Università degli Studi di Napoli “Federico II”

Scuola Politecnica e delle Scienze di Base
Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica “Ettore Pancini”

Laurea triennale in Fisica

Sviluppo e applicazioni del metodo Monte Carlo quantistico

Relatori:
Prof. Dario Alfè
Dott. Andrea Zen

Candidato:
Roberto Rivezzi
Matricola N85001239

A.A. 2020/2021

Introduzione

Fin dalla sua nascita, negli anni '30, una delle principali applicazioni della Meccanica Quantistica è stata lo studio delle proprietà microscopiche della materia. Nonostante le equazioni che descrivono correttamente questi sistemi siano note da decenni, esse sono, come disse Dirac "troppo complicate per essere risolvibili". Risulta perciò fondamentale cercare soluzioni approssimate dell'equazione di Schrodinger a molti elettroni di accuratezza soddisfacente.

Di fatto, questo problema è una delle più grandi sfide in Fisica della Materia.

Uno degli approcci storicamente più rilevanti è quello proposto da Hartree e Fock, un metodo variazionale in cui si assume che la funzione di prova sia un determinante di Slater, ovvero la più semplice espressione funzionale che può assumere la funzione d'onda di un sistema composto da più fermioni. Questa assunzione permette di ottenere un' equazione "autoconsistente", in cui il potenziale a cui sono soggetti gli elettroni dipende dalle loro autofunzioni, che può essere risolta in modo iterativo.

La teoria così sviluppata, è una "teoria di campo medio", in cui ogni elettrone è indipendente, ma risente di un campo esterno generato da tutti gli altri.

La scelta di usare come forma funzionale un determinante di Slater sarebbe esatta solo nel caso in cui fosse possibile fattorizzare l'Hamiltoniana in termini di singolo elettrone. La presenza del termine di repulsione coulombiana rende tale fattorizzazione non possibile, quindi la funzione d'onda dovrebbe accoppiare esplicitamente i gradi di libertà elettronici. Da queste considerazioni si può capire come il metodo Hartree-Fock sia inadeguato a descrivere sistemi dove emergono effetti di correlazione dovuti alle interazioni fra elettroni[1].

Sono stati perciò sviluppati nel tempo numerosi approcci " post-Hartree-Fock" che possiamo suddividere in due categorie: i metodi del tipo "interazione di configurazione" che usano come funzione di prova una combinazione lineare di determinanti di Slater, e metodi che, partendo da funzioni d'onda ricavate con il metodo HF, trattano la correlazione con la teoria delle perturbazioni.

L'utilizzo di approcci di quantum chemistry come quelli citati, diventa computazionalmente proibitivo al crescere del numero di elettroni. Si consideri ad esempio che per il metodo CI, il numero di determinanti di Slater possibili, cresce esponenzialmente con il numero di elettroni.

Per questi motivi, negli ultimi anni il metodo più popolare per stimare le proprietà delle strutture elettroniche è quello che fa uso della "Teoria del Funzionale Densità" (DFT),

[1] Escludendo la correlazione dovuta alla repulsione di Pauli, presente in HF solo tra particelle di stesso spin e derivante dalla antisimmetrizzazione persente anche in un singolo determinante di Slater.

in quanto è spesso abbastanza accurato e computazionalmente efficiente. Tuttavia, il DFT non presenta la stessa accuratezza con tutti i sistemi, pertanto occorre avere dei metodi alternativi più accurati.

Per lo studio di sistemi aventi oltre un centinaio di elettroni, l'unica alternativa al DFT è data dai metodi Quantum Monte Carlo (QMC), una famiglia di metodi per la soluzione dell'eq. di Schrodinger basati su un approccio stocastico. I metodi QMC presentano infatti uno scaling favorevole del costo computazionale al crescere del sistema e utilizzano algoritmi che si prestano ad essere implementati efficientemente nelle moderne strutture di supercalcolo.

Lo scopo di questa tesi sarà quello di illustrare la teoria alla base dei più popolari tra gli approcci QMC, il Variatonal e il Diffusion Monte Carlo, si è inoltre sviluppato un codice che implementa l'algoritmo alla base del DMC nel semplice caso di sistemi monoelettronici.

Nel primo capitolo, si illustrerà la teoria alla base dei metodi Monte Carlo, ed in particolare quella alla base dei metodi Variational Monte Carlo (VMC) e Diffusion Monte Carlo (DMC).

Nel secondo capitolo si descriverà un esempio di implementazione del metodo, svolto durante la tesi sviluppando un algoritmo in linguaggio C.

Nel terzo capitolo si analizzeranno i risultati della simulazione, e si verificherà la coerenza di questi con le previsioni analitiche.

Indice

1 I metodi Quantum Monte Carlo **5**
- 1.1 I metodi Monte Carlo . . . 5
  - 1.1.1 Integrazione numerica col metodo Monte Carlo . . . 5
  - 1.1.2 L'algoritmo di Metropolis . . . 6
- 1.2 Il metodo Variational Monte Carlo . . . 9
  - 1.2.1 Il metodo . . . 9
  - 1.2.2 Le funzioni d'onda di Slater-Jastrow . . . 10
- 1.3 Il metodo Diffusion Monte Carlo . . . 10
  - 1.3.1 Equazione di Schrodinger in tempo immaginario . . . 11
  - 1.3.2 Proiezione sullo stato fondamentale . . . 12
  - 1.3.3 Funzione di Green . . . 12
  - 1.3.4 Formalismo integrale e discretizzazione del tempo . . . 13
  - 1.3.5 Analogia con l'equazione di diffusione . . . 13
  - 1.3.6 Il ruolo del potenziale . . . 15
  - 1.3.7 Il processo di branching . . . 16
  - 1.3.8 Il parametro E_T . . . 16

2 L'algoritmo **17**
- 2.1 Schema dell'algoritmo . . . 17
  - 2.1.1 Input e inizializzazione . . . 18
  - 2.1.2 Diffusione . . . 19
  - 2.1.3 Branching . . . 19
  - 2.1.4 Il ricalcolo di E_t . . . 20
  - 2.1.5 Output . . . 21

3 Risultati **22**
- 3.1 Oscillatore armonico . . . 22
  - 3.1.1 Risultati analitici . . . 22
  - 3.1.2 Simulazione . . . 22
  - 3.1.3 Analisi dei dati . . . 25
  - 3.1.4 Risultati della simulazione . . . 26
  - 3.1.5 Il bias della popolazione . . . 27
- 3.2 Atomo di idrogeno . . . 28
  - 3.2.1 Risultati analitici . . . 28
  - 3.2.2 Risultati della simulazione . . . 29

3.3 Appendice: listato del programma 33

Capitolo 1

I metodi Quantum Monte Carlo

1.1 I metodi Monte Carlo

I metodi Monte Carlo sono una famiglia di metodi che può essere usata per simulare il comportamento di un sistema fisico o matematico. Essi si distinguono da altri metodi di simulazione per il fatto che sono stocastici, ovvero non deterministici.

Tale caratteristica, come suggerisce il nome, si ottiene generando una grande mole di numeri casuali (nella pratica si usano funzioni che generano numeri pseudo-casuali), come quelli che si generano in una serata in un casinò.

1.1.1 Integrazione numerica col metodo Monte Carlo

Buona parte delle simulazioni Monte Carlo, si possono ridurre alla stima numerica di un integrale multidimensionale, risulta perciò utile discutere il metodo alla base di questa stima.

Consideriamo un integrale del tipo

$$I = \int d\mathbf{R} g(\mathbf{R}) \tag{1.1.1}$$

dove $\mathbf{R}$ è un vettore 3N-dimensionale, solitamente chiamato "walker" o "configurazione".

Introduciamo una funzione P($\mathbf{R}$) che possiamo interpretare come una densità di probabilità, ovvero una funzione tale che:

$$P(\mathbf{R}) \geq 0\,;$$

$$\int d\mathbf{R} P(\mathbf{R}) = 1$$

Possiamo riscrivere l'integrale nella forma

$$\int d\mathbf{R} f(\mathbf{R}) P(\mathbf{R}) \tag{1.1.2}$$

dove $f(\mathbf{R}) \equiv g((R))/P((R)$.

Si noti che adesso l'integrale può essere interpretato come una media. Possiamo quindi estrarre casualmente, con la distribuzione di probabilità $P(\mathbf{R})$, un insieme di M vettori $\mathbf{R_m}$e stimare l'integrale con la media

$$I \approx \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{R_m}) \tag{1.1.3}$$

Essendo i vettori $\mathbf{R_m}$ scelti casualmente e in modo indipendente l'uno dall'altro, possiamo applicare il teorema del limite centrale per calcolare l'errore sulla stima dell'integrale:

$$\sigma_I = \frac{\sigma_f}{\sqrt{M-1}} = \sqrt{\frac{\overline{f^2} - \overline{f}^2}{M-1}} \tag{1.1.4}$$

dove σ_f è la deviazione standard della variabile stocastica f($\mathbf{R}$).

Si può notare come l'errore stocastico ha un andamento $\propto \frac{1}{\sqrt{M}}$, quindi può essere ridotto migliorando il sampling della distribuzione, cosa che avviene però aumentando il costo computazionale.

Un altro modo per agire sull'errore stocastico si può trovare osservando che la presenza di σ_f al numeratore implica che l'incertezza sulla stima può essere diminuito con un'opportuna scelta di P($\mathbf{R}$): da un punto di vista formale la scelta ottimale sarebbe

$$P(\boldsymbol{R}) = \frac{|g(\boldsymbol{R})|}{\int |g(\boldsymbol{R'})|\, d\boldsymbol{R'}} \tag{1.1.5}$$

in questo modo la funzione $f(\boldsymbol{R})$ sarebbe costante e avrebbe errore stocastico nullo.

Tale scelta risulta spesso impossibile, poichè l'integrale al denominatore è simile all'incognita del problema che si vuole risolvere, però la considerazione fatta fa intuire come si possa sviluppare la simulazione migliorando il campionamento nei punti in cui la funzione g assume valori più alti: questo metodo di campionamento viene detto *importance sampling*.

Una scelta ragionevole sarebbe quella di sostituire la funzione g con una funzione analiticamente integrabile che abbia un andamento simile. Dato che anche questa scelta non è sempre possibile, si usano spesso un approccio diverso: si simula un processo stocastico dinamico (e.g. un cammino aleatorio) pensato in modo tale che le regioni in cui l'integrando contribuisce maggiormente all'integrale, abbiano una probabilità maggiore di essere visitate, e che gli stati ottenuti durante questo processo, ad esempio le configurazioni ottenute con un random walk, abbiano una distribuzione con le caratteristiche richieste. Un esempio di questo approccio è costituito dall'algoritmo di Metropolis.

1.1.2 L'algoritmo di Metropolis

Quando si usano metodi Monte Carlo, è spesso necessario campionare distribuzioni di probabilità $P(\boldsymbol{R})$ complicate in spazi ad elevatà dimensionalità. In generale, non risulta sempre possibile usare il metodo della cumulativa inversa, perchè, ammesso che esista,

potrebbe essere troppo complicata per effettuare il campionamento. L'algoritmo di Metropolis invece, permette di campionare qualsiasi distribuzione di probabilità senza bisogno di conoscere la relativa funzione cumulativa.

L'algoritmo di Metropolis genera una sequenza di campioni $\mathbf{R_m}$ muovendo un walker secondo le seguenti regole:

1. Si inizializzi il walker in una posizione casuale $\mathbf{R}$.
2. Si provi uno spostamento verso una nuova posizione $\mathbf{R}'$, scelta secondo una certa funzione densità di probabilità $\mathrm{T}(\mathbf{R}' \leftarrow \mathbf{R})$.
3. Si accetti lo spostamento di prova nella posizione $\mathbf{R}'$ con probabilità

$$A(\mathbf{R}' \leftarrow \mathbf{R}) = Min\left(1, \frac{T(\mathbf{R} \leftarrow \mathbf{R}')P(\mathbf{R}')}{T(\mathbf{R}' \leftarrow \mathbf{R})P(\mathbf{R})}\right) \tag{1.1.6}$$

   Se lo spostamento viene accettato, il punto $\mathbf{R}'$ diviene il punto successivo del random walk. In caso contrario il punto successivo sarà $\mathbf{R}$.Possiamo notare che se $\mathrm{P}(\mathbf{R})$ è alta, buona parte degli spostamenti dalla posizione $\mathbf{R}$ verranno rifiutati e il punto $\mathbf{R}$ si presenterà più volte nella sequenza generata dall'algoritmo.
4. Si ritorni allo step 2.

