k] * f34
            f5 += fc[next_k] * fc[next_j] * fA[next_j] * dbij * dgThetaijk[next_k] * dCosThetaijk[next_k]
        ei     += fc[next_j]*(fR[next_j] + bij*fA[next_j])
        f1      = dfc[next_j]*(fR[next_j] + bij*fA[next_j])
        f2      = fc[next_j]*(dfR[next_j] + bij*dfA[next_j])

        fTot    = 0.5*((f1 + f2 + f3)*dr[next_j] + f4 + f5)
        fx[indcp[atom[next_j]]] += fTot[0]
        fy[indcp[atom[next_j]]] += fTot[1]
        fz[indcp[atom[next_j]]] += fTot[2]
        fx[indcp[i]] -= fTot[0]
        fy[indcp[i]] -= fTot[1]
        fz[indcp[i]] -= fTot[2]
    # Fine ciclo su next_j
# Fine ciclo sulle celle (k)
e1xx[indcp[i]] += ei
```

Per verificare che il codice funzioni correttamente, un controllo utile nella fase di debug è, prendendo il cristallo a temperatura ambiente, controllare che ogni atomo oscilli attorno alla posizione iniziale mantenendo energia totale costante e bassa energia cinetica. Si riporta il file di output per il cristallo a 300°K ogni 100 timestep di 0,5fs

```
# starting eqmd trajectory
    'pas'    'enep'    'enek'      'enet'      'vcm'
    0.000   -2.8578
# velocities sampled from maxwell distribution at timestep 0
    1.000   -2.8391    0.0230 -2.8160889 -6.2e-15 -5.2e-14       0
    2.000   -2.8327    0.0184 -2.8142476 -2.7e-14  -6e-14 7.1e-15
    3.000   -2.8338    0.0195 -2.8143287  -2e-14 -7.1e-14 1.8e-14
    4.000   -2.8381    0.0287 -2.8093988 -1.8e-14 -6.3e-14 7.1e-15
    5.000   -2.8314    0.0248 -2.8066545 -2.5e-14 -7.8e-14 4.6e-14
    6.000   -2.8159    0.0238 -2.7920491 -3.9e-14 -9.2e-14 5.5e-14
    7.000   -2.8223    0.0197 -2.8025724 -4.1e-14 -8.9e-14 3.2e-14
    8.000   -2.8186    0.0227 -2.7959550  -4e-14 -1.1e-13 4.5e-14
    9.000   -2.8108    0.0233 -2.7875773 -3.6e-14 -1.1e-13 3.4e-14
   10.000   -2.8228    0.0250 -2.7978622 -4.6e-14 -1.1e-13 4.4e-14
# ending eqmd trajectory  <ep>=-2443.6  <ek>=20.3482   T=   0.016  P=   0.000
```

3.4 Calcolo della funzione di distribuzione radiale

Per il calcolo della funzione di distribuzione radiale $g(r)$ usiamo una routine che conta il numero di atomi per ciascuna coppia. I valori ottenuti saranno poi normalizzati nella funzione che scrive su file di testo i valori di $g(r)$

```
def calcgdr(self, N, rx, ry, rz):
    from numpy import sqrt, rint, zeros, int32, pi
    for k in range(N-1) :
        j=k+1
        dx = rx[k]-rx[j:N]
        dy = ry[k]-ry[j:N]
        dz = rz[k]-rz[j:N]
        dx-= rint(dx)
        dy-= rint(dy)
        dz-= rint(dz)
        dx = dx*self.Lx
        dy = dy*self.Ly
        dz = dz*self.Lz
        irdf = zeros(N-j, dtype=int32)
        irdf[:] = self.sp[j:N]+self.sp[k]
        r2 = dx*dx + dy*dy + dz*dz
        b = r2 < self.r2max
        lm  = sqrt(r2[b])
        ind = irdf[b]
        for j in range(len(lm)) :
            self.gcount[int(lm[j]/self.ldel),ind[j]]+=2.
