# Simulazione Isobarica di un gas di ellissoidi duri

Marco De Tommasi

## 1 Introduzione

In questa simulazione è studiato il comportamento di un gas di ellissoidi duri attraverso un algoritmo Metropolis-Montecarlo in condizioni isobariche (NPT). Dall'analisi dei risultati ottenuti nella simulazione, si è ricavata l'equazione di stato che è stata in seguito confrontata con quella ottenuta dallo sviluppo del viriale al primo ordine.

$$P\beta = \frac{\phi}{v_{he}}\Big(1 + \frac{\phi}{2v_{he}}v_{excl}\Big) \tag{1}$$

Il volume escluso $v_{excl}$ è stato stimato a sua volta attraverso un' integrazione Montecarlo. Gli ellissoidi utilizzati nella simulazione sono uniassiali con *aspect ratio* $X_0 = 2$. I semiassi sono $b = c = 1$ e $a = X_0$. Il volume degli ellissoidi è quindi ottenuto come $v_{he} = \frac{4\pi}{3}X_0$. Oltre alla stima dell'equazione di stato, è stata ricavata la funzione di distribuzione radiale $g(r)$.

## 2 Algoritmo

L'algoritmo utilizzato è un Montecarlo-Metropolis. La simulazione è effettuata mantenendo fissati il valore del numero di particelle $N$, della pressione $P$ e della temperatura $T$. Facendo evolvere il sistema, questo tenderà ad una condizione di equilibrio in cui è possibile calcolare il volume e quindi la densità $\rho$ del sistema. Uno dei vantaggi della simulazione NPT consiste appunto nella possibilità di ricavare l'equazione di stato del sistema senza dover calcolare il valore della pressione che, al contrario di quello della densità, risulta più costoso in termini computazionali. Dalla teoria sull'ensamble isobarico, sappiamo che la matrice di accettazione del processo stocastico sarà definita come:

$$acc(n \to m) = min\Big\{e^{-\beta\Delta G}, 1\Big\} \tag{2}$$

In cui $\Delta G$ coincide con la variazione di energia libera di Gibbs $G(N, T, P)$, che può essere calcolata attraverso la relazione:

$$\Delta G = P\beta\Delta V - N\ln\Big(\frac{V\prime}{V}\Big) + \beta\Delta U \tag{3}$$

Con $V$ volume iniziale, $V\prime$ quello finale, $\beta = \frac{1}{k_b T}$ il fattore di Boltzmann e $\Delta U$ dato dalla variazione di energia interna del sistema.
L'algoritmo compie 3 mosse distinte. Gli ellissoidi possono infatti essere spostati tramite una traslazione definita con un vettore generato randomicamente oppure ruotare randomicamente la loro orientazione tridimensionale all'interno del sistema. La scelta della mossa rotazionale o traslazionale è stabilita in maniera randomica ed equiprobabile, in modo da soddisfare il bilancio dettagliato. Per quanto riguarda la mossa di volume invece, essendo un movimento che causa la variazione di posizione della totalità degli elementi del sistema, questa avrà una probabilità di verificarsi pari ad $\frac{1}{N}$.

### 2.1 Condizioni iniziali

Il sistema è stato generato in una configurazione randomica iniziale. La simulazione è stata effettuata con un numero di particelle pari a $N = 250$. Ad ogni ellissoide è stata associata un' orientazione ed una posizione randomica all'interno del volume cubico scelto (nella simulazione è stato posto $L = 20$, con $L$ spigolo del cubo). La posizione associata agli ellissoidi è quella riferita al loro centro di massa. Al fine di generare correttamente il sistema, tendendo conto delle condizioni periodiche al bordo, all'interno del codice è presente una sezione che controlla di non avere overlap tra le particelle in prossimità dei bordi.

```
1  ...
2  else if(abs(distance.x())>L_[0]/2 || abs(distance.y())>L_[1]/2 || abs(distance.z())
3  >L_[2]/2){
4
5          d.setX(L_[0]*(int)(distance.x()/(L_[0]/2)));
```

```
6           d.setY(L_[1]*(int)(distance.y()/(L_[1]/2)));
7           d.setZ(L_[2]*(int)(distance.z()/(L_[2]/2)));
8
9           temp.SetPos(hardell_[j].posizione()+d);
10          temp.SetOrientation(hardell_[j].VectX(),hardell_[j].VectY(),hardell_[j].VectZ());
11
12          if(hardell_[i].EllOverlap(temp)==true){
13            hardell_[i].RandomPos(L_[0],L_[1],L_[2]);
14            hardell_[i].RandomOrient();
15  ...
```