I primi punti della sequenza dipendono dalla posizione iniziale del walker, quindi si tratta di una fase di equilibrazione, che è bene trascurare. Al termine dei passi, si vedrà che i punti del random walk sono distribuiti secondo la distribuzione di probabilità $\mathrm{P}(\mathbf{R})$.

Per capire come funziona l'algoritmo, possiamo immaginare un grande numero di walker che eseguono random walk in accordo con le regole precedenti. Dopo un certo periodo di equilibrazione, i walker raggiungeranno uno stato di equilibrio dinamico, in cui indichiamo con $\mathrm{n}(\mathbf{R})\mathrm{d}\mathbf{R}$ il numero medio di walker nell'elemento di volume $\mathrm{d}\mathbf{R}$. Si può dimostrare che all'equilibrio il numero medio di spostamenti dall'elemento di volume $\mathrm{d}\mathbf{R}$a $\mathrm{d}\mathbf{R}'$è uguale al numero di spostamenti avvenuti da $\mathrm{d}\mathbf{R}'$ a $\mathrm{d}\mathbf{R}$.

La probabilità che un walker posizionato in $\mathbf{R}$ arrivi nell'elemento di volume $\mathrm{d}\mathbf{R}'$allo step successivo è

$$d\mathbf{R}'A(\mathbf{R}' \leftarrow \mathbf{R})T(\mathbf{R}' \leftarrow \mathbf{R}) \tag{1.1.7}$$

allora il numero medio di walker che si spostano da $\mathrm{d}\mathbf{R}$ a $\mathrm{d}\mathbf{R}'$ in un singolo step sarà

$$d\mathbf{R}'A(\mathbf{R}' \leftarrow \mathbf{R})T(\mathbf{R}' \leftarrow \mathbf{R})n(\mathbf{R})\mathrm{d}\mathbf{R} \tag{1.1.8}$$

Imponendo che questo numero sia bilanciato dal numero dei walker che si spostano da $\mathrm{d}\mathbf{R}'$a $\mathrm{d}\mathbf{R}$,

$$d\mathbf{R}'A(\mathbf{R}' \leftarrow \mathbf{R})T(\mathbf{R}' \leftarrow \mathbf{R})n(\mathbf{R})\mathrm{d}\mathbf{R} = d\mathbf{R}A(\mathbf{R} \leftarrow \mathbf{R}')T(\mathbf{R} \leftarrow \mathbf{R}')n(\mathbf{R}')\mathrm{d}\mathbf{R}' \tag{1.1.9}$$

si ha quindi

$$\frac{n(\mathbf{R})}{n(\mathbf{R}')} = \frac{A(\mathbf{R} \leftarrow \mathbf{R}')T(\mathbf{R} \leftarrow \mathbf{R}')}{A(\mathbf{R}' \leftarrow \mathbf{R})T(\mathbf{R}' \leftarrow \mathbf{R})} \tag{1.1.10}$$

Dalla (4) si ha che il rapporto tra le probabilità di accettazione è

$$\frac{A(\mathbf{R} \leftarrow \mathbf{R}')}{A(\mathbf{R}' \leftarrow \mathbf{R})} = \frac{T(\mathbf{R}' \leftarrow \mathbf{R})P(\mathbf{R})}{T(\mathbf{R} \leftarrow \mathbf{R}')P(\mathbf{R}')} \tag{1.1.11}$$

da cui

$$\frac{n(\mathbf{R})}{n(\mathbf{R}')} = \frac{P(\mathbf{R})}{P(\mathbf{R}')} \tag{1.1.12}$$

Si ha allora che la densità n($\mathbf{R}$)dei walker all'equilibrio è proporzionale alla probabilità P($\mathbf{R}$),e la probabilità di trovare un dato walker in d$\mathbf{R}$ è P($\mathbf{R}$)d($\mathbf{R}$) come richiesto.

1.2 Il metodo Variational Monte Carlo

1.2.1 Il metodo

Il metodo Variational Monte Carlo nasce, come suggerisce il nome, dall'uso combinato di un principio variazionale e dell'integrazione numerica con il metodo Monte Carlo.

Per usare il metodo variazionale, è richiesta una funzione d'onda di prova Ψ_T, scelta ragionevolmente in modo da rispettare le caratteristiche che dovrebbe avere la vera funzione d'onda. La funzione ψ_T inoltre deve soddisfare alcune condizioni:

- Ψ_Te $\nabla\Psi_T$devono essere continue dovunque il potenziale sia finito
- Devono esistere gli integrali $\int \Psi_T^* \Psi_T$, $\int \Psi_T^* \hat{H} \Psi$ e $\int \Psi_T^* \hat{H^2} \Psi_T$

Scelta Ψ_T, il valore di aspettazione dell'Hamiltoniana E_V calcolato usando la funzione di prova, costituisce, per il principio variazionale, un limite superiore per il valore reale dell'energia dello stato fondamentale:

$$E_V = \frac{\int d\boldsymbol{R}\, \Psi_T^*(\boldsymbol{R}) \hat{H} \Psi(\boldsymbol{R})}{\int d\boldsymbol{R}\, \Psi_T^*(\boldsymbol{R}) \Psi_T(\boldsymbol{R})} \geq E_0 \tag{1.2.1}$$

Questo integrale può essere calcolato numericamente usando il metodo Monte Carlo e l'algoritmo di Metropolis. L'integrale nell'eq.1.2.1 può essere infatti riscritto come

$$E_V = \frac{\int d\boldsymbol{R}\, |\Psi(\boldsymbol{R})|^2 [(\Psi(\boldsymbol{R})^{-1} \hat{H} \Psi_T(\boldsymbol{R})]}{\int d\boldsymbol{R}\, |\Psi(\boldsymbol{R})|^2} \tag{1.2.2}$$

Il termine dell'integrando $P(\boldsymbol{R}) = \frac{|\Psi(\boldsymbol{R})|^2}{\int d\boldsymbol{R} |\Psi(\boldsymbol{R})|^2}$ può essere interpretato come una distribuzione di probabilità, mentre il secondo termine $E_L = \Psi_T(\boldsymbol{R})^{-1} \hat{H} \Psi_T(\boldsymbol{R})$ viene chiamato «energia locale». L'integrale

$$\int d\boldsymbol{R}\, P(\boldsymbol{R}) E_L(\boldsymbol{R}) \tag{1.2.3}$$

è del tutto analogo all'integrale (1.1.1) , e può essere calcolato generando con l'algoritmo di Metropolis un set di punti $\{\boldsymbol{R}_m : m \in [1, M]\}$ con distribuzione $P(\boldsymbol{R})$ e approssimare l'integrale con la media $\frac{1}{M} \sum_{m=1}^{M} E_L(\boldsymbol{R}_m)$

$$E_V \approx \frac{1}{M} \sum_{m=1}^{M} E_L(\boldsymbol{R}_m) \tag{1.2.4}$$

Si noti che se Ψ_T fosse una autofunzione di $\hat{H}$

$$\hat{H} \Psi_T = E_i \Psi_T \tag{1.2.5}$$

allora

$$E_L(\boldsymbol{R}) = E_i \; \forall \boldsymbol{R} \tag{1.2.6}$$

quindi la varianza sarebbe nulla e si otterrebbe l'autovalore $E_i$come unica valutazione di $E_L(\boldsymbol{R})$ in una qualunque posizione $\boldsymbol{R}$.Se $\Psi_T(\boldsymbol{R})$ fosse esattamente l'autofunzione del minimo livello energetico E_0allora nella relazione (1.2.1) sarebbe soddisfatta l'uguaglianza.

In generale, le parametrizzazioni di $\Psi_T(\boldsymbol{R})$ non permetteranno di descrivere esattamente l'autofunzione di minima energia, ma tanto meglio verrà approssimata e tanto più piccole saranno l'energia e la varianza associata. Pertanto i parametri di Ψ_T sono solitamente ottenuti dalla minimizzazione dell'energia variazionale $\langle E_L(\boldsymbol{R})\rangle$ $|\Psi(\boldsymbol{R})|^2$o della varianza $VAR[E_L(\boldsymbol{R})]$.

1.2.2 Le funzioni d'onda di Slater-Jastrow

L'accuratezza che si ottiene con il metodo VMC dipende dal tipo di funzione di prova che si sceglie. Dato che le funzioni d'onda vengono campionate più volte durante una simulazione, dalla scelta di queste dipenderà anche il costo computazionale della simulazione. Si cercano quindi delle funzioni d'onda che siano allo stesso tempo accurate e semplici da simulare.

Nei metodi di quantum chemistry si cerca di rappresentare le funzioni d'onda usando combinazioni lineari di determinanti di Slater. Queste espansioni convergono lentamente alla funzione esatta e non sono efficienti nel rappresentare le cuspidi che si hanno nel momento in cui due elettroni si avvicinano. Nei metodi QMC si usa perciò una rappresentazione più compatta nel rappresentare la correlazione fra elettroni: le funzioni d'onda Slater-Jastrow.

Queste sono composte da un solo determinante di Slater moltiplicato per un fattore di correlazione simmetrico e non negativo, detto fattore di Jastrow:

$$\Psi(\boldsymbol{X}) = e^{J(\boldsymbol{X})} D(\boldsymbol{X}) \tag{1.2.7}$$

dove $D(\boldsymbol{X})$ è un determinante di Slater e $\boldsymbol{X} = (x_1, ...x_n)$, con $x_i = (r_i, \sigma_i)$ è un vettore che determina le coordinate spaziali e di spin.

Il determinante di Slater da utilizzare viene determinato attraverso altri metodi, ad esempio Hartree-Fock, mentre per il fattore di Jastrow si sceglie solitamente una forma del tipo

$$J(\boldsymbol{X}) = \sum_{i=1}^{N} \chi(x_i) - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} u(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{1.2.8}$$

dove la funzione χdescrive la correlazione elettrone-nucleo, mentre il termine u descrive la correlazione elettrone-elettrone. La forma che si sceglie per queste funzioni varia a seconda del sistema che si sta studiando.

1.3 Il metodo Diffusion Monte Carlo

Il Diffusion Monte Carlo è un metodo proiettivo per ottenere lo stato fondamentale dell'equazione di Schrodinger, basato sull'evoluzione a tempo immaginario. Si utilizza la somiglianza della stessa con un'equazione di diffusione. A differenza del metodo VMC, il DMC non richiede in principio l'uso di funzioni di prova, almeno nella sua più semplice formulazione, che sarà discussa e implementata in questa tesi.

Esiste anche una formulazione del DMC con campionamento d'importanza, che di fatto è il metodo solitamente utilizzato per problemi di struttura elettronica. Questa seconda formulazione non verrà discussa in questa tesi in quanto richiederebbe un lavoro di implementazione molto più lungo e complesso.

Il metodo può essere diviso idealmente in tre passaggi, che saranno approfonditi nei paragrafi successivi:

1. Equazione di Schrodinger in tempo immaginario In questo passaggio si effettua nell'equazione di Schrodinger del sistema considerato, una trasformazione dal tempo reale t al tempo immaginario $T = it$, facendo dunque la sostituzione $t \to -iT$. E si dimostrerà che la funzione d'onda converge allo stato fondamentale per $T \to \infty$.

$$\lim_{T\to\infty} \Phi(T) = \Phi_0 \tag{1.3.1}$$

2. Formalismo integrale e integrazione Monte Carlo Nel secondo passaggio si scrive l'equazione di Schrodinger nel tempo immaginario in forma integrale. In questa forma, la soluzione del problema si traduce nel valutare un integrale multidimensionale con un approccio Monte Carlo.

Il metodo Monte Carlo per il calcolo dell' integrale relativo alla funzione d'onda viene implementato in questo caso campionando la stessa con un set discreto di configurazioni o «walker», i quali saranno soggetti a processi di diffusione, e di «nascita e morte» . La distribuzione dei walker convergerà dopo diversi passaggi ad una densità di probabilità che rappresenta lo stato fondamentale del sistema.

3. Stima dell'energia e campionamento dello stato fondamentale del sistema La distribuzione spaziale dei walker ad ogni step, costituisce un'approssimazione della funzione d'onda ad un tempo dato. La funzione d'onda converge poi allo stato fondamentale del sistema, se e solo se si fa partire la scala dell'energia dall'energia dello stato fondamentale. Dato che quest'ultima rappresenta un'incognita del problema, si parte da un valore ragionevole, e si migliora la propria stima dell'energia ad ogni passaggio, finchè l'energia non converge a quella dello stato fondamentale, e la distribuzione dei walker converge alla relativa funzione d'onda.

