Variational Montecarlo

Ricerca dello stato Fondamentale per He

Anthea Boiani, Alex Martinelli

L'obiettivo è la scrittura di un codice in Python che permetta di determinare l'energia dello stato fondamentale dell'atomo di He attraverso l'utilizzo del metodo variational MonteCarlo

Introduzione

Il metodo Variational MonteCarlo si basa sull'approccio variazionale per l'approssimazione dello stato fondamentale del sistema. In particolare, il principio variazionale viene applicato all'equazione di Schrödinger per l'Hamiltoniano $\hat{H}(\vec{r})$, con $\vec{r}$ il verrore posizione di tutte le particelle, che ci permette di definire lo stato fondamentale del sistema come funzione ψ che minimizza il funzionale dell'energia $E[\psi] = \frac{<\psi|\hat{H}|\psi>}{<\psi|\psi>}$

Partendo da una famiglia di funzioni di prova $\phi_T(\vec{r}, \alpha, \beta, \dots)$ il problema si riduce a trovare il valore appartenente al set dei parametri variazionale {$\alpha, \beta, \dots$} che minimizza il valore di aspettazione dell'energia $\bar{H}(\alpha, \beta, \dots)$ = $E(\alpha, \beta, \dots)$:

$$\bar{H}(\alpha, \beta, \dots) = \frac{\int d\overrightarrow{r}\ \phi_T^*(\overrightarrow{r}; \alpha, \beta, \dots)\widehat{H}(\overrightarrow{r})\,\phi_T(\overrightarrow{r}; \alpha, \beta, \dots)}{\int d\overrightarrow{r}\ |\phi_T(\overrightarrow{r}; \alpha, \beta, \dots)|^2}$$

$$= \int d\overrightarrow{r}\, w(\overrightarrow{r}; \alpha, \beta, \dots) E_L(\overrightarrow{r}; \alpha, \beta, \dots) = \langle E_L \rangle_{(\alpha,\beta,\dots)}$$

dove $E_L(\vec{r}, \alpha, \beta, \dots)$ utilizzata è la funzione Energia Locale

$$E_L(\vec{r}, \alpha, \beta) = \frac{\widehat{\mathbf{H}}(\overrightarrow{r})\,\phi_T(\overrightarrow{r}; \alpha, \beta, \dots)}{\phi_T(\overrightarrow{r}; \alpha, \beta, \dots)}$$

e $\omega(\vec{(}r), \alpha, \beta, \dots)$ è la funzione peso così definita:

$$w(\overrightarrow{r}; \alpha, \beta, \dots) = \frac{|\phi_T(\overrightarrow{r}; \alpha, \beta, \dots)|^2}{\int d\overrightarrow{r}\,|\phi_T(\overrightarrow{r}; \alpha, \beta, \dots)|^2};$$

che soddisfa le condizioni di positività e di normalizzazione che definiscono la funzione densità di probabilità nello spazio {$\vec{r}$}:

$w(\overrightarrow{r}, \alpha, \beta, \dots) \geq 0\,,\ \forall \overrightarrow{r}\ e\ \int d\overrightarrow{r}\ w(\overrightarrow{r}, \alpha, \beta, \dots) = \int d\overrightarrow{r}\ \frac{|\phi_T(\overrightarrow{r}, \alpha, \beta, \dots)|^2}{\int d\overrightarrow{r}\ |\phi_T(\overrightarrow{r}, \alpha, \beta, \dots)|^2} = 1$

Conseguentemente, il problema si riduce alla ricerca del minimo del valore medio della

funzione Energia Locale che è definito dall'integrale $\bar{H}(\alpha, \beta, \dots) = \langle E_L \rangle_{(\alpha,\beta,\dots)}$. Questo integral epuò essere calcolato come media sulla sequenza $\{\vec{r}_j, j = 1, 2, \dots, M\}$ di campionamento dello spazio generata attraverso il metodo Metropolis MonteCarlo secondo la densità di probabilità $\vec{r}, \alpha, \beta, \dots)$ per ogni valore dei parametri variazionali $\{\alpha, \beta, \dots\}$

Metodo MonteCarlo Metropolis

Il metodo Metropolis MonteCarlo consiste nella generazione di una catena di Markov, quindi di una sequenza di valori casuali $(x_1, x_2, \dots, x_n, \dots)$ campionati secodno una certa probabilità arbitraria $\rho(x)$. Per passare da un termine x_i all'altro della catena viene generato un valore x_m di prova che può essere scartato o diventare un nuovo termine della catena. Qundi, se $\rho(x_{i+1}) \leq \rho(x_m)$ il valore generato lo accettiamo, altrimenti lo scartiamo.