La condizione riportata, calcola la distanza tra la particella $i$ appena generata e quella tra le altre particelle $j$ già presenti nel sistema, selezionando quelle che si trovano ad una distanza maggiore di $\frac{L}{2}$. Successivamente è calcolato con le condizioni periodiche al bordo il corrispondente della particella $j$ nei cubi adiacenti (riga 9) ed eseguito un controllo atto a verificare l'assenza di eventuali overlap (riga 12).

## 2.2 Generatore di numeri casuali

Al fine di poter utilizzare liberamente una funzione che permettesse la generazione di numeri casuali su un elevato numero di computazioni, si è utilizzato il generatore *Marsenne Twister* (*mt*19937). All'interno del codice è allegata una classe contenente una funzione che a partire da questo algoritmo genera un numero randomico compreso nell'insieme $[0, 1]$, ovvero la funzione $RandM01()$ utilizzata nella simulazione.

## 2.3 Mossa traslazionale

La mossa traslazionale è stata ottenuta variando randomicamente la posizione inziale di un ellissoide sorteggiato tra quelli presenti nel sistema secondo le equazioni:

$$\begin{aligned} x\prime_i &= x_i + \Delta(Rand - 0.5) \\ y\prime_i &= y_i + \Delta(Rand - 0.5) \\ z\prime_i &= z_i + \Delta(Rand - 0.5) \end{aligned}$$

Il parametro $\Delta$ è stato ricavato in modo da rendere ottimale[1] il metodo di integrazione secondo un processo di calibrazione (Sezione 2.6).

## 2.4 Mossa rotazionale

La variazione di orientazione è stata ottenuta secondo la relazione

$$\widehat{u}^{ROT} = \frac{\widehat{u} + \gamma\widehat{v}}{|\widehat{u} + \gamma\widehat{v}|} \tag{4}$$

in cui $\widehat{v}$ è un versore generato randomicamente su una sfera unitaria e $\gamma$ è un parametro che ha la funzione di regolare l'efficienza della mossa. Nella simulazione, l'asse di riferimento degli ellissoidi (parallelo in questo caso all'asse x della terna) è quello che modifica randomicamente la sua orientazione ($\widehat{x} \rightarrow \widehat{x\prime}$). Al fine di ricavare la nuova terna ortogonale in seguito alla rotazione, partendo dalla nuova direzione $\widehat{x\prime}$ e dall'angolo $\alpha = arccos(\frac{\widehat{x}\cdot\widehat{x\prime}}{|x||x\prime|})$ generato tra i due versori $\widehat{x}$ ed $\widehat{x\prime}$ è stata utilizzata la relazione di Rodrigues[2]:

$$R = I + sin(\alpha)[n] + (1 - cos(\alpha))[n]^2 \tag{5}$$

con $[n]$ si intende la matrice associata al versore $\widehat{n}$ normale generato dal prodotto vettoriale $\widehat{x} \times \widehat{x\prime}$ normalizzato $\widehat{n} = \frac{1}{|n|}(n_1, n_2, n_3)$

$$[n] = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} \tag{6}$$

[1] Per questo metodo come per i successivi, il valore è stato scelto in modo da rendere il più efficiente possibile la simulazione, ovvero facendo tendere il numero di mosse accettate dalla simulazione a circa il 50%.

[2] La formula di Rodrigues è un algoritmo per ruotare un vettore nello spazio, dato un asse e un angolo di rotazione. Per estensione, questo metodo può essere usato per trasformare i tre vettori di una base, e quindi per calcolare la matrice di rotazione corrispondente alla rappresentazione asse-angolo.