1.3.1 Equazione di Schrodinger in tempo immaginario

Si consideri l'equazione di Schrodinger

$$i\partial_t \Phi(\mathbf{R}, t) = \hat{H}\Phi(\mathbf{R}, t) \tag{1.3.2}$$

con $H = \frac{1}{2}\nabla^2 + V$ e sono state usate le unità atomiche di Hartree: $\mathrm{e} = \mathrm{m_e} = 4\pi\epsilon_0 = \hbar = 1$.

Ponendo $T = it$, l'equazione assume la forma

$$-\partial_T \Phi(\mathbf{R}, T) = \hat{H}\Phi(\mathbf{R}, T) \tag{1.3.3}$$

e si ha

$$\Phi(\mathbf{R}, T) = e^{-T\hat{H}}\Phi(\mathbf{R}, 0) \tag{1.3.4}$$

A questo punto bisogna introdurre uno shift della scala di energia. Ovviamente questo, essendo l'energia definita a meno di una costante, non cambierà la fisica del sistema, ma risulta fondamentale per la costruzione del metodo. Facciamo quindi la sostituzione

$$V(\boldsymbol{R}) \to V(\boldsymbol{R}) - E_t; \qquad \epsilon \to \epsilon_n - E_t$$

dove E_t è una costante e gli $\epsilon_n$sono autovalori dell'Hamiltoniana.

1.3.2 Proiezione sullo stato fondamentale

Si può far vedere che per $T \to \infty$ l'operatore $e^{-T\hat{H}}$ proietta lo stato $|\Phi(\mathbf{R},T)\rangle$ sullo stato fondamentale, o comunque sull' autostato ad energia più bassa tale che $\langle\Psi_0|\Phi(\mathbf{R},0)\rangle \neq 0$.

Infatti, usando la relazione di completezza si ha

$$|\Phi(T)\rangle = e^{-T\hat{H}} \sum_n |\Psi_n\rangle\langle\Psi_n|\Phi(0)\rangle \tag{1.3.5}$$

che può essere riscritta come

$$|\Phi(\tau)\rangle = e^{-T(\epsilon_o - E_t)} c_0|\Psi_o\rangle + \sum_1^\infty e^{-T(\epsilon_n - E_t)} c_n|\Psi_n\rangle \tag{1.3.6}$$

dove $c_n = \langle\Psi_n|\Phi(0)\rangle$.

Scegliendo $E_t = \epsilon_0$ l'esponenziale al primo addendo diviene costante, e gli altri termini della serie decadono esponenzialmente.

Si ha quindi

$$\lim_{T\to\infty} |\Phi(T)\rangle = c_0|\Phi(0)\rangle \tag{1.3.7}$$

Dunque $\Phi(\tau)$ converge allo stato fondamentale.

E' importante notare che se non fosse stata fatta la scelta $E_t = \epsilon_0$, $\Phi(T)$ non avrebbe avuto un limite finito e non nullo.

Infatti, per $E_t > \epsilon_0 \Rightarrow lim_{T\to\infty}\Phi(T) = \infty$, per $E_t < \epsilon_0 \Rightarrow lim_{T\to\infty}\Phi(T) = 0$

1.3.3 Funzione di Green

Per rappresentare la soluzione $\Phi(\boldsymbol{R}$,T) con un approccio stocastico, conviene riscrivere l'eq. (1.3.2) in forma integrale. Allo scopo è utile ricordare la definizione di funzione di Green.

Un'equazione differenziale lineare può essere posta genericamente nella forma

$$\hat{L}\Phi(\boldsymbol{R}) = u(\boldsymbol{R}) \tag{1.3.8}$$

dove $\hat{L}$è un operatore differenziale lineare.

La distribuzione G, appartenente allo spazio delle distribuzioni D', che soddisfa l'equazione

$$\hat{L}G = \delta(\boldsymbol{R}) \tag{1.3.9}$$

è detta funzione di Green o soluzione fondamentale dell'operatore differenziale $\hat{L}$.

Vale inoltre il seguente **teorema:**

sia $u \in D'$ tale che esista in D' il prodotto di convoluzione $G * u$, allora la soluzione dell'eq. (1.3.7) esiste in D' ed è data dalla formula

$$\Phi = G * u$$

1.3.4 Formalismo integrale e discretizzazione del tempo

L'eq. di Schrodinger (1.3.2) nel tempo immaginario, può essere riscritta in forma integrale usando la funzione di Green:

$$\Phi(\boldsymbol{R},T)=\int G(\boldsymbol{R},\boldsymbol{R'},T)\Phi(\boldsymbol{R'},0)d\boldsymbol{R'} \tag{1.3.10}$$

Scegliendo $T=n\tau$ quest'espressione può essere riscritta in modo ricorsivo:

$$\Phi(\boldsymbol{R_1},\tau)=\int G(\boldsymbol{R_0},\boldsymbol{R_1},\tau)\Phi(\boldsymbol{R_0},0)d\boldsymbol{R_0}$$

$$\Phi(\boldsymbol{R_2},2\tau)=\int G(\boldsymbol{R_1},\boldsymbol{R_2},\tau)\Phi(\boldsymbol{R_1},\tau)d\boldsymbol{R_1}$$

$$\vdots$$

$$\Phi(\boldsymbol{R_n},n\tau)=\int G(\boldsymbol{R_{n-1}},\boldsymbol{R_n},\tau)\Phi(\boldsymbol{R_{n-1}},(n-1)\tau)d\boldsymbol{R_{n-1}}$$

così da poter dividere il processo in un numero discreto di step, ognuno dei quali può essere simulato al calcolatore, valutando gli integrali in cui è stata scomposta la (17) con un approccio Monte Carlo.

1.3.5 Analogia con l'equazione di diffusione

Per trovare $\Phi(\tau)$ bisognerà comprendere il processo governato dall'equazione e simularlo in modo stocastico. Per farlo, si studierà la funzione di Green, che compare nella forma integrale, analizzando prima separatamente la parte cinetica e quella relativa al potenziale per poi discutere l'equazione con l'Hamiltoniana completa.

Trascuriamo quindi per un momento il termine di potenziale e consideriamo solo la parte cinetica dell'Hamiltoniana. L'equazione diventa:

$$-\partial_\tau\Phi(\boldsymbol{R},T)=\frac{1}{2}\nabla^2\Phi(\boldsymbol{R},T) \tag{1.3.11}$$

ovvero un'equazione di diffusione.

Studiamo l'equazione nel dettaglio:

$$\Phi(\boldsymbol{R},T)=\langle\boldsymbol{R}|\Phi(T)\rangle=\langle\boldsymbol{R}|e^{-T\frac{\nabla^2}{2}}|\Phi(0)\rangle \tag{1.3.12}$$

Usando la relazione di completezza $I=\int d\boldsymbol{R'}\,|\boldsymbol{R'}\rangle\langle\boldsymbol{R'}|$ si può riscrivere come

$$\int d\boldsymbol{R'}\,\langle\boldsymbol{R}|e^{-T\frac{\nabla^2}{2}}|\boldsymbol{R'}\rangle\langle\boldsymbol{R'}|\Phi(0)\rangle=\int d\boldsymbol{R'}\,\langle\boldsymbol{R}|e^{-T\frac{\nabla^2}{2}}|\boldsymbol{R'}\rangle\Phi(\boldsymbol{R'},0) \tag{1.3.13}$$

e si può ricavare la funzione di Green per analogia con l'eq. 1.2.8:

$$G_D=\langle\boldsymbol{R}|e^{-T\frac{\nabla^2}{2}}|\boldsymbol{R'}\rangle \tag{1.3.14}$$

Per trovarne un'espressione esplicita si può usare nuovamente la relazione di completezza, questa volta espressa in funzione degli operatori di proiezione sugli autostati dell'impulso: $I = \int d\boldsymbol{p}\, |\boldsymbol{p}\rangle\langle\boldsymbol{p}|$.Si ottiene

$$G_D = \int d\boldsymbol{p}\, \langle \boldsymbol{R}|e^{-T\frac{\nabla^2}{2}}|\boldsymbol{p}\rangle\langle\boldsymbol{p}|\boldsymbol{R'}\rangle \tag{1.3.15}$$

e ricordando che $\langle\boldsymbol{p}|\boldsymbol{R'}\rangle = \frac{1}{\sqrt{2\pi}} e^{i\boldsymbol{p}\cdot\boldsymbol{R'}}$ si ha

$$G_D = \int d\boldsymbol{p} \left(e^{-T\frac{p^2}{2}}\right)\left(\frac{1}{\sqrt{2\pi}} e^{i\boldsymbol{p}\cdot(\boldsymbol{R'}-\boldsymbol{R})}\right) \tag{1.3.16}$$

dove si è indicato con p l'autovalore dell'operatore impulso: $\hat{p}|p\rangle = p|p\rangle$.

L'integrale nell'eq. 1.2.14 è un integrale gaussiano e risulta

$$G_D(\boldsymbol{R},\boldsymbol{R'},T) = (2\pi T)^{-\frac{3N}{2}} e^{-\frac{|\boldsymbol{R}-\boldsymbol{R}|'^2}{2T}} \tag{1.3.17}$$

ovvero una gaussiana di varianza T per ogni dimensione.

Sostituendo l'espressione trovata per la soluzione fondamentale nella 1.2.11 si ottiene la soluzione $\Phi(\boldsymbol{R},T)$:

$$\Phi(\boldsymbol{R},T) = \int d\boldsymbol{R'}\, (2\pi T)^{-\frac{3N}{2}} e^{-\frac{|\boldsymbol{R}-\boldsymbol{R}|'^2}{2T}} \Phi(\boldsymbol{R'},0) \tag{1.3.18}$$

Se si sceglie, ad esempio, come condizione iniziale $\Phi(\boldsymbol{R'},0) = \delta(\boldsymbol{R'} - \boldsymbol{R_0})$ si ha:

$$\Phi(\boldsymbol{R},T) = (2\pi T)^{-\frac{3N}{2}} e^{-\frac{|\boldsymbol{R}-\boldsymbol{R_0}|^2}{2T}} \tag{1.3.19}$$

cioè una distribuzione la cui varianza aumenta al passare del tempo e che quindi può rappresentare un processo di diffusione.

Si può dimostrare che la soluzione $\Phi(\boldsymbol{R},\tau)$ descrive anche la distribuzione di particelle in moto browniano nello spazio e nel tempo.

Ciò permette di simulare stocasticamente la diffusione rappresentando l'evoluzione della distribuzione con un set discreto di random walker:

$$\Phi(\boldsymbol{R},T) \to \sum_k \delta(\boldsymbol{R} - \boldsymbol{R_k}) \tag{1.3.20}$$

e interpretando la funzione di Green $G_D(\boldsymbol{R}, \boldsymbol{R'},T)$ come una densità di probabilità di transizione per l'evoluzione dei walker.

Infatti, scrivendo la (1.2.8) in forma integrale si ha

$$\Phi(\boldsymbol{R},T+\tau) = \int G(\boldsymbol{R},\boldsymbol{R'},\tau)\Phi(\boldsymbol{R'},T)d\boldsymbol{R'} \tag{1.3.21}$$

e sostituendo nella (1.2.9) si ottiene

$$\Phi(\boldsymbol{R},T+\tau) \to \sum_k G(\boldsymbol{R},\boldsymbol{R_k},\tau) \tag{1.3.22}$$

ovvero una sovrapposizione di gaussiane di varianza 3τ.

Per discretizzare nuovamente la rappresentazione della funzione d'onda, si può campionare ogni gaussiana con una delta di Dirac, ottenendo così una nuova posizione per ogni walker. Iterando il procedimento si vedrà che le posizioni dei walker avranno una distribuzione al tempo T che ben approssima la soluzione dell'equazione di diffusione $\Phi(\boldsymbol{R},T)$.

1.3.6 Il ruolo del potenziale

Per comprendere il ruolo del potenziale, occorre evidenziare il suo ruolo nella funzione di Green dell'equazione di Schrodinger. Per semplicità si può procedere analizzando prima il caso di una Hamiltoniana con il solo termine di potenziale, per poi procedere con il caso dell'Hamiltoniana completa.