```

Questo calcolo è molto importante, poiché la funzione di distribuzione radiale permette di darci una visione generale delle proprietà del campione. Verrà utilizzata successivamente per verificare di aver raggiunto la fase vetrosa del campione confrontando i picchi finali con quelli cristallini.

3.5 Variazione della temperatura nel sistema

Il riscaldamento e il raffreddamento del cristallo vengono effettuati da una routine che permette di campionare randomicamente le velocità atomiche data la temperatura desiderata. La routine segue la distribuzione di Maxwell per ricalibrare le velocità. Si decide inizialmente una frequenza, che indica ogni quanti timestep effettuare questa riscalatura, e l'energia termica desiderata in eV.

```
if pas%(freq*100) == 0 and mode == 2:
    pstd=sqrt(self.m*kt)
    self.px[0:N] = pstd[0:N]*random.normal(0., 1., N)
    self.py[0:N] = pstd[0:N]*random.normal(0., 1., N)
    self.pz[0:N] = pstd[0:N]*random.normal(0., 1., N)
    mtot  = sum(self.m)
    vcmx  = sum(self.px)/mtot
    vcmy  = sum(self.py)/mtot
    vcmz  = sum(self.pz)/mtot
    self.px[0:N] -= self.m[0:N]*vcmx
    self.py[0:N] -= self.m[0:N]*vcmy
    self.pz[0:N] -= self.m[0:N]*vcmz
```

Bisogna effettuare questo campionamento delle velocità più volte per riscaldare o raffreddare portando asintoticamente il cristallo alla temperatura desiderata.

La velocità di variazione termica è data sia dalla differenza tra la temperatura che noi vogliamo e quella attuale, sia dalla frequenza che si sceglie. Più `freq` sarà minore, maggiore sarà la rapidità di variazione di temperatura e viceversa, infatti nel quenching, per passare dalla fase liquida a quella vetrosa, si sceglie un valore piccolo di `freq` molto minore rispetto alla fase di riscaldamento.

Capitolo 4

Confronto dei risultati con LAMMPS

4.1 Cos'è LAMMPS

LAMMPS [6] è un codice open source scritto in linguaggio C/C++ che permette a chi lo installa di effettuare simulazioni di dinamica molecolare per un vasto numero di modelli di sistemi fisici. L'unica cosa che viene lasciata all'utente, dopo aver installato LAMMPS, è di scrivere un file di input diviso in due parti: descrizione delle caratteristiche del sistema e comandi di simulazione vera e propria. Nel nostro caso ad esempio si specificano nella prima parte le caratteristiche del cristallo utilizzato (posizioni, masse, legami, etc), il tipo di potenziale scelto [7], e altri parametri fisici del sistema. Nella seconda parte, invece si inseriscono diverse righe di comandi in cui si passano in sequenza al programma il numero di timestep, la temperatura desiderata, le grandezze fisiche da misurare (nel nostro caso, oltre alle usuali temperatura, pressione e energia puntuale, si richiede la funzione di distribuzione radiale). Un esempio di file di input utilizzato è il seguente:

```
units           metal
boundary        p p p
atom_style      atomic
read_data       data.quartz
replicate       8 8 6
velocity        all create 4000.0 277387 mom yes
displace_atoms  all move 0.05 0.9 0.4 units box
pair_style      tersoff
pair_coeff      * *  SiO.tersoff Si O
neighbor        0.5 bin
neigh_modify    delay 10
comm_modify  cutoff 7.5
fix             1 all nve
thermo          100
timestep        0.0005

velocity        all create 500.0 277387 mom yes
run             500
velocity        all create 500.0 277387 mom yes
run             500

reset_timestep  0
compute initial_rdf all rdf 256  1 1  2 2  1 2  2 1 cutoff 7.0  # Si-Si, O-O, Si-O, O-Si
fix rdf all ave/time 100 100 10000 c_initial_rdf[*] file sio2_tersoff_intialRDF.dat mode vector
run             10000

velocity        all create 7000.0 277387 mom yes
run             1000
...