## 2.5 Mossa di volume

La simulazione è stata effettuata a partire da una configurazione cubica, se questa mossa è selezionata viene generato un nuovo volume:

```
newVol=volIniz+(gen.RandM01()-0.5)*deltaVmax;
```

Il parametro *deltaVmax* è calibrato in modo da ottimizzare la computazione. Successivamente, la mossa può essere accettata o rifiutata in base al criterio espresso in precedenza (Relazione 2). Essendo una simulazione di particelle dure, l'energia interna sarà tale da annullare il movimento nel caso in cui ho che la variazione di volume sia tale da causare sovrapposizioni nelle particelle. Nel caso in cui la mossa sia accettata, la posizione delle particelle è stata riscalata in proporzione alla variazione di volume sorteggiata, secondo la relazione

```
for(int i=0; i<hardell_.size();i++){
   hardell_[i].SetPos(hardell_[i].posizione()*(newL/L_[0]));
 }
```

Dove *newL* coincide con lo spigolo del nuovo cubo di volume *newVol* generato. In seguito al controllo di eventuali overlap oppure nel caso in cui la mossa sia semplicemente rifutata secondo il criterio di accettazione è possibile ritornare alla configurazione iniziale tramite un' operazione inversa.

## 2.6 Equilibrazione

La simulazione prima di effettuare la stima della densità effettua una procedura di equilibrazione. Si può dimostrare che il sistema ha equilibrato se la seguente relazione è soddisfatta:

$$MSD = \frac{1}{N}\sum_i |\vec{r_i}(t) - \vec{r_i}(0)|^2 > \sigma^2 \tag{7}$$

Dove il Mean Square Displacement è ottenuto come la media sulle particelle della differenza in modulo tra le posizioni al passo $t$ della simulazione $(\vec{r}(t))$ e la posizione iniziale $\vec{r}(0)$ . Se MSD risulta maggiore di $\sigma^2$, dove con sigma si intende il diametro delle sfere dure del sistema allora questo risulterà equilibrato. Nella simulazione, invece di utilizzare il parametro $\sigma$ è stata utilizzata una condizione più stringente, ovvero quella di assumere che le particelle abbiano un displacement pari almeno alla metà della taglia del sistema, ovvero $\frac{L}{2}$. Questa scelta è stata effettuata in modo da essere certi di aver correttamente raggiunto un regime in cui il sistema ha equilibrato.
Durante il processo di equilibrazione è effettuata la calibrazione dei parametri $\Delta$ , $\gamma$ e $\Delta V$ utilizzati nelle mosse del Montecarlo-Metropolis.

```
if(i%2000==0 && i!=0 && p==0 && Volume==true ){
  if(EquilibrationCriterion(L_[0]/2)==true){
    if(verbose==true){
      cout<<"il sistema    equilibrato, fine fase di calibrazione."<<endl;
      cout<<"msd : "<< msd()<<endl;
    }
   ...
    cout<<"Inizio calcolo della densit "<<endl;
  }else{
    if(verbose==true){
      cout<<"il sistema non    equilibrato, continuo la fase preliminare.. "<<endl;
      cout<<"msd:"<<msd()<<endl;
    }

    if(acceptM>trialM/5*3){
      delta=delta*1.1;
      acceptM=0;
      trialM=0;
    }else if(acceptM<trialM/5*2){
      delta=delta/1.1;
      acceptM=0;
      trialM=0;
    }else{
      cout<<"delta trovato:"<<delta<< " accepted: "<<acceptM<< " trials: "<<trialM<<endl;
      acceptM=0;
      trialM=0;
    }
    ....
```

Per ogni duemila passi della simulazione, il sistema calcola la varianza delle posizioni e la confronta con il valore di riferimento impostato tramite la funzione *EquilibrationCriterion*. Se il sistema non ha

equilibrato, si calcolano il numero di mosse accettate e rifiutate in questi steps per ogni tipo di mossa. Se il numero di mosse accettate dal sistema è compreso tra il 40% ed il 60% il sistema utilizza lo stesso parametro nei duemila passi successivi (riga 23), altrimenti lo modifica del 10%. Nella porzione di codice riportata è presente a titolo di esempio il processo di calibrazione delle mosse traslazionali. Nel caso in cui il sistema abbia un numero di mosse accettate maggiori del 60% (riga 16) il parametro $\Delta$ dovrà essere incrementato, in modo da permettere alle particelle di spaziare su una regione di volume maggiore, aumentando la possibilità di interazione con le altre. Viceversa nel caso in cui il numero di mosse accettate sia minore del 40% (riga 16). Analogo ragionamento è stato utilizzato per le mosse rotazionali. Per le mosse di volume si ha che nell'ipotesi in cui il numero di mosse rifiutate sia troppo elevato, la variazione di volume dovrà diminuire, in modo da mantenere il sistema in una configurazione mediamente simile a quella iniziale, riducendo la probabilità di sovrapposizioni tra le particelle.