Trascurando per un attimo il termine cinetico, equazione di Schrodinger diventa

$$-\partial_\tau \Phi(\boldsymbol{R},T) = (\hat{V} - E_T)\Phi(\boldsymbol{R},T) \tag{1.3.23}$$

La funzione d'onda assumerà quindi la forma

$$\langle \boldsymbol{R}|\Phi(T)\rangle = \langle \boldsymbol{R}|e^{-T(V-E_t)}|\Phi(0)\rangle = e^{-T(V(\boldsymbol{R})-E_t)}|\langle \boldsymbol{R}|\Phi(0)\rangle \tag{1.3.24}$$

possiamo perciò associare all'equazione (numero) la seguente funzione di Green:

$$G_V(\boldsymbol{R},\boldsymbol{R'},T) = e^{-T(V(\boldsymbol{R})-E_t)}\delta(\boldsymbol{R}-\boldsymbol{R'}) \tag{1.3.25}$$

Adesso si può considerare la funzione di Green relativa all'Hamiltoniana completa $\hat{H} = \hat{K} + \hat{V} - E_t$, dove $\hat{K}$è l'operatore relativo all' energia cinetica:

$$G(\boldsymbol{R},T) = \langle \boldsymbol{R}|e^{-T(\hat{K}+\hat{V}-E_t)}|\boldsymbol{R'}\rangle \tag{1.3.26}$$

Il calcolo esplicito della (1.3.26) risulta impossibile, ma si può semplificare l'espressione ricorrendo all'approssimazione di Trotter-Suzuki $e^{-T(\hat{A}+\hat{B})} = e^{-T\frac{\hat{B}}{2}}e^{-TA}e^{-T\frac{\hat{B}}{2}} + O(\tau^3)$ e scegliendo $\hat{A} = \hat{K}$ e $\hat{B} = \hat{V} - E_t$.

In questo modo si ha

$$G(\boldsymbol{R},\boldsymbol{R'},T) \approx e^{-T(V(\boldsymbol{R})-E_t)/2}\langle \boldsymbol{R}|e^{-T\hat{K}}|\boldsymbol{R'}\rangle e^{-T(V(\boldsymbol{R'})-E_t)/2} \tag{1.3.27}$$

e, riprendendo il risultato relativo all'equazione di diffusione, si ottiene

$$G(\boldsymbol{R},\boldsymbol{R'},T) \approx (2\pi\tau)^{-\frac{3N}{2}} e^{-\frac{|\boldsymbol{R}-\boldsymbol{R}|'^2}{2\tau}} e^{-T(V(\boldsymbol{R})+V(\boldsymbol{R'})-2E_t)/2} \tag{1.3.28}$$

che è valida fino al secondo ordine in τ.

Analizzando la funzione di Green in questa forma, si riconosce subito un termine uguale alla funzione di Green dell'equazione di diffusione, che questa volta è moltiplicato per il termine

$$P = e^{-\tau(V(\boldsymbol{R})+V(\boldsymbol{R'})-2E_t)/2} \tag{1.3.29}$$

che introduce un peso dipendente dal tempo e dal potenziale V.

Con le approssimazioni fatte per la funzione di Green, l'errore risulta essere dell'ordine $O(n\tau^3) = O(\tau^2 T)$, e può quindi essere controllato con un'opportuna scelta del timestep τ.

1.3.7 Il processo di branching

In linea di principio, ci sono vari modi di introdurre nella simulazione il reweighting della distribuzione dei walker. Si potrebbe pensare , ad esempio, di assegnare ad ogni walker un peso iniziale e registrare il prodotto dei pesi che gli vengono assegnati durante la propagazione. Questo processo non è efficiente nella pratica, poichè i pesi dei walker diventano rapidamente molto diversi tra loro, e il campionamento risulta inefficiente, dato che si utilizzano uguali risorse per propagare i walker con peso grande e quelli con peso piccolo, che spesso diventa pressochè nullo e quindi trascurabile.

Un processo più efficace è dato dall'algoritmo "di branching" o di nascita e di morte": si sceglie di far "nascere" o "morire" dei walker in base al peso che è stato loro assegnato.

In particolare:

- Se $P < 1$ il walker rimane "in vita" con probabilità P.
- Se $P \geq 1$ il walker rimane in vita e ne viene generato un secondo con probabilità P-1 nella stessa posizione, un terzo con probabilità P-2 ecc.

Queste due condizioni possono essere implementate più facilmente chiedendo

$$M_{new} = INT(P + \eta) \tag{1.3.30}$$

dove M_{new}è il numero di walker che si avranno al prossimo step in una data posizione, e η è un numero casuale con distribuzione uniforme nell'intervallo [0,1].

In questo modo si svilupperà una maggiore densità nelle regioni ad energia più bassa, mentre i walker tenderanno a "morire" nelle regioni ad energia più alta.

1.3.8 Il parametro E_T

Nella sezione 1.2.2 è stato discussa la convergenza della soluzione dell'equazione di Schrodinger nel tempo immaginario allo stato fondamentale del sistema considerato. Si è visto anche come questa convergenza avviene solo se si sceglie $E_t = \epsilon_0$, diversamente la funzione d'onda si annulla o diverge asintoticamente.

Nella simulazione, dove la funzione d'onda viene approssimata dalla distribuzione spaziale dei walker, il caso in cui la funzione d'onda si annulla corrisponde al caso in cui muoiono tutte le particelle. Il caso in cui la funzione d'onda diverge invece corrisponde al caso in cui la popolazione dei walker cresce in maniera improvvisa così da occupare velocemente tutta lo spazio di memoria reso disponibile dal calcolatore. Nel caso si scelga invece $E_t = \epsilon_0$, ci si aspetta il numero dei walker fluttuare attorno un valore costante.

Nella risoluzione del problema, in realtà, ϵ_0 è un'incognita del sistema, ma le considerazioni fatte ci forniscono una metodologia per farne una stima: si può partire da un valore arbitrario di E_T e aggiustare questo valore ad ogni step con un algoritmo, richiedendo che E_t sia tale da mantenere stabile il numero delle particelle.

Capitolo 2

L'algoritmo

In questo capitolo verranno discussi i punti principali dell'algoritmo sviluppato durante il lavoro di tesi per implementare il metodo DMC. In appendice si può trovare il codice a cui si fa riferimento.

2.1 Schema dell'algoritmo

Per implementare la simulazione c'è bisogno di un algoritmo che metta insieme gli elementi discussi nel primo capitolo e ci permetta di implementare i processi descritti mediante l'uso di numeri pseudocasuali. Nell'algoritmo sviluppato in questa tesi i passi principali sono:

- **Input:** in questo passo l'utente inserisce in input sia informazioni legate al problema fisico, come ad esempio il potenziale del sistema da rappresentare, sia preferenze legate alla simulazione, ad esempio il valore iniziale del parametro E_t e il numero di walker target $N_w$con cui si vuole rappresentare la funzione d'onda.
- **Inizializzazione:** viene allocata una matrice in cui vengono registrate le posizioni iniziali dei walker. Tali posizioni sono generate casualmente secondo una distribuzione uniforme o gaussiana, scelta dall'utente in input.
- **Diffusione:** Ogni walker compie un passo di random walk, ovvero si aggiornano le posizioni registrate nella matrice con un opportuno numero casuale. Parallelamente vengono calcolati i potenziali relativi alle posizioni dei walker prima e dopo lo step di diffusione.
- **Branching:** I walker vengono sottoposti ad un processo di nascita e di morte sulla base del loro peso. Per implementare questo processo si introduce una matrice ausiliaria in cui le posizioni di ogni walker vengono copiate M_{new}volte. Una volta completato il branching, si ricalcola il parametro E_t in base all'evoluzione della popolazione dei walker. I processi di diffusione e branching si alternano in un loop, finchè non si raggiunge un numero di passi N_{steps} stabilito dall'utente.
- **Output:** Raggiunto il numero di step prestabilito viene generato un file contenente i valori di E_t e della loro media E_R ottenuti durante i vari step della simulazione,

e degli istogrammi contenenti informazioni sull'evoluzione della distribuzione delle posizioni dei walker.

Nei primi passi , prima che i walker raggiungano uno stato di equilibrio dinamico, il parametro E_t compie grandi oscillazioni, prima di iniziare ad oscillare attorno un valore costante. C'è quindi una fase di equilibrazione dove i valori di E_t vengono trascurati.

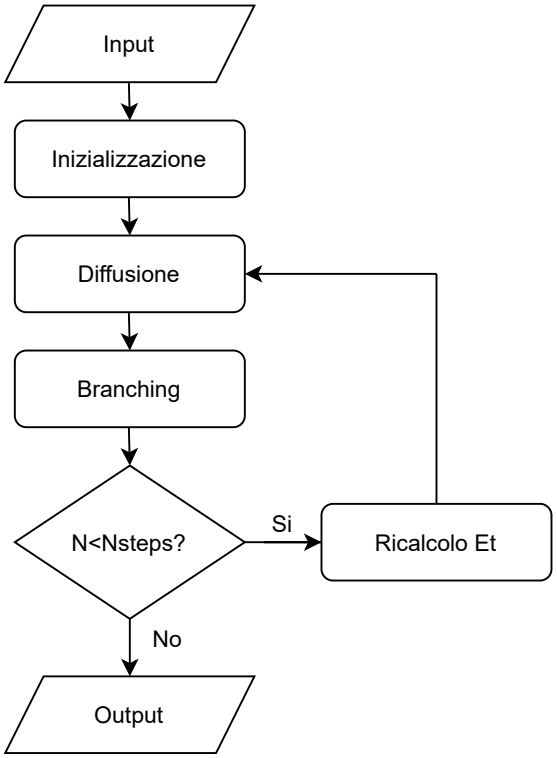

Figura 2.1.1: Flowchart dell'algoritmo usato per implementare il metodo Diffusion Monte Carlo

2.1.1 Input e inizializzazione

In fase di input vengono richiesti all'utente i seguenti parametri:

- Valore del time-step τ, tipicamente vale 0.1 a.u. o meno
- Dimensionalità del problema D
- Numero di step da eseguire N_{steps}
- Numero di walker iniziale N_w (che sarà anche un target, dato che si cerca di far oscillare il numero di walker N_{wt} attorno ad un valore costante)
- Distribuzione iniziale delle posizioni dei walker (si può scegliere uniforme o gaussiana)

- Potenziale V (E' stato implementato il caso del potenziale armonico e quello dell'atomo di idrogeno)
- I parametri $Rmax$ e N_{bin} dell'istogramma che servirà a rappresentare lafunzione d'onda.

Il valore iniziale di E_t, che di norma può essere scelto arbitrariamente, è stato impostato di default a $E_t = 0.0$ allo scopo di semplificare l'input. E' stato scelto un tempo di equilibrazione pari a 5 a.u.

Successivamente, per «generare» i walker e e inizializzare le loro posizioni, si alloca dinamicamente una matrice $w[i][j]$ di $(MaxNwf \times N_w)$ righe e D colonne, in cui ogni riga fino alla $N_w - esima$, contiene le D coordinate cartesiane che individuano la posizione di un walker nello spazio. Le righe in più saranno utilizzate per registrare le posizioni di nuovi walker generati nei passi successivi. Gli elementi della matrice vengono inizializzati attraverso le funzioni *GeneraWalkerU* e *GeneraWalkerG* che assegnano ad ogni elemento della matrice un numero pseudocasuale con distribuzione rispettivamente uniforme, o gaussiana. Per le distribuzioni uniformi, è stata usata la funzione *rand*()presente nelle librerie del linguaggio C, mentre le distribuzioni gaussiane sono state ottenute con la funzione $norm(media, dev.standard)$ basata sul metodo box-muller.

Si noti che nel metodo che stiamo considerando, il risultato non dipende da $\Phi(\boldsymbol{R},0)$. Per questo, la scelta della distribuzione iniziale dei walker è totalmente arbitraria, a patto che non rappresenti una funzione d'onda ortogonale a quella dello stato fondamentale Φ_0.

2.1.2 Diffusione

Per simulare la diffusione, è sufficiente aggiornare la posizione di ogni walker in accordo con quanto detto nella sezione 1.2.4.

Detta $\boldsymbol{R_i^{(n)}} = (R_{i1}, ..., R_{iD})^{(n)}$ la posizione dell'i-esimo walker al passo n, la posizione dopo il processo di diffusione sarà

$$\boldsymbol{R_i^{(n+1)}} = (R_{i1}^n + \rho_1, ..., R_{ij}^n + \rho_D) \tag{2.1.1}$$

dove i numeri $\rho_1, ..., \rho_D$ sono numeri pseudocasuali estratti da una distribuzione gaussiana di media 0 e varianza τ.

Nel codice che è stato sviluppato quest'operazione è stata affidata alla funzione *Diffusione*.