In altre parole, se definiamo la probabilità di transizione come $W_{k,k+1} = min\{1, \omega\}$, con $\omega = \frac{p(x_m)}{p(x_k)}$ e preso ν un valore casuale estratto con probabilità uniforme nell'intervallo $[0, 1)$ si ha che se $\omega \geq 1 \; per \; x_{k+1} = x_T$ e per

$$\omega < 1 \begin{cases} \omega \geq \nu & x_{i+1} = x_m \\ \omega < \nu & x_i = x_m \end{cases}$$

Per garantire l'ergodicità della catena vediamo che $x_m = x_i + \eta$ con η compreso tra $[-\delta, \delta]$ dove δ è un parametro da specificare e η campionata con una probabilità di $\rho = \frac{1}{2\delta}$

Sistema del problema

Il sistema in analisi è l'atomo di He, il cui numero atomico è Z = 2. Lavoriamo in unità atomiche, $\hbar = e = m_e = 4\pi\epsilon_0 = 1$, immaginando il nucleo ($m_n \sim 8000 m_e$) fermo nell'origine del sistema di coordinate. Trascurando i termini di spin, l'Hamiltoniana per un atomo di elio si può scrivere come

$$\widehat{\mathbf{H}} = -\frac{1}{2}\nabla_1^2 - \frac{1}{2}\nabla_2^2 - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}$$

dove $\vec{r_1}$ e $\vec{r_2}$ sono le coordinate dei due elettroni e r_{12} = $|\vec{r_{12}}|$ = $|\vec{r_1} - \vec{r_2}|$ quindi pari alla distanza relativa.

Nel caso in cui la funzione d'onda di spin degli elettroni è antisimmetrica (quindi che corrisponde allo stato di singoletto) la funzione d'onda orbitale dei due elettroni è scrivibile come prodotto simmetrizzato delle funzioni di singola particella. Nel caso manchi il termine repulsivo il problema è equivalente a quello di due elettroni non-interagenti

Task A. Atomo idrogenoide con elettroni non-interagenti

Nel caso di atomi non-interagenti, la funzione d'onda dello stato fondamentale è proporzionale a $e^{-Zr_1}e^{-Zr_2}$. Utilizzando il principio variazionale possiamo calcolare analiticamente e minimizzare $\langle E_L \rangle_Z$ troviamo Z = 2 e autovalore E_0 = 4.

Nel momento in cui aggiungiamo il termine repulsivo $\frac{1}{r_{12}}$ otteniamo un valore minimo di Z

= 27/16, $\langle E_L \rangle_Z = Z^2 - \frac{27}{8}Z = -2.8477$ che rispetto al valore sperimentale dello stato fondamentale dell'elio non è molto lontano E_0 = -2.904.

Scopo della Task è verificare i risultati applicando il metodo VMC utilizzando la funzione d'onda di prova $\psi_T(r_1, r_2; Z)e^{-Zr_1}e^{-Zr_2}$

Task B. Funzione d'onda di prova di Padé-Jastrow

In questa seconda task si richiede di utilizzare come funziona di prova la funzione d'onda di Padé-Jastrow ottenuta falla funzione d'onda precedente miltiplicato per il fattore esponenziale di correlazione di Jastrow con all'esponente l'approssimante di Padé $p(x) = \frac{x}{1+\beta x}$

$$\psi_T(r_1, r_2; Z, \alpha, \beta)e^{-Zr_1}e^{-Zr_2}e^{\frac{\alpha r_{12}}{1+\beta r_{12}}}$$

con α e β sono parametri variazionali.

E' possibile ricavarsi l'espressione dell'energia locale come

$$E_L(\overrightarrow{r_1}, \overrightarrow{r_2}; Z, \alpha, \beta) = -Z^2 + \frac{(Z-2)}{r_1} + \frac{(Z-2)}{r_2} + \frac{1}{r_{12}}[1 - \frac{2\alpha}{(1+\beta r_{12})^2}] + \frac{2\beta\alpha}{(1+\beta r_{12})^3}$$

Imponendo Z = 2 e α = 0.5, utilizzando il metodo VMC trovare il valore d'aspettazione $\langle E_L \rangle_{\beta(Z=2,\alpha=\frac{1}{2})}$ utilizzando un'ottimizzazione per selezionare il β per cui l'energia l'energia è minima. Infine, confrontare i valori ottenuti con quelli ottenuti nel task precedente.