# 3 Equazione di Stato

L'equazione di stato è stata costruita a partire dalla media dei valori di densità, ottenuti durante i passi della simulazione. I valori di pressione utilizzati sono compresi nell'insieme $[0.005, 0.030]$ e si è utilizzata una fattore $\beta = 1$.

## 3.1 Calcolo della densità

Gli andamenti delle densità ottenute sono riportati nel seguente grafico:

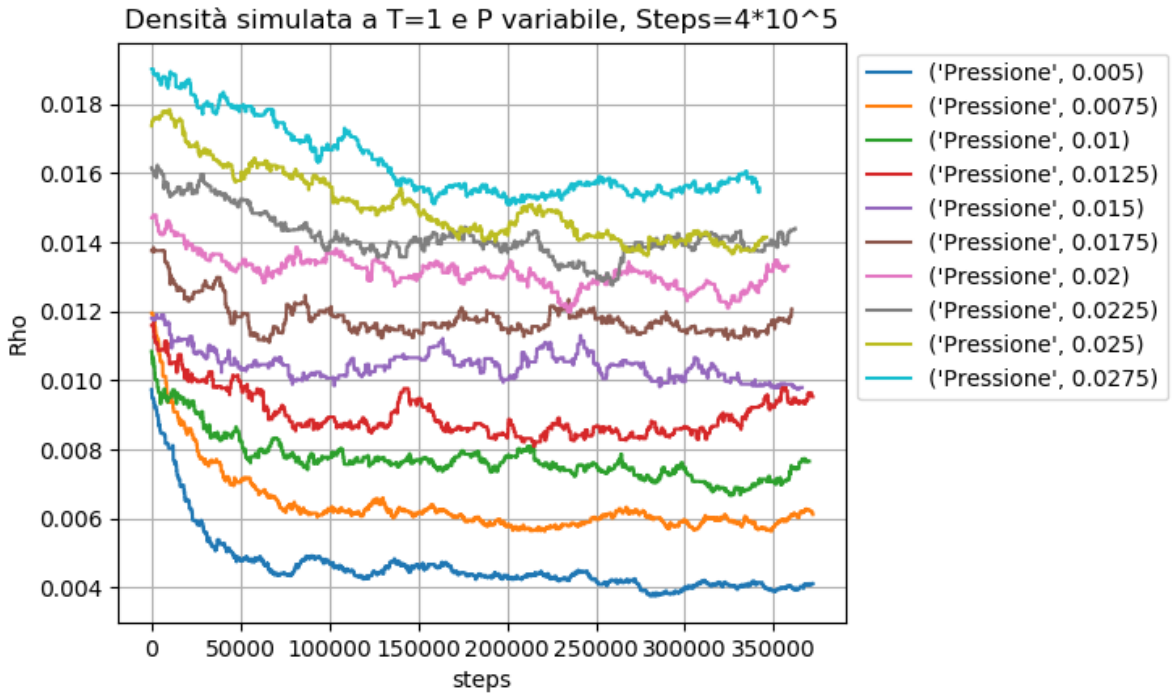


Figure 1: Andamento Delle densità ottenute in seguito al processo di equilibrazione per un intervallo di pressioni compreso tra 0.05 e 0.030.

Dove si osserva come il valore dei passi risulta differente per ogni simulazione, ciò è dovuto al processo di equilibrazione che può richiedere un numero di passi variabile. Con uno script in Python si sono poi ricavati gli andamenti medi della densità per ogni valore di pressione.

## 3.2 Confronto con lo sviluppo del viriale

Attraverso un metodo di Montecarlo è stato ricavato il volume escluso degli ellissoidi con $X_0 = 2$ che risulta essere

$$v_{excl} = 76.8 \tag{8}$$

La stima è stata ottenuta tramite l'utilizzo della classe *vexcl* con un numero di tentavi pari a $10^6$. Questo risultato è stato utilizzato nella relazione data dallo sviluppo del viriale che è stata poi usata

come confronto per la simulazione (Relazione 1). I valori medi della densità sono stati moltiplicati per il volume degli ellissoidi, in modo da ottenere un grafico rispetto alla *packing fraction* $\phi = \rho * v_e$. Il risultato ottenuto è riportato nel seguente grafico:

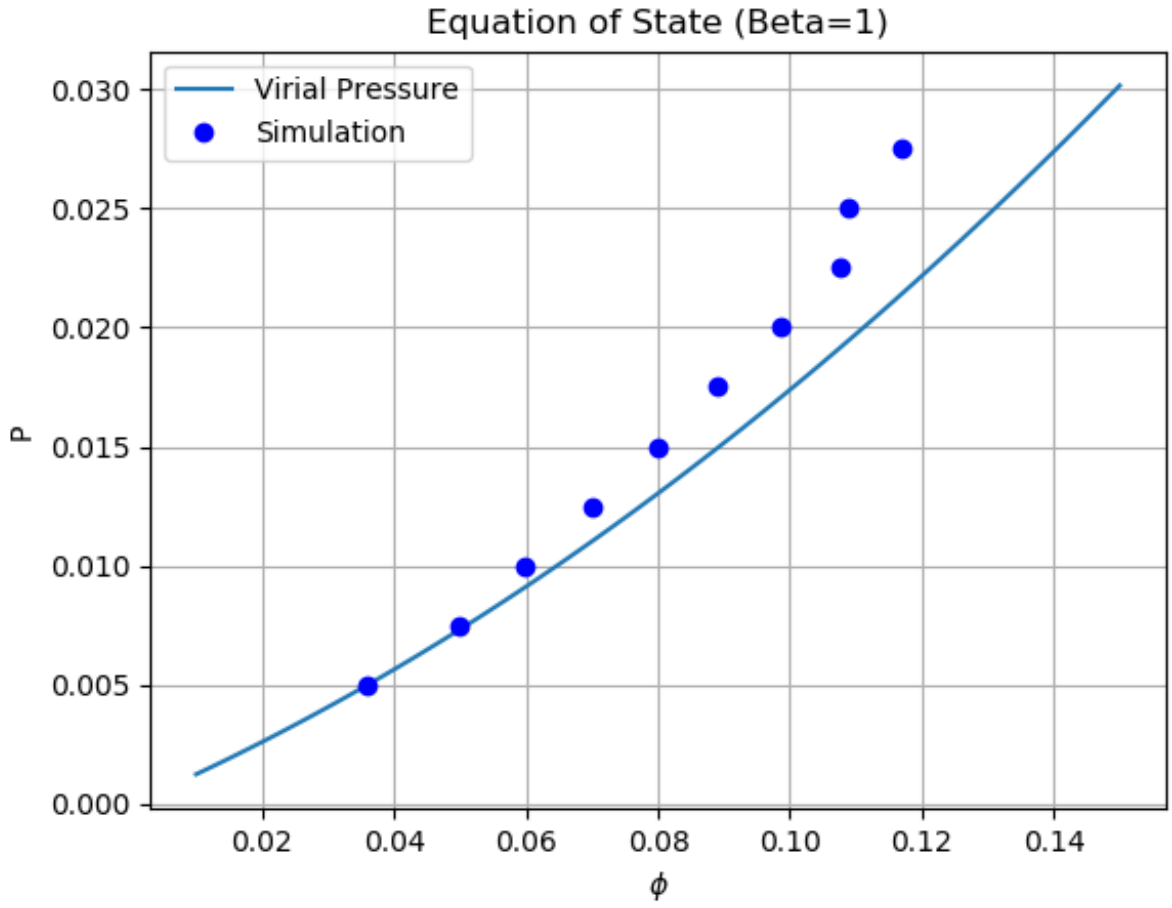


Figure 2: Equazione di stato per un gas di ellissoidi duri ottenuta da una simulazione Metropolis-Montecarlo isobarica confrontata con l'equazione del viriale. Numero di passi utilizzati $4 \times 10^5$.

L'andamento dell'equazione di stato ottenuta risulta avere una pendenza maggiore nella regione a densità più elevate, in linea con le aspettative teoriche.