Prima e dopo lo step di diffusione, viene calcolato il potenziale relativo alla posizione di ogni walker, attraverso la funzione V_i, i potenziali così ottenuti vengono registrati nelle componenti degli array A e B.

2.1.3 Branching

Il processo di nascita e di morte viene implementato nella funzione *Branching*, che esegue i seguenti passi:

1. Viene calcolato il peso P relativo ad ogni walker, secondo la 1.2.19.
2. Viene calcolato il numero di repliche M_i dell'iesimo walker che verranno generate alla fine del branching, in accordo con la 1.2.20. Si ha che per $M_i = 0$ il walker muore, per $M_i = 1$ il walker sopravvive, per $M_i > 1$ vengono generate $M_i - 1$ nuove

copie della particella. Tali numeri vengono registrati nelle componenti $M[i]$ di un array.

3. Viene calcolato il numero N_{new} di particelle che si avranno alla fine del processo: $N_{new} = \sum_i^{Nwt} M_i$.

4. Si alloca una matrice $w_{new}[i][j]$ delle stesse dimensioni di w.

5. Si avvia un ciclo in cui per ogni i, si legge il valore M_i e si copia M_i volte la posizione dell'iesimo walker nella nuova matrice w_{new}, ovvero si pone

$$w_{new}[i'][j] = w[i][j] \; \forall j \in [0, D-1] \tag{2.1.2}$$

6. La funzione restituisce w_{new} nel main e si pone $N_{wt} = N_{new}$

In questo modo si è ottenuta una nuova matrice w_{new} in cui N_{new} righe sono utilizzate per registrare delle posizioni. Nel codice sono presenti dei controlli per arrestare il programma nel caso $N_{new} = 0$ (muoiono tutti i walker) o $N_{new} > maxNwtf \times Nw$ (il numero di walker è tale da riempire tutto lo spazio di memoria che si è voluto dedicare per memorizzarne le posizioni)

2.1.4 Il ricalcolo di E_t

Completato il branching, è necessario ricalcolare il parametro E_t come descritto nella sezione 1.2.7. Si può intuire che la correzione deve essere abbastanza «veloce» da mantenere stabile il numero delle particelle ed evitare l'arresto del programma, ma anche abbastanza «lenta» da permettere la convergenza di E_t.

Una soluzione è scegliere una correzione logaritmica del tipo

$$E_t^{(n+1)} = E_R^{(n)} + \frac{1}{\tau} \ln\left(\frac{N_w}{N_{wt}}\right) \tag{2.1.3}$$

dove $E_R^{(n)}$ è la media ottenuta dalla sequenza $\{E_t^{(0)}, ..., E_t^{(n)}\}$ e può essere calcolata ricorsivamente come segue:

$$E_R^{(n+1)} = \frac{(n \times E_R^{(n)} + E_t)}{(n+1)} \tag{2.1.4}$$

La (2.1.3) è costruita in modo che $E_t^{(n+1)} < E_R^{(n)}$ *se* $N_{wt} > N_w$ e $E_t^{(n+1)} > E_R^{(n)}$ *se* $N_{wt} < N_w$.

Usare questo tipo di correzione fin dall'inizio della simulazione risulta però sconveniente, poichè la mediaE_R dipenderà anche dai primi valori di $E_t$ottenuti in fase di equilibrazione, rallentando quindi la convergenza.

Un modo per rendere il ricalcolo di E_t più efficace consiste nel dividere la fase di equilibrazione in blocchi da un certo numero di step (nel nostro caso abbiamo scelto $stepsperblock = 20$) e porre $n = 0$ nella (2.1.4) all'inizio di ogni blocco. In questo modo, le medie da usare come riferimento sono calcolate sui singoli blocchi e non sono affette dai valori di $E_t$ottenuti in precedenza.

Superata la fase di equilibrazione si può tornare ad applicare la (2.1.3) normalmente.

2.1.5 Output

I processi di diffusione e branching si alternano in un loop finchè non si raggiunge il numero di passi N_{steps} richiesto dall'utente. I dati relativi all'energia generati ad ogni passo E_t ed E_R, assieme ai dati necessari per generare degli istogrammi opportunamente normalizzati sulla distribuzione delle posizioni dei walker , vengono salvati su un apposito file.

Capitolo 3

Risultati

In questo capitolo si analizzerano i risultati ottenuti in due sistemi monoelettronici: l'oscillatore armonico e l'atomo di idrogeno. Tali risultati saranno confrontati con quelli ottenibili in modo analitico.

3.1 Oscillatore armonico

3.1.1 Risultati analitici

Il potenziale di un oscillatore armonico unidimensionale di pulsazione ω é (in unità SI):

$$V(x) = \frac{1}{2} m\omega^2 x^2 \tag{3.1.1}$$

Si è scelto di lavorare in unità atomiche e di scegliere il caso $\omega = 1$. In queste unità si ha $E_0 = 1/2$ a.u. e la funzione d'onda dello stato fondamentale risulta:

$$\Phi_0(x) = \pi^{-\frac{1}{4}} e^{-\frac{x^2}{2}} \tag{3.1.2}$$

3.1.2 Simulazione

Energia

Come spiegato in precedenza, in un algoritmo DMC si corregge iterativamente il parametro E_t in modo che il numero di walker resti stabile al variare del tempo. Si può poi ottenere una stima dell'energia E_R mediando i valori di E_t ottenuti durante la simulazione.

Nella figura seguente (3.1.1) sono riportati gli andamento di E_Re di E_t ottenuti usando l'algoritmo descritto nel capitolo precedente e scegliendo come parametri $\tau = 0.05$ a.u., $N_{steps} = 1500$, e $N_w = 10000$. Si è scelto $E_t = 3$ a.u. come valore iniziale.

Osservando il grafico, si può notare che per E_R dopo una fase iniziale di equilibrazione, si ha una rapida convergenza verso il valore analitico E_0, attorno al quale si hanno però ancora delle oscillazioni. Superata questa fase, E_R viene mediata in modo continuo e non più per blocchi, il valore di E_t viene corretto in base alla (2.1.4) e le fluttuazioni vengono ridotte notevolmente.

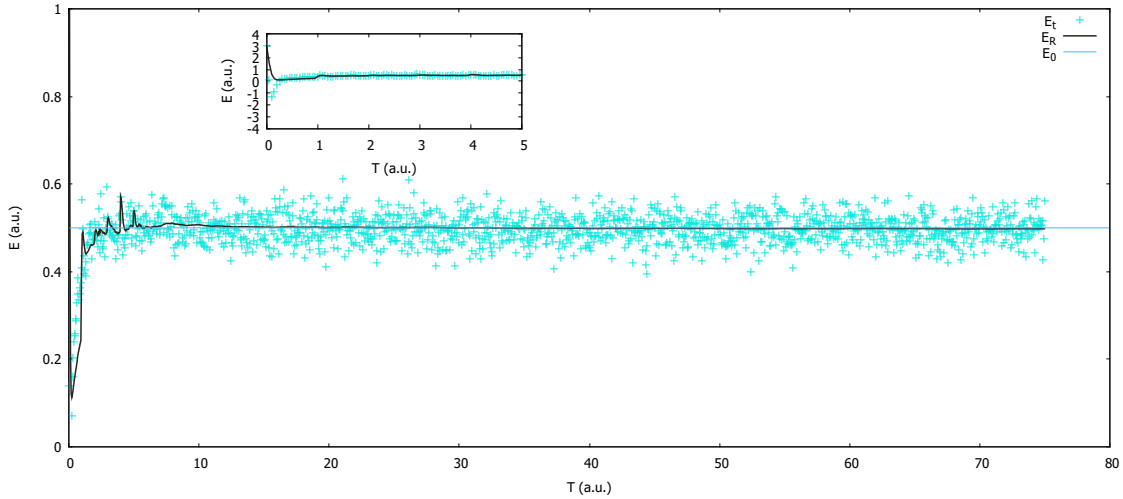

Figura 3.1.1: Andamento dell'energia (E_R ed E_t).Dall'inset si può notare che il valore di E_t è stato scelto fuori scala.

Funzione d'onda

Il DMC, a differenza di altri metodi quali il VMC, non richiede in linea di principio l'uso di una funzione d'onda di prova. Partendo da una distribuzione qualsiasi, la distribuzione dei walker converge gradualmente a quella dello stato fondamentale del sistema attraverso i passaggi di diffusione e branching.

Nella figure seguenti viene riportata la distribuzione istantanea delle posizioni dei walker campionata in diversi momenti della simulazione. I parametri scelti per l'istogramma sono: $N_{bin} = 200$ e $R_{max} = 10,0$ a.u. ed è stata scelta una distribuzione iniziale uniforme.

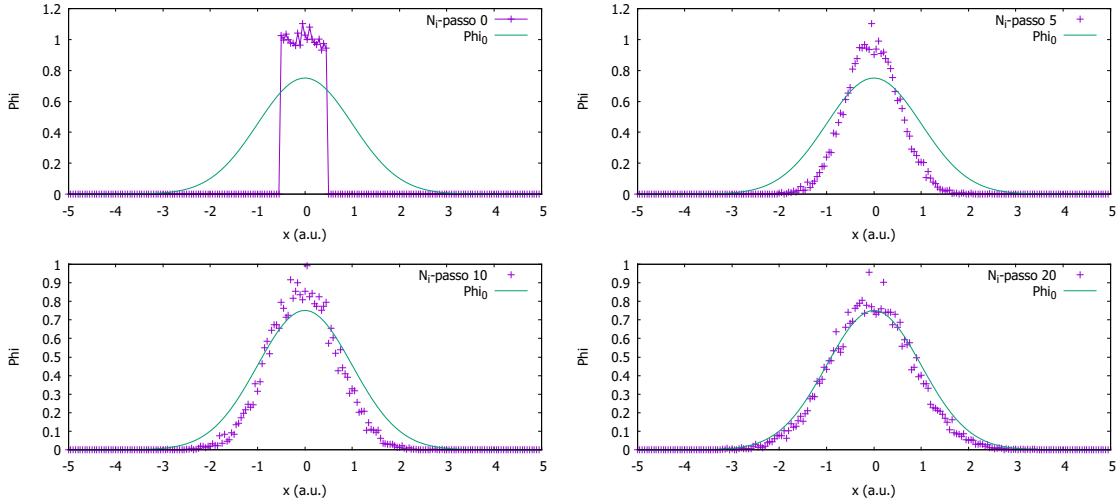

Figura 3.1.2: Distribuzione istantanea dei walker a 0,5,10 e 20 Monte Carlo Step

Superata la fase di equilibrazione, i walker raggiungono uno stato di equilibrio dinamico, e ci si aspetta che la loro distribuzione vari poco nel tempo. Questa considerazione permette di implementare un metodo di campionamento più efficace: si può campionare cumulativamente per più step per poi dividere per il numero di passi. In figura è mostrata la distribuzione dei walker all'equilibrio, campionata con questo criterio.

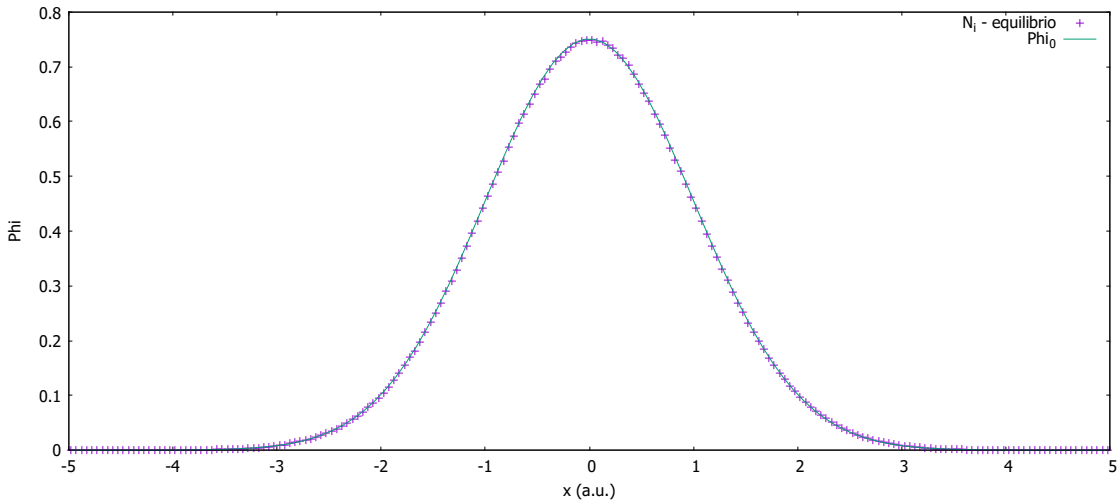

Figura 3.1.3: Distribuzione dei walker all'equilibrio

3.1.3 Analisi dei dati

La stima dell'incertezza sull'energia E_{best} presenta alcune complicazioni rispetto al caso mostrato nel capitolo 1. La simulazione Monte Carlo, aggiorna i valori delle variabili in gioco ad ogni step e ,soprattutto dopo che è stato raggiunto l'equilibrio, le configurazioni ottenute tra due step successivi possono essere significativamente correlate. In generale quindi, non si può stimare l'errore su una media $\langle A \rangle$ con la formula

$$\sigma_{<A>} \simeq \frac{\langle A^2 \rangle - \langle A \rangle^2}{\sqrt{M-1}} \tag{3.1.3}$$

dove si è indicato con M il numero di misure. Tale espressione infatti è valida sotto l'ipotesi che due misure successive siano scorrelate fra loro.