```

Senza entrare nel dettaglio di come scrivere un input valido per LAMMPS, basti sapere che l'intento del nostro codice di input è di:

- Costruire un cristallo in configurazione di α-quarzo
- Calcolare la funzione di distribuzione radiale della fase cristallina
- Scaldarlo per farlo liquefare
- Calcolare la funzione di distribuzione radiale della fase liquida
- Raffreddare rapidamente il cristallo per ottenere una fase vetrosa
- Calcolare la funzione di distribuzione radiale della fase vetrosa

Al termine dell'esecuzione avremo quindi tre file diversi in cui si trovano i valori delle funzioni di distribuzione atomiche parziali in funzione della distanza, nelle tre fasi principali oggetto della simulazione.

4.2 Creazione di vetri in silice

Costruito il programma per calcolare forza ed energia di ogni singolo atomo di un campione, prendiamo una configurazione di α-quarzo SiO_2. Per creare un vetro bisogna prima fondere il cristallo, portandolo ad una temperatura dell'ordine di 3000°K, in modo da ottenere un liquido. Dopodiché si procede con il cosiddetto quenching, cioè si raffredda il liquido molto rapidamente, in modo da non consentire agli atomi di riposizionarsi nella configurazione cristallina. Noi in particolare portiamo il cristallo fino a temperature di 7000°K, per accelerare la liquefazione, per poi raffreddarlo rapidamente fino a tornare alla temperatura ambiente ($\sim$ 300°K). Le grandezze che andremo a confrontare tra il nostro programma e LAMMPS saranno $g_{AB}(r)$ e numero di coordinazione $n_{AB}(r)$ con $A = Si,\ O$ e $B = Si,\ O$, graficando e confrontando i risultati nel prossimo capitolo.

Ci si aspetta che le $g_{AB}(r)$, le quali hanno un picco principale ad una distanza corrispondente alla lunghezza dei legami tra i due atomi A e B, man mano che le posizioni atomiche perdono la loro periodicità, il picco dovrebbe allargarsi abbassandosi, simbolo che le distanze di legame sono distribuite su più lunghezze. Per calcolare il numero di coordinazione dell'atomo B nell'intorno dell'atomo A, si calcola semplicemente l'integrale della $g_{AB}(r)$ moltiplicato per all'opportuno elemento di volume e per la densità atomica corrispondente ρ_B.

4.3 Confronto dei risultati

In questo capitolo si riportano i grafici della funzione di distribuzione radiale per cristallo (300°K), liquido (3000°K) e amorfo (300°K), per ogni coppia di elementi (Si-Si, Si-O, O-O). Si precisa che come timestep è stato scelto $0.5fs$ in entrambi i casi. Cambia invece il numero di timestep, che per LAMMPS sono stati in totale circa $130k$, mentre per il codice in Python sono circa la metà ($70k$). I grafici riportano sia la $g(r)$ che il numero di coordinazione, che si ricava dalla seguente espressione

$$n_{AB}(R) = 4\pi\rho_B \int_0^R r^2 g_{AB}(r) dr \tag{4.1}$$

dove si è indicato con ρ_B la densità degli atomi B e $g_{AB}(r)$ è la funzione di distribuzione radiale per la coppia di atomi A-B. Nello specifico si grafica in rosso la fase cristallina iniziale, in blu la fase liquida e in verde quella vetrosa. Si traccia anche, nelle coppie Si-Si, Si-O e O-Si, una retta orizzontale che indica il numero di coordinazione teorico, rispettivamente 4, 4 e 2. Si mostrano sulla sinistra i dati ottenuti con LAMMPS, e sulla destra i risultati ottenuti con il codice in Python.