### 3.3 Calcolo della g(r)

In ultimo, è stato ricavato l'andamento della funzione di distribuzione radiale $g(r)$. Computazionalmente, questa funzione è ottenuta generando un istogramma all'interno del quale è contato il numero di volte in cui una specifica distanza tra due particelle è ottenuta (riga 14).

```
1  ...
2     for(int i=0;i<hardell_.size();i++){
3       for(int j=i+1;j<hardell_.size();j++){
4
5          distance=hardell_[i].posizione()-hardell_[j].posizione();
6
7          d.setX(L_[0]*(int)(distance.x()/(L_[0]/2)));
8          d.setY(L_[0]*(int)(distance.y()/(L_[0]/2)));
9          d.setZ(L_[0]*(int)(distance.z()/(L_[0]/2)));
10
11         m=round((distance-d).mod()/width);
12
13         deltaV=4*M_PI*pow((distance-d).mod(),2)*width;
14         hist[m]=hist[m]+1/(rho*deltaV*hardell_.size());
15      }
16    }
17    ...
```

L'istogramma generato tramite i singoli conteggi è rinormalizzato per un fattore $deltaV$ e per un fattore $\rho * N$. Il risultato è quello di ottenere una stima del numero di particelle contenute nella regione infinitesima $deltaV$ rispetto al numero di particelle contenute in media nella stessa (quantità definita dalla densità del sistema $\rho$).
La presenza delle condizioni periodiche al bordo (righe 7-9) ci permette di osservare il comportamento della $g(r)$ nel limite di $r > \sigma$. Come è noto, il comportamento della funzione di distribuzione dovrebbe essere tale da raggiungere in questo limite il valore unitario $g(r >> \sigma) = 1$. Nella simulazione, avendo utilizzato degli ellissoidi duri con diametro minimo pari a $\sigma = 2$, ci aspettiamo di poter verificare anche l'assenza di conteggi nel limite $r < \sigma$. In seguito è riportato l'andamento della funzione $g(r)$ nel caso di un sistema con un numero di particelle pari a $N = 300$ e mediato su un numero di stati pari a $10^5$.

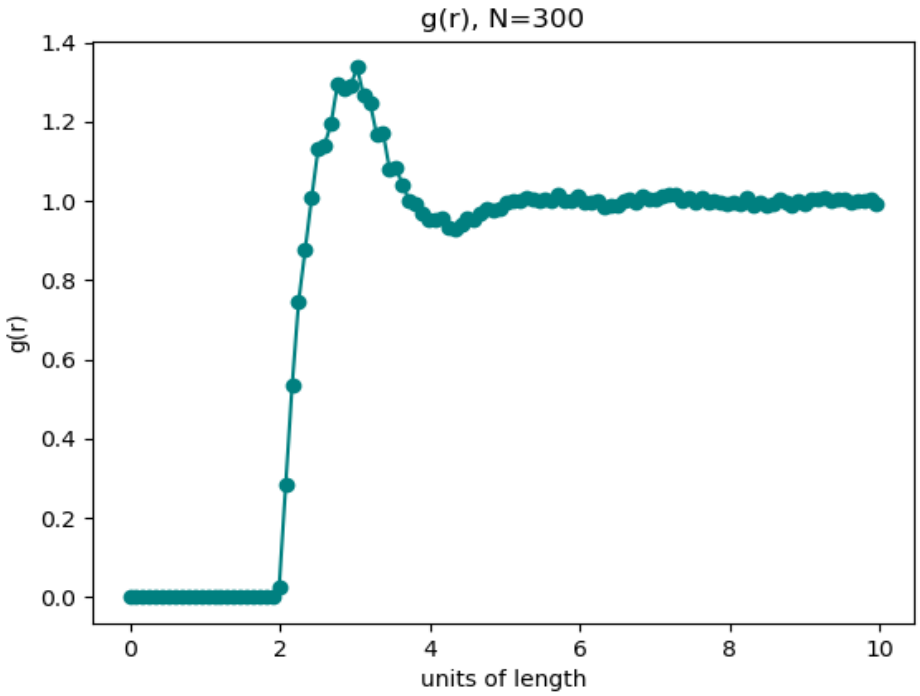


Figure 3: Funzione di distribuzione radiale $g(r)$ per un sistema fromato da $N = 300$ particelle di ellissoidi duri.

Come si può osservare in figura, il comportamento a $r < \sigma$ è rispettato, $g(r) = 0$ fino al punto in cui $r = 2$, coincidente appunto con il valore $\sigma = 2$. Nel limite in cui $r >> \sigma$ si ottiene appunto l'andamento teorico atteso, in cui si ha $g(r) = 1$.