Sebbene sia possibile da un punto di vista teorico correggere questa stima calcolando il tempo di autocorrelazione τ_A, esiste un metodo alternativo più efficiente chiamato «**block average**» o «**reblocking**» . L'idea alla base è quella di dividere i dati in blocchi, e generare delle nuove variabili indipendenti mediando le misure sui singoli blocchi. Le nuove variabili avranno la stessa media di quelle da cui si è partiti, ma si potrà stimare l'errore con la formula (3.1.1.)

Reblocking

Si supponga di avere un set di M misure $\{A_t\}_{t=0,..,M-1}$, aventi un tempo di correlazione τ_A.

Per ottenere una stima corretta dell'errore sulla media si può procedere nel modo seguente:

- Si raggruppano le misure in blocchi contenenti b misure consecutive e si chiama $A_t^{(b)}$ la media delle misure del t-esimo blocco.

$$A_t^{(b)} = \frac{1}{b} \sum_{t'=bt}^{b(t+1)-1} A_{t'} \tag{3.1.4}$$

  e si avrà che il numero dei blocchi, ovvero il numero di misure nella sequenza che si è appena ottenuta,sarà M/b.[1] Si ha che per $b < \tau_A$ i dati della sequenza $\{A_t^{(b)}\}$ sono ancora correlati, mentre per $b > \tau_A$ i blocchi riescono a includere la correlazione temporale, e gli elementi della nuova sequenza diventano indipendenti.

- Si definisce l'incertezza sulla media $\sigma_{\langle A \rangle}^{(b)}$

$$\sigma_{\langle A \rangle}^{(b)} \equiv \sqrt{\frac{var(A_t^{(b)})}{(M/b-1)}} \tag{3.1.5}$$

  dove $var(A_t^{(b)})$ è la varianza della sequenza $\{A_t^{(b)}\}$:

$$var(A_t^{(b)}) = \frac{b}{M} \sum_{t=0}^{M/b-1} (A_t^{(b)})^2 - \langle A \rangle^2 \tag{3.1.6}$$

[1] per semplicità si è assunto M multiplo di b.

L'incertezza così definita, è corretta solo nel caso in cui le dimensioni dei blocchi siano tali da rendere gli elementi della sequenza $\{A_t^{(b)}\}$ indipendenti, in caso contrario la 3.1.2 sottostima l'incertezza sulla media.

Non conoscendo il tempo di autocorrelazione $\tau_A$si può capire quale deve essere la corretta dimensione dei blocchi guardando l'andamento di $\sigma^{(b)}_{\langle A\rangle}$in funzione di b: fintanto che $b < \tau_A$ la (3.1.2.) sottostima l'incertezza, perciò all'aumentare di b si osserverà un andamento crescente di $\sigma^{(b)}_{\langle A\rangle}$ fino a raggiungere un plateau quando $b \geq \tau_A$.Tale procedura può essere automatizzata con un ciclo in cui in ogni passo si raggruppano, ad esempio, per due gli elementi della sequenza. Nel codice scritto durante il lavoro di tesi è presente una funzione che implementa il reblocking, per l'analisi dati si è preferito però usare il modulo Pyblock di Python, che è ottimizzato per lo scopo.

Di seguito viene mostrato l'andamento dell'errore al variare del numero di passi di reblocking (e quindi della grandezza dei blocchi) ottenuto in una simulazione con parametri $\tau = 0.05$ a.u., $Nsteps = 30500$, e 500 passi di equilibrazione.

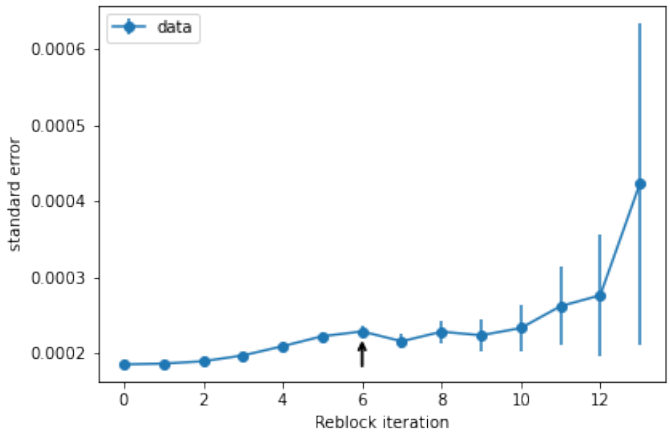

Figura 3.1.4: Andamento dell'errore in funzione del numero di passaggi di reblocking, ottenuto usando il modulo «Pyblock» di Python. La freccia indica il numero ottimale di iterazioni.

Si noti che non sempre è semplice osservare il plateau, perchè all'aumentare di b si riduce il numero di blocchi M/b e si ha di conseguenza un'incertezza maggiore sugli ultimi punti del grafico.

3.1.4 Risultati della simulazione

Si riportano di seguito i risultati ottenuti. Sono state effettuate più simulazioni al variare del time-step τ per evidenziare l'effetto dell'errore dovuto alla discretizzazione del tempo sui risultati del DMC.

$\tau\,(a.u.)$	N_w	$T(a.u.)$	$E\,(a.u.)$	$\sigma_E\,(a.u.)$
1	10000	10000	0.48119	0.00008
0.5	10000	10000	0.49510	0.00008
0.25	10000	10000	0.49866	0.00009
0.1	10000	10000	0.49993	0.00009
0.01	10000	10000	0.49993	0.00009

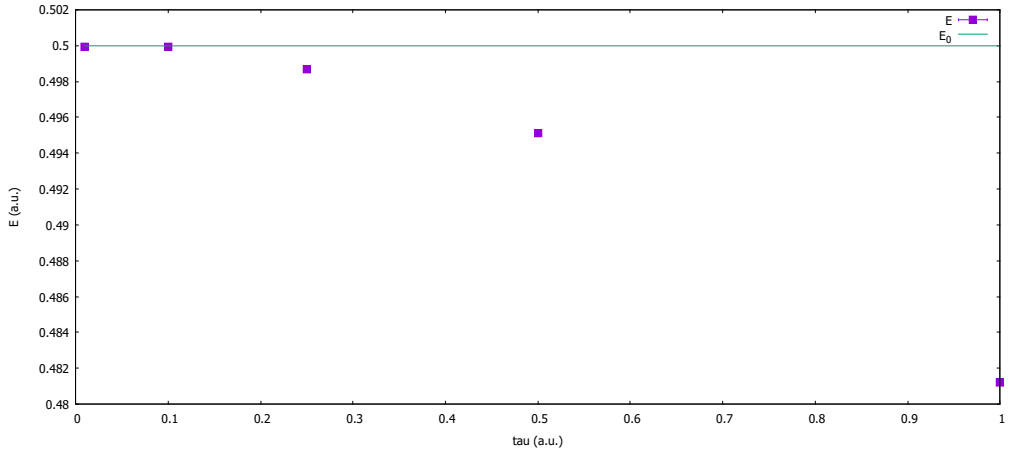

Figura 3.1.5: Andamento di E al variare del time-step τ

Si può notare come il risultato si avvicini a quello analitico al decrescere di τ. Diminuire il time-step, se si vuole mantenere la lunghezza totale T della simulazione costante, non è però esente da costi: per una simulazione di lunghezza totale T, il costo computazionale sarà proporzionale a $1/\tau$. Aumenterà inoltre la correlazione tra due Monte Carlo step successivi: bisognerà implementare il reblocking con dei blocchi più grandi e quindi aumentare considerevolmente il numero di step per avere la stessa statistica.

3.1.5 Il bias della popolazione

Oltre all'errore dovuto alla discretizzazione del tempo, nel DMC è presente anche un bias legato alla popolazione, avendo rappresentato la funzione d'onda con un set di walker discreto. Ci si aspetta che questo diminuisca al crescere della popolazione e per verificarlo sono state effettuate diverse simulazioni al variare del parametro N_w. E' stato scelto $\tau = 0.1$ come time-step poichè dai risultati precedenti l'energia ottenuta per l'oscillatore con questo time-step risulta già statisticamente indistinguibile da quella ottenuta con un τ di un ordine di grandezza più piccolo.

$\tau\,(a.u.)$	N_w	$T(a.u.)$	$E\,(a.u.)$	$\sigma_E\,(a.u.)$
0.1	10^2	100000	0.5041	0.0003
0.1	10^3	10000	0.5000	0.0003
0.1	10^4	10000	0.49993	0.00009
0.1	10^5	1000	0.49986	0.00009

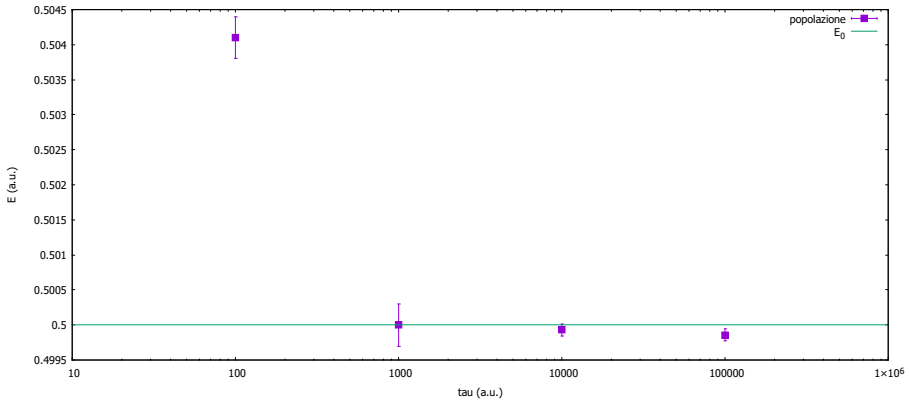

Figura 3.1.6: Andamento di E rispetto alla popolazione target N_w, il time-step scelto è $\tau = 0.1$

Si nota dal grafico (3.1.6) che l'andamento di E inizia a convergere dopo $N_w = 1000$ e si ha una deviazione significativa solo per $N_w = 100$. Per ottenere una stima quantitativa del bias, si può considerare come riferimento il valore a cui convergono gli ultimi punti del grafico. Ad esempio, si può notare che per $N_w = 10^5$ si ha convergenza fino alla quarta cifra significativa dell'energia; prendendo $E = 0.4999$ a.u. come valore di riferimento, il bias per $N_w = 100$ sarà di circa $0,004$ a.u.

Piuttosto che concentrarsi sulla stima numerica, è importante notare che campionando la funzione d'onda con un numero di walker sufficientemente grande ($N_w > 1000$ in questo caso), il bias relativo alla popolazione diventa rapidamente trascurabile rispetto a quello relativo al time-step.

3.2 Atomo di idrogeno

3.2.1 Risultati analitici

Il potenziale dell'atomo di idrogeno assume una forma molto semplice in unità atomiche:

$$V(r) = -\frac{1}{r} \qquad r = \sqrt{x^2 + y^2 + z^2} \tag{3.2.1}$$

La funzione d'onda dello stato fondamentale è

$$\Phi_0(r) = \frac{e^{-r}}{\sqrt{\pi}} \tag{3.2.2}$$

e la relativa energia è $E_0 = -0.5$.

3.2.2 Risultati della simulazione

Sono state effettuate delle simulazioni DMC per l'atomo di idrogeno utilizzando la stessa metodologia illustrata nella sezione precedente.

Andamento dell'energia Si osserva anche per l'atomo di idrogeno un andamento dell'energia simile a quello osservato nel caso dell'oscillatore: di seguito viene riportato quello osservato in una simulazione con parametri $\tau = 0.05$ a.u. $N_{steps} = 1500$ $N_w = 10000$ ed è stato scelto $E_t = 3$ a.u. come valore iniziale.