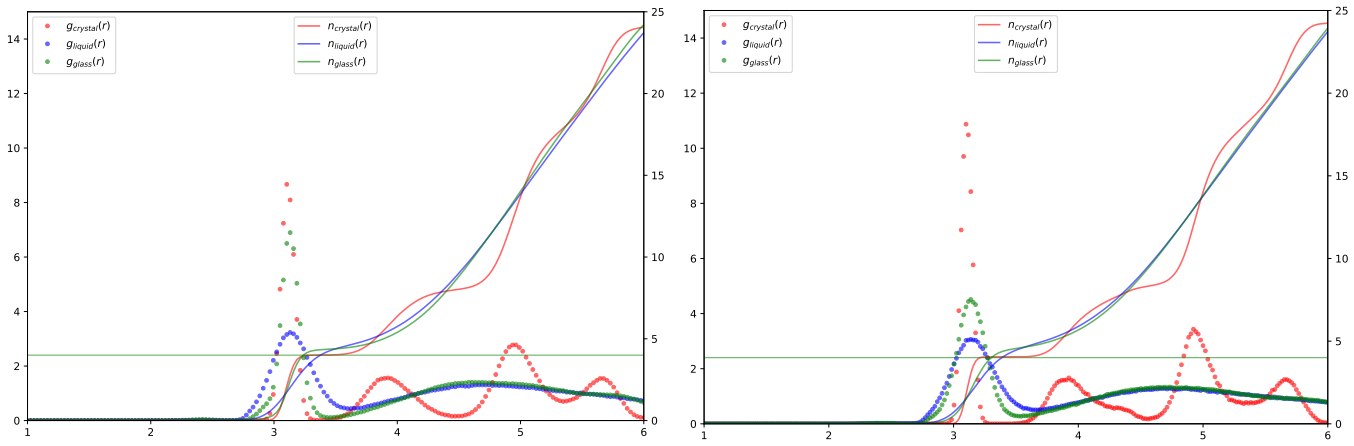

Figura 4.1: g(r) e numero di coordinazione (Si-Si)

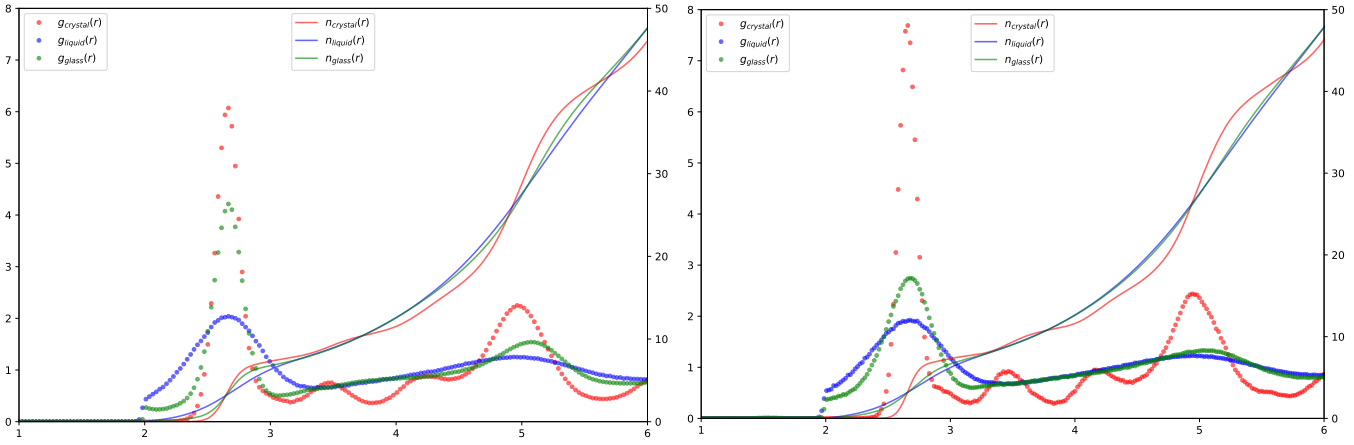

Figura 4.2: g(r) e numero di coordinazione (O-O)

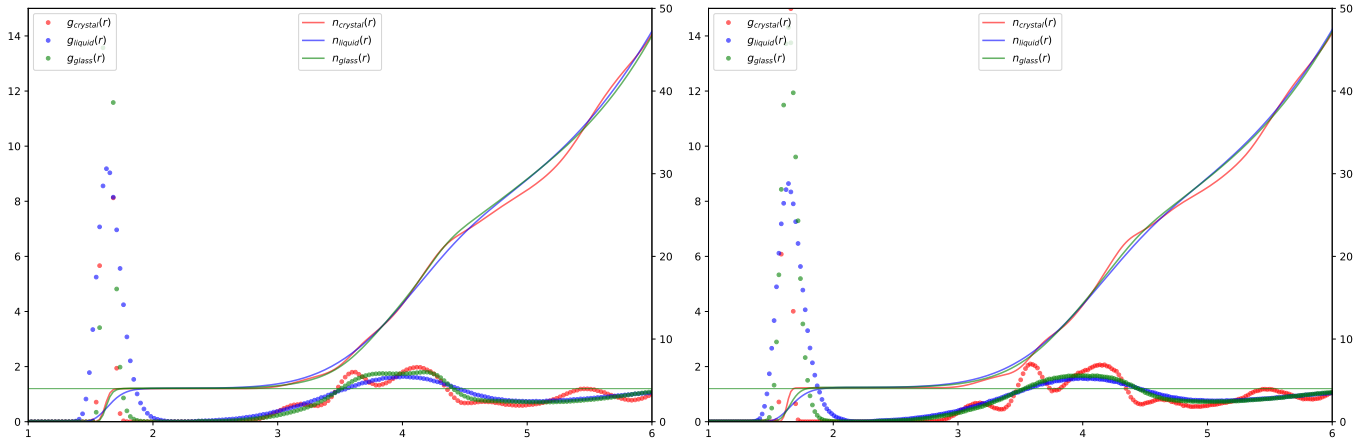

Figura 4.3: g(r) e numero di coordinazione (Si-O)

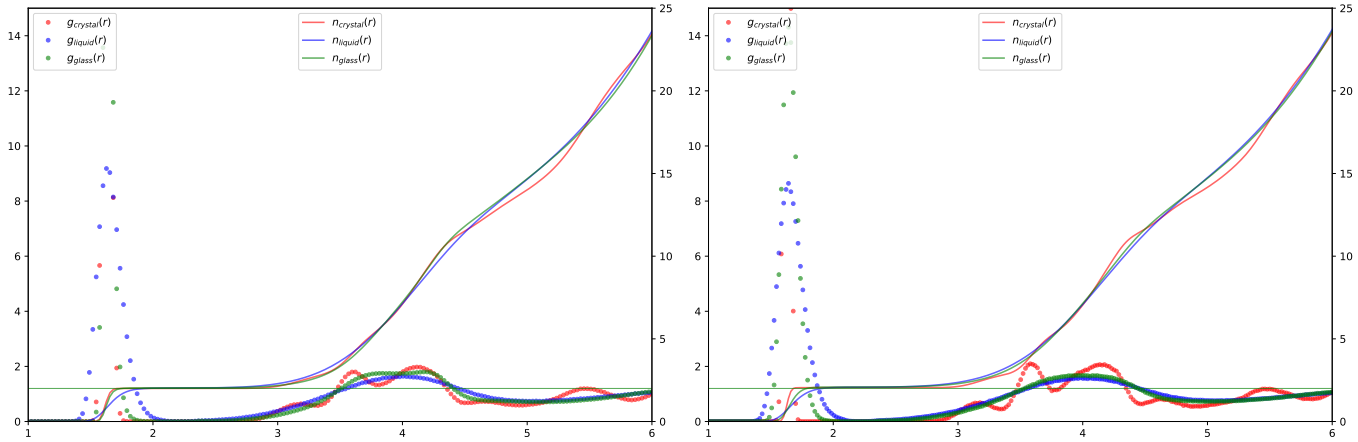

Figura 4.4: g(r) e numero di coordinazione (O-Si)

Si nota che i risultati ottenuti, calcolati su traiettorie con numeri di timestep differenti, sono tra loro consistenti, e riportano tutti la diminuzione e l'allargamento del picco principale in modo evidente. Gli altri picchi confluiscono invece in un andamento molto più smooth della g(r). Ciò indica che le distanze dei primi vicini e successivi per le coppie di atomi in questione, sono distribuite su un range maggiore di valori. Infatti guardando l'andamento dell'integrale si nota che gli scalini sono molto meno pronunciati nella fase liquida e vetrosa del campione, mentre nel cristallo sono molto più netti.