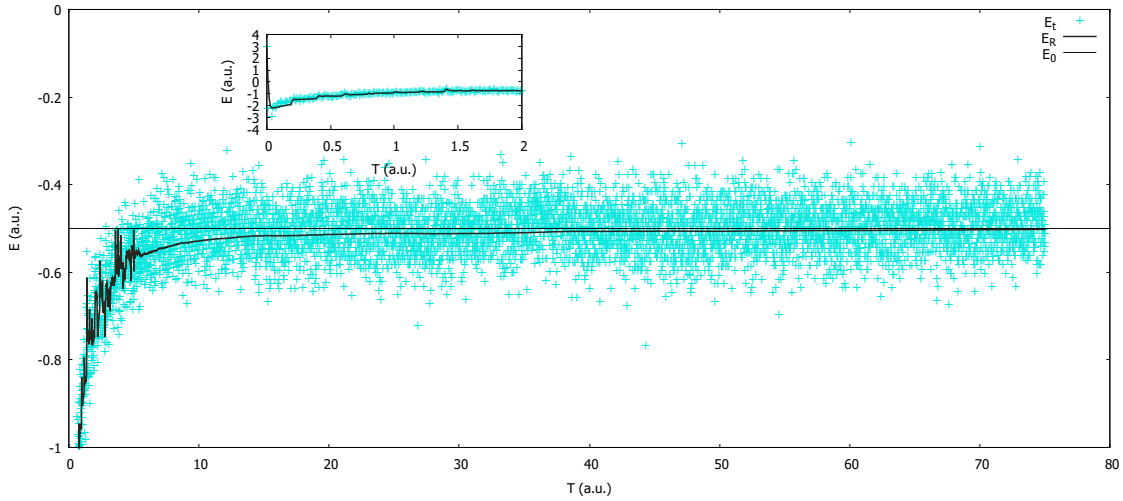

Figura 3.2.1: Andamento di E_R e di $E_t$in funzione di T. Dall'inset si può notare che il valore di $E_t$iniziale è stato scelto fuori scala.

Funzione d'onda Nel caso dell'atomo di idrogeno, si osserva una convergenza più lenta alla distribuzione dello stato fondamentale, a causa della singolarità presente nel potenziale che rende meno efficiente il sampling. All'equilibrio si osserva comunque un buon accordo con il risultato analitico. Di seguito si riporta la distribuzione radiale delle posizioni dei walker all'equilibrio ottenuta con la simulazione precedente, ed il confronto con il risultato atteso $f(r) = 4\pi r^2 \Phi_0$.

Stima dell'energia Si riportano le stime dell'energia ottenute per diversi valori del time-step τ. In questo caso si è mantenuto costante la lunghezza totale T della simulazione, per evidenziare il costo in termini di MCS, che si ha nel diminuire il valore di τ.

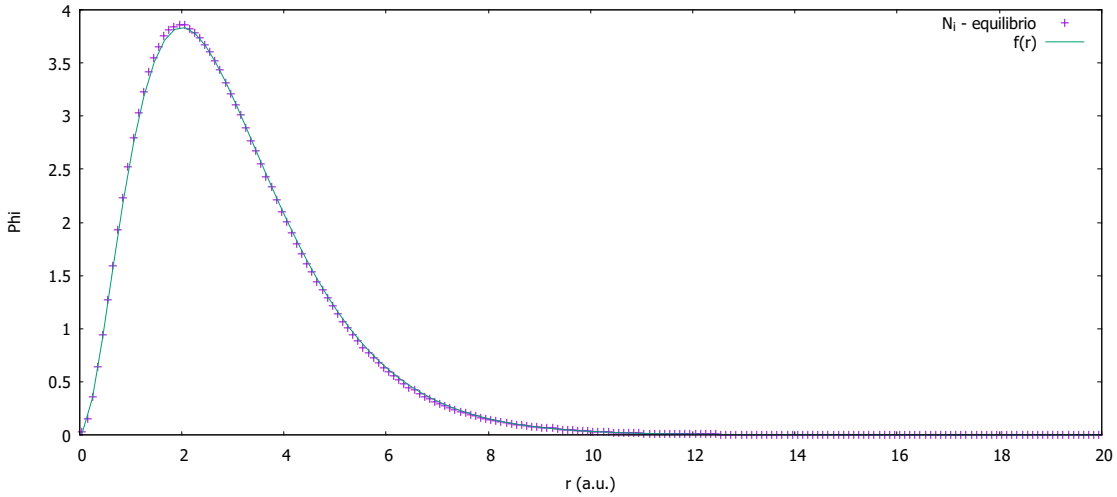

Figura 3.2.2: Distribuzione delle posizioni dei walker all'equilibrio

τ (a.u.)	N_w	T (a.u.)	E_R (a.u.)	ΔE_R (a.u.)
0.01	10000	1000	-0.5011	0.0005
0.005	10000	1000	-0.5003	0.0004
0.002	10000	1000	-0.5008	0.0003

A differenza del caso dell'oscillatore non è possibile usare valori del time-step molto grandi a causa della singolarità nel potenziale: i walker che si avvicinano al centro del potenziale potrebbero iniziare a riprodursi in maniera incontrollata, occupando tutta la memoria allocata inizialmente. Con un time-step più piccolo si ottiene invece una diffusione più «controllata» (la gaussiana associata ad uno step di diffusione ha varianza τ), e si riesce a «mitigare» questo effetto.

Il problema descritto può essere risolto definitivamente introducendo l'importance sampling nella simulazione DMC.

Conclusioni

Un'ampia classe di problemi in Fisica della Materia richiede la modellizzazione di «grandi» sistemi e di complesse strutture elettroniche, per i quali l'approccio computazionale diventa necessario.

Il Quantum Monte Carlo è un metodo stocastico solido e accurato per la risoluzione dell'equazione di Schrodinger a molti elettroni. Sebbene esistano metodi computazionali con un costo meno elevato, esso rimane, nello studio di sistemi con un grande numero di elettroni, il più accurato tra gli approcci disponibili e rappresenta uno strumento fondamentale per lo studio dell'energia di correlazione.

In questo lavoro di tesi, oltre ad illustrare la teoria alla base del Variational e del Diffusion Monte Carlo, è stato sviluppato autonomamente un codice per implementare il DMC nel caso di sistemi monoelettronici. L'algoritmo è stato poi applicato allo studio dell'oscillatore armonico e dell'atomo di idrogeno, due sistemi scelti in quanto vengono considerati dei «toy-model» in Fisica della Materia. I risultati ottenuti sono molto vicini a quelli ottenibili con una risoluzione analitica. Altri sistemi monoelettronici possono essere poi trattati facendo una semplice modifica del potenziale.

Lo stesso algoritmo può essere applicato con successo allo studio di sistemi più complessi, a patto di fare alcune modifiche: le più importanti sono l'*importance sampling* e la $fixed-node\ approximation$. La prima è una tecnica che permette di migliorare il sampling e permette, ad esempio, di gestire in maniera più efficiente le singolarità nei potenziali; la seconda è un'approssimazione che consente di introdurre il vincolo che la funzione d'onda sia antisimmetrica rispetto allo scambio di particelle, necessario per sistemi di più fermioni.

Sono stati esplorati quindi solo i fondamenti del metodo Quantum Monte Carlo, la ricerca di nuove tecniche per rendere le simulazioni QMC più efficienti è ancora attiva, e risulta una delle più promettenti per conoscere in modo accurato le proprietà di grandi sistemi correlati.

Bibliografia

[1] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal, *Quantum Monte Carlo simulation of solids*. Rev. Mod. Phys. 73, 33 (2001).

[2] B.L. Hammond, *Quantum Monte Carlo in ab initio quantum chemistry*. World scientific (1994).

[3] I. Kosztin, B. Faber, K. Schulten, *Introduction to the Diffusion Monte Carlo method*. Am. J. Phys. 64, 633 (1996).

[4] L.M. Barone, E.Marinari, G.Organtini, F. Ricci-Tersenghi, *Programmazione scientifica*. Pearson (2019).

[5] V.S.Vladimirov, *Equation of Mathematical Physics*. M.Dekker incorporated (1971).

[6] C. J. Umrigar, M. P. Nightingale, and K. J. Runge, J. Chem. Phys. 99, 2865 (1993).

3.3 Appendice: listato del programma

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


#define K_el 1
#define PI          3.14159265
#define printall 0
#define debug 1
#define maxNwf 10
#define stepsperblock 20
#define Tequil 5.0  // atomic units


 void GeneraWalkerG(long int D, long int Nw, double **w);
 void GeneraWalkerU(long int D, long int Nw, double **w);
 double V_i(long int D, long int i,long int tipo, double **w);
 void Propagazione(long int D,long int N, double **w, double tau);
 void Branching(long int D, long int *Nwt,long int Nw, double tau, double *A, double *B, double
      Et, double **w);
 double norm(double mean, double std_dev); //genera numeri gaussiani con la box-muller
 double rand_val(int seed);          // funzione simile a rand, genera un double casuale
     compreso fra 0 e 1
 void f_onda(long int Nbin, double Rmax,long int Nwt,long int D, double **w, double *);
 double binning(double *data , long int numMeas); // reblocking



int main(){
     double tau, Et, Ebest, E;
     long int i, j, l, D, Nsteps, Nwt, Nw, tipo, n, distr, NstepsEq, nblock;
     double **w, **wnew;      double *A, *B, alfa, beta;
     double *E_block, delta;
     double *phi, *phi_eq, Rmax=10.0;
     long int Nbin=100;
     long int numMeas, binSize, equilibrio, flag;


     //input      do {
          printf("Potenziale: digita 1 per il potenziale armonico, 2 per l'atomo di idrogeno \n"
              );
          scanf("%ld",&tipo);
     }while((tipo!=1)&&(tipo!=2));

     printf("inserire dimensione\n");
     scanf("%ld",&D);
     do{
          printf("Scegli la distribuzione iniziale dei walker: digita 1 per uniforme, 2 per
              gaussiana \n");
          scanf("%ld", &distr);
     }while((distr!=1) && (distr!=2));

     printf("inserire il time step tau\n");
     scanf("%lf",&tau);
     printf("inserisci il numero di step\n");
     scanf("%ld",&Nsteps);
     printf("inserisci un numero di walker target\n");
     scanf("%ld",&Nw);
     Nwt = Nw;     // initial number of walkers = target
     Et = 0.0;

//     printf("inserisci il valore di Et iniziale \n");
//     scanf("%lf",&Et);

     printf("inserisci flag funzione d'onda (1: si (Rmax=10,Nbin=100); 2: si input Rmax,Nbin;
         else:altro) \n");
     scanf("%ld",&flag);
     if(flag==2){
          printf("inserisci distanza massima Rmax \n");
         scanf("%lf",&Rmax);
          printf("Inserisci numero di bin Nbin \n");
         scanf("%ld",&Nbin);
          flag = 1;
     }

     // w: matrice con le posizioni dei walker
     w = (double**) malloc (maxNwf*Nw*sizeof(double*));

            for (i=0; i<Nw; i++) {
                     w[i]=(double*) malloc (D*sizeof(double));
            }
     printf("memoria allocata \n");

     // inizializzazione funzioni random
     rand_val(time(NULL));
     srand(time(NULL));

     if(distr==1){
```

```
85            GeneraWalkerU(D,Nw,w);
86        }
87        else if(distr==2){
88            GeneraWalkerG(D,Nw,w);
89        }
90        else{
91            printf("Distribuzione iniziale non definita correttamente \n");
92            exit(EXIT_FAILURE);
93        }
94
95        printf("walker generati \n");
96
97        Ebest = 0.0;
98        E=0.0;
99
100       //file di output
101       FILE *fe;
102       FILE *fb;
103       FILE *fp;
104       FILE *ff;
105
106       fe=fopen("energie.txt","w");
107       fprintf(fe,"# tau  \t \t Et \t \t Ebest \n");
108
109       fp=fopen("popolazione.txt","w");
110       fprintf(fp, "T \t Nwt \n");
111       fb=fopen("reblocking.txt","w");
112       fprintf(fb, "binSize \t \t varianza \t \t numMeas \n");
113
114       if (flag==1) {
115           ff=fopen("f_onda.txt","w");
116           fprintf(ff,"R \t \t Phi \n");
117           phi = (double*) malloc (Nbin*sizeof(double));
118           phi_eq = (double*) malloc (Nbin*sizeof(double));
119       }
120
121       NstepsEq = (long)( Tequil / tau );
122       numMeas = Nsteps - NstepsEq;
123
124       if (numMeas>0){
125           E_block = (double*) malloc( numMeas * sizeof(double) );
126       } else {
127           E_block = (double*) malloc(0);
128           numMeas = 0.0;
129       }
130
131       //inizio ciclo
132       for( n=0; n<Nsteps; n++ ){
133
134               // A: potenziali prima della diffusione
135               // B: potenziali dopo la diffusione
136
137               printf("n walker %ld\n", Nwt);
138               A = (double*) malloc (Nwt*sizeof(double));
139               for(i=0;i<Nwt;i++){
140                   if(*(w+i)!=NULL){
141                       A[i]=V_i(D, i, tipo, w);
142                   }
143               }
144
145               if(flag==1){
146                   delta=Rmax/(double)Nbin;
147                   if(n<Tequil/tau && ((n % stepsperblock)==0)){
148                       fprintf(ff,"# funzione d'onda al passo numero %ld \n",n);
149                       fprintf(ff, "\n");
150                       f_onda(Nbin,Rmax,Nwt,D,w,phi);
151
152                       if(D>1){
153                           for(i=0;i<Nbin;i++){
154                               fprintf(ff,"%f \t %f \n",delta*(i+0.5), phi[i]);
155                           }
156                           fprintf(ff, "\n");
157                       }
158                       else{
159                           for(i=0;i<Nbin;i++){
160                               fprintf(ff,"%f \t %f \n", delta*(i+0.5-(Nbin/2)), phi[i]);
161                           }
162                           fprintf(ff, "\n");
163                       }
164                   }
165               }
166
167               fprintf(fp,"%ld\t%f\t%ld\n",n,n*tau, Nwt);
168
169
170               Propagazione(D,Nwt,w,tau);          // Step diffusion
171
172               B=(double*) malloc (Nwt*sizeof(double));
173               for(i=0;i<Nwt;i++){
174                   if(*(w+i)!=NULL){
175                       B[i]=V_i(D, i, tipo, w);
176                   }
177               }
```

```
178               printf(">>> step propagazione n. %ld \n",n);
179
180
181               Branching(D, &Nwt, Nw, tau, A,B, Et,w);        //step di branching
182               printf("step branching n. %ld eseguito \n",n);
183
184              printf("Nwt adesso e' %ld \n", Nwt);
185               free(A);
186              free(B);
187
188               // definire nuova energia Et
189              if ( n<NstepsEq ){
190                   nblock = n % stepsperblock;
191                   Ebest = ( Et + Ebest * nblock ) / (1+nblock);
192               }else {
193                   nblock = n - NstepsEq;
194                   Ebest = ( Et + Ebest * nblock ) / (1+nblock);
195               }
196
197               fprintf(fe,"%ld\t%f\t%f\t%f\n", n, n*tau, Et, Ebest);              // Aggiornare Et
198               Et = Ebest-(1.0/tau)*log((double) Nwt / (double) Nw);        //ricalcolo di Et all'
                      equilibrio
199
200               // riempio array con valore Et dopo equilibrazione
201               if(n>NstepsEq){
202                   E_block[n-NstepsEq]=Et;
203               }
204               printf("Et= %f \t Ebest=%f \n", Et,Ebest);
205
206               if(flag==1){
207                   if(n>NstepsEq){
208                       f_onda(Nbin,Rmax,Nwt,D,w,phi);
209                       for(i=0;i<Nbin;i++){
210                           phi_eq[i]+=phi[i];
211                       }
212                   }
213               }
214      }
215  // fine ciclo DMC
216      free( w );
217
218
219      if(flag==1){
220          // calcolo funzione d'onda all'equilibrio
221          for(i=0;i<Nbin;i++){
222              phi_eq[i]=phi_eq[i]/(Nsteps-NstepsEq);
223          }
224          fprintf(ff, "#funzione d'onda all'equilibrio \n" );
225          fprintf(ff, "\n");
226
227          if(D>1){
228              for(i=0;i<Nbin;i++){
229                  fprintf(ff,"%f \t %f \n", delta*(i+0.5), phi_eq[i]);
230              }
231              fprintf(ff, "\n");
232          }
233          else{
234              for(i=0;i<Nbin;i++){
235                  fprintf(ff,"%f \t %f \n", delta*(i+0.5-(Nbin/2)), phi_eq[i]);
236              }
237              fprintf(ff, "\n");
238          }
239
240         fclose(ff);
241         free(phi);
242         free(phi_eq);
243      }
244      fclose(fe);
245      fclose(fp);
246
247      //reblocking
248
249     binSize = 1;
250
251     while (numMeas >= 100) {
252          fprintf(fb,"%ld \t %lf \t %ld \n",binSize , binning(E_block, numMeas), numMeas);
253          binSize *= 2;
254          numMeas /= 2;
255      }
256      fclose(fb);
257      free(E_block);
258      printf("programma eseguito \n");
259      return 0;
260  }
261
262
263
264
265  void GeneraWalkerG(long int D, long int Nw, double **w){
266      long int i, j;
267      for(i=0;i<Nw;i++){
268          for(j=0;j<D;j++){
269              w[i][j] = norm(0,1);
```

```
270             }
271         }
272     }
273
274     void GeneraWalkerU(long int D, long int Nw, double **w){
275         long int i, j;
276         for(i=0;i<Nw;i++){
277             for(j=0;j<D;j++){
278                 w[i][j]=rand()/(double)(RAND_MAX) -0.5;
279             }
280         }
281     }
282
283
284     double V_i(long int D, long int i,long int tipo, double **w){
285         double a=0;
286         long int j;
287         if(tipo == 1){
288             a=0;
289             for(j=0;j<D;j++){
290                 a+=(0.5)*K_el*(pow((w[i][j]),2));
291             }
292         }
293         else if(tipo == 2){
294             a=0;
295             for(j=0;j<D;j++){
296                 a+=(pow((w[i][j]),2));
297             }
298             a=-1.0/sqrt(a);
299         }
300         else{
301             printf("potenziale non definito correttamente \n");
302             exit(EXIT_FAILURE);
303         }
304
305         return a;
306     }
307
308
309     void Propagazione(long int D,long int Nwt, double **w, double tau){
310         long int i,j;
311         for(i=0;i<Nwt;i++){
312                 for(j=0;j<D;j++){
313                     w[i][j]+= sqrt(tau) *norm(0,1);
314                 }
315         }
316     }
317
318
319     void Branching(long int D, long int *pNwt, long int Nw, double tau, double *A, double *B,
          double Et, double **w){
320         long int i,j, k, Mi, Nnew=0.0, inew;      double wi;      double **wnew;      long int *M;
321         M = (long int *) malloc( *pNwt * sizeof(long int) );
322         for( i=0 ; i < *pNwt; i++ ){
323             wi = exp(-( tau * ( A[i] + B[i] - 2*Et )/2) );
324             Mi = (long int)(wi+(rand()/(double)RAND_MAX));
325            if (printall) printf("Mi= %ld \n",Mi);
326             M[i] = Mi;
327             Nnew += Mi;
328         }
329
330        printf("Nwt=%ld\tNnew=%ld\n", (*pNwt), Nnew );
331
332         if ( Nnew <= maxNwf * (*pNwt) ) {
333             wnew = (double**) malloc( Nnew * sizeof(double*) );
334         } else {
335             printf("errore, programma ha utilizzato tutta la memoria allocata, aumentare maxNwf \n
                    ");
336             exit(EXIT_FAILURE);
337         }
338
339         inew = 0;
340         for( i=0; i < *pNwt; i++ ){
341             if ( M[i]==0 ) {
342                 // do not replicate walkers
343                 free( w[i] );
344             }
345             else {
346                 wnew[inew] = w[i];
347                 inew++;
348                 for ( j=1; j<M[i]; j++ ) {
349                     wnew[inew] = (double*) malloc( D*sizeof(double) );
350                     for (k=0;k<D;k++)
351                         wnew[inew][k] = w[i][k];
352                     inew++;
353                 }
354             }
355         }
356
357         if (inew != Nnew){
358             printf("errore: inew dopo ciclo != Nnew");
359             exit(EXIT_FAILURE);
360         }
```

```
361
362       *pNwt = Nnew;
363       if(*pNwt == 0){
364            printf("errore, tutti i walker sono morti \n");
365            exit(EXIT_FAILURE);
366       }
367
368       // libero memoria usata da wnew
369      for( i=0; i<Nnew; i++ ){
370            w[i] = wnew[i];
371       }
372       free(wnew);
373       free(M);
374  }
375
376
377  double norm(double mean, double std_dev) {
378    double   u, r, theta;           // Variables for Box-Muller method
379    double   x;                     // Normal(0, 1)rv
380    double   norm_rv;                 // The adjusted normal rv
381
382    // Generate u
383    u = 0.0;
384    while (u == 0.0)
385      u = rand_val(0);
386
387    // Compute r
388    r = sqrt(-2.0 * log(u));
389
390    // Generate theta
391    theta = 0.0;
392    while (theta == 0.0)
393      theta = 2.0 * PI * rand_val(0);
394
395    // Generate x value
396    x = r * cos(theta);
397
398    // Adjust x value for specified mean and variance
399    norm_rv = (x * std_dev) + mean;
400    // Return the normally distributed RV value
401    return(norm_rv);
402  }
403
404
405  double rand_val(int seed) {
406    const long  a =       16807; // Multiplier
407    const long  m = 2147483647;  // Modulus
408    const long  q =      127773;  // m div a
409    const long  r =        2836;  // m mod a
410    static long x;               // Random int value
411    long        x_div_q;         // x divided by q
412    long        x_mod_q;         // x modulo q
413    long        x_new;           // New x value
414
415    // Set the seed if argument is non-zero and then return zero
416
417   if (seed > 0)    {
418      x = seed;
419      return(0.0);
420   }
421    // RNG using integer arithmetic
422    x_div_q = x / q;
423    x_mod_q = x % q;
424    x_new = (a * x_mod_q) - (r * x_div_q);
425    if (x_new > 0)
426      x = x_new;
427    else
428      x = x_new + m;
429
430    // Return a random value between 0.0 and 1.0
431    return((double) x / m);
432  }
433
434
435  void f_onda(long int Nbin, double Rmax,long int Nwt,long int D, double **w, double *histo){
436       double deltaR,R_i=0.0,ausR=0.0, normalizzazione=0.0;
437       long int i, j, bin;
438
439       for(i=0;i<Nbin;i++){
440            histo[i]=0;
441       }
442
443       if(D==1){
444            for(i=0;i<Nwt;i++){
445                 bin=(long int)(w[i][0]*Nbin/Rmax+(Nbin/2));
446                 histo[bin]++;
447            }
448       }
449       else{
450            for(i=0;i<Nwt;i++){
451                 for(j=0;j<D;j++){
452                      ausR+=pow(w[i][j],2);
453                 }
```

```
454                 R_i=sqrt(ausR);
455                 ausR=0.0;
456
457                 bin=(long int)(R_i*Nbin/Rmax);
458                 if(bin<Nbin){
459                     histo[bin]++;
460                 } else {
461                     printf("aumenta Rmax");
462                     //exit(EXIT_FAILURE);
463                 }
464             }
465         }
466
467         if(D==1){
468             for(i=0;i<Nbin;i++){
469                 normalizzazione+=pow(histo[i],2);
470             }
471             normalizzazione=sqrt(normalizzazione*Rmax/Nbin) ;
472             for(i=0;i<Nbin;i++){
473                 histo[i]=histo[i]/normalizzazione;
474             }
475         }
476         if(D==3){
477             normalizzazione=Nwt*(Rmax/Nbin)/(8*sqrt(PI));
478             for(i=0;i<Nbin;i++){
479                 histo[i]=histo[i]/normalizzazione;
480             }
481         }
482     }
483
484
485     double binning(double *data , long int numMeas){
486     long int i, tmp = numMeas / 2; double mean = 0.0, variance = 0.0;
487
488     for (i = 0; i < tmp; i++) {
489         mean += data[2 * i] + data[2 * i + 1];
490         variance += data[2 * i] * data[2 * i] +     data[2 * i + 1] * data[2 * i + 1];
491         data[i] = 0.5 * (data[2 * i] + data[2 * i + 1]);
492     } if (2 * i < numMeas) {
493         mean += data[2 * i];
494         variance += data[2 * i] * data[2 * i]; } mean /= numMeas;
495     variance /= numMeas;
496
497
498     return sqrt((variance - mean * mean ) / (numMeas - 1));
499     }
```