Guardando le funzioni di distribuzione radiale inoltre sembra che il vetro ottenuto con LAMMPS e quello con il codice Python abbiano caratteristiche molto simili tra loro, indipendentemente dal percorso generato da sequenze di campionamento random differenti per le velocità.

4.4 Confronto delle prestazioni

Prima di confrontare i tempi di esecuzione è necessario fare una premessa. LAMMPS permette di parallelizzare l'esecuzione della simulazione, infatti nelle simulazioni si sono utilizzati fino a 8 core in parallelo. Viene prima fatto un confronto tra simulazioni con diversi numeri di core usati con LAMMPS. Si specifica che i tempi ricavati sono quelli riportati da LAMMPS come output della simulazione. Si è visto, con prove indipendenti, che si possono avere fluttuazioni attorno al

valore riportato del 5%, accettabile per il confronto che si vuole fare. Si riporta la tabella contenente il confronto tra tempi di esecuzione di simulazioni con diversi numeri di core con LAMMPS (i tempi sono riportati in secondi) di una super-cella formata da 3456 atomi.

# step	1 core	2 core	ratio 2/1 (%)	4 core	ratio 4/1 (%)	8 core	ratio 8/1 (%)
1000	8,05	4,28	53,22	2,36	29,35	1,56	19,39
2000	16,02	8,55	53,34	4,47	27,91	2,57	16,04
5000	40,22	20,55	51,09	10,95	27,21	6,66	16,56
10000	80,68	40,97	50,78	21,58	26,74	11,44	14,18
20000	161,78	82,25	50,84	43,46	26,87	22,51	13,91

Tabella 4.1: Prestazioni LAMMPS 1/2/4/8 core (3456 atomi)

Si nota che LAMMPS parallelizza egregiamente gli algoritmi della simulazione, sfruttando al meglio i core a disposizione. Tuttavia per poter fare un paragone consistente con le prestazioni del codice Python, bisogna eseguire una simulazione, andando a confrontare i tempi di esecuzione di LAMMPS con un solo core a disposizione, quindi senza parallelizzare, con quelli del codice Python. Bisogna inoltre che il numero di atomi all'interno della super-cella sia lo stesso, nel nostro caso 864. Si riporta di seguito la tabella contenente i risultati ottenuti al variare del numero di timestep:

# timestep	LAMMPS	Python	Ratio (%)
1000	2,310	5,373 s	232,59
2000	4,360	10,366 s	237,75
5000	10,359	25,356 s	244,77
10000	20,435	50,521 s	247,23
20000	40,612	101,379 s	249,63

Tabella 4.2: Prestazioni LAMMPS/Python (864 atomi)

Eseguendo le simulazioni su un unico core si nota che il programma Python ha tempi di esecuzione circa doppi, in linea con le prestazioni della compilazione *just-in-time* del codice Python per mezzo del compilatore *llvmlite* di Numba. I tempi superiori del codice Python potrebbero essere dati dal fatto di non aver richiesto ottimizzazioni particolari, al contrario di LAMMPS, oppure per una inefficienza dei calcoli di energia potenziale e forze, essendo, a livello di peso computazionale, la più dispendiosa.

Conclusioni

In questo studio si è voluto verificare che il potenziale di Tersoff fosse un buon modello fisico per il processo di creazione di vetri in silice. Analizzando le equazioni per l'energia potenziale ci si è posti come primo obiettivo di ricavare le forze agenti sugli atomi della super-cella contenente il campione. Questo processo di analisi è stato fatto più volte per verificare la correttezza dei calcoli e considerare diverse strade per l'implementazione nel codice.

Per l'esecuzione del codice si è utilizzata la libreria Numba, la quale permette una compilazione del codice Python. Numba richiede però che il codice sia scritto in un certo modo, affinché sia compatibile con la libreria, come ad esempio l'utilizzo di array della libreria Numpy al posto delle liste standard di Python. Queste accortezze sono tuttavia poche, rendendo poco dispendioso il processo di apprendimento dello stile con cui va scritto il codice. Discorso differente è invece il processo di debugging, in quanto, in caso di errore, il messaggio che restituisce il compilatore non è sempre chiaro. Questo problema tuttavia è facilmente evitabile: basta commentare la chiamata del comando per la compilazione e, facendo un run del programma, si otterrà un messaggio più chiaro di quale possa essere l'errore all'interno codice.

In conclusione, il programma scritto dà un risultato consistente con quello di LAMMPS, in quanto le funzioni di distribuzione radiale sono molto simili nei due codici. Si ha tuttavia una prestazione che è peggiore rispetto a quella di LAMMPS, dovuta in parte alla possibilità con il compilatore C/C++ di attivare livelli di ottimizzazione più complessi rispetto alle opzioni standard in cui opera il compilatore llvmlite, in parte all'uso in LAMMPS di una lista di Verlet come ottimizzazione ulteriore nella ricerca delle coppie interagenti.

Si progetta quindi di provare a migliorare l'efficienza del calcolo di energia potenziale e forze, e, successivamente, utilizzando le features di Numba per l'approccio open-mp, parallelizzare il codice Python e confrontare i nuovi risultati delle prestazioni con LAMMPS, inizialmente agendo sulle routine per il calcolo delle forze, e, successivamente sulle altre, come la costruzione della super-cella iniziale, la routine per la randomizzazione delle velocità e il calcolo delle funzioni di distribuzione radiali. Ci si aspetta che l'impatto maggiore sulle prestazioni sia dato dalla parallelizzazione della routine per il calcolo delle forze e quella per determinare le g(r), poiché sono le più pesanti a livello di carico computazionale.

Bibliografia

[1] Furio Ercolessi, *A molecular dynamics primer* (1997)

[2] Donald W Brenner, Olga A Shenderova, Judith A Harrison, Steven J Stuart, Boris Ni and Susan B Sinnott, *A second-generation reactive empirical bond order (REBO) potential energy expression for hydrocarbons*, J. Phys.: Condens. Matter **14** (2002) 783–802

[3] Zheyong Fan1, Luiz Felipe C. Pereira, Hui-Qiong Wang, Jin-Cheng Zheng, Davide Donadio and Ari Harju, *Force and heat current formulas for many-body potentials in molecular dynamics simulation with applications to thermal conductivity calculations* (2015)

[4] Shinji Munetoh, Teruaki Motooka, Koji Moriguchi and Akira Shintani, *Interatomic potential for Si–O systems using Tersoff parameterization*, Computational Materials Science **39** (2007) 334–339.

[5] M. J. Mehl, D. Hicks, C. Toher, O. Levy, R. M. Hanson, G. L. W. Hart and S. Curtarolo, *ENCYCLOPEDIA OF CRYSTALLOGRAPHIC PROTOTYPES* (2017) `http://aflow.org/prototype-encyclopedia/A2B_hP9_152_c_a.html`.

[6] S. Plimpton, *Fast Parallel Algorithms for Short-Range Molecular Dynamics*, J. Comp. Phys. **117** (1995) 1-19.

[7] S. J. Plimpton and A. P. Thompson, *Computational Aspects of Many-body Potentials*, MRS Bulletin **37** (2012) 513-521.