Il codice

Task A.

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
class VMC_He:
    #Definisco il sistema di coordinate: un sistema a due elettroni
    #che si muovono lungo le tre coordinate x, y, z
    def __init__(self):
        self.coords = np.zeros((2,3))
    #Questo comando mi permette di generare numeri random
        self.rg = np.random.default_rng()

    def SetParams(self,params):
        self.params=params.copy()
        #Permette la scelta tra i potenziali, se utilizzare quello interagente o no
        self.pot = self.params[1]

    def SetCoords(self,coords):
        self.coords=coords.copy()

        #Definisco la funzione d'onda di prova
```

```
    def WaveFunction(self,coords):
        alpha=self.params[0]
        r1 = np.linalg.norm(coords[0,:])
        r2 = np.linalg.norm(coords[1,:])
        return np.exp(-alpha*(r1+r2))

    def LocalEnergy(self,coords):
        KE = -0.5*self.LaplacianPsiOverPsi(coords)
        V  = self.Potential(coords)
        return V+KE

    def Potential(self,coords):
        r1 = np.linalg.norm(coords[0,:])
        r2 = np.linalg.norm(coords[1,:])
        #Scelta del potenziale da utilizzare per il calcolo dell'energia locale
        if self.pot:
            #sistema interagente
            r12 = np.linalg.norm(coords[0,:] - coords[1,:])
            potential = (-2*r12*(r1 + r2) + r1*r2)/(r1*r2*r12)
        else:
            #sistema non interagente
            potential = (-2*r2-2*r1)/(r1*r2)
        return potential

    def LaplacianPsiOverPsi(self,coords,delta=0.0001):
        total=0.0
        tempVal3=self.WaveFunction(coords)
        for i in range(0,len(coords)):
            for j in range(0,len(coords[0])):
                coords[i,j]=coords[i,j]+delta
                tempVal=self.WaveFunction(coords)
                coords[i,j]=coords[i,j]-2*delta
                tempVal2=self.WaveFunction(coords)
                coords[i,j]=coords[i,j]+delta
                total +=(tempVal+tempVal2)-2.0*tempVal3
        return total/(delta*delta*tempVal3)

    #Definisco il Variational MonteCarlo inizializzando le coordinate e le energie
    def VMC(self,numSteps=1000,delta=1.618034):
        EnergyList=np.zeros(numSteps)
        CoordsList_r1=np.zeros((numSteps,3))
        CoordsList_r2=np.zeros((numSteps,3))
        #Inizializzo gli steps:
        movesAttempted=0.0
        movesAccepted=0.0
        #Inizio a costruire la funzione d'onda e l'energia locale
        Psi=self.WaveFunction(self.coords)
        energy=self.LocalEnergy(self.coords)
        #Seleziono il vettore corrispondente ad un elettrone e
        #genera un numero casuale tra 0 e 1 poiché il sistema è composta da due ele
        #successivamente mi genera un numero tra 0 e 2 per le coordinate (x, y, z)
        for step in range(numSteps):
            nu1 = np.random.randint(2)
            nu2 = np.random.randint(3)
            #seleziono nuove coordinate e mi costruisco una nuova funzione d'onda
            #con le nuove coordinate
            RT = self.coords.copy()
            RT[nu1,nu2] = self.coords[nu1,nu2] + self.rg.uniform(-delta,delta,1)
            newPsi=self.WaveFunction(RT)
            #Applicazione dell'algoritmo Metropolis per determinare se la
            #nuova coordinata dell'elettrone va mantenuta o scartata
            if ( newPsi**2/Psi**2 > self.rg.random() ):
                self.coords=RT.copy()
                Psi=newPsi
```

```
                movesAccepted+=1.
                #Dopo aver accettato la nuova coordinata vado a calcolare
                #la nuova energia locale e salvo le nuove coordinate
                energy=self.LocalEnergy(self.coords)
            movesAttempted+=1.
            EnergyList[step] = energy
            CoordsList_r1[step,:] = self.coords[0,:]
            CoordsList_r2[step,:] = self.coords[1,:]
        return EnergyList,CoordsList_r1,CoordsList_r2,movesAccepted/movesAttempted
```

Prima di andare a verificare le energie, andiamo a determinare quale sia il δ migliore sulla quale fare i calcoli

Analisi errore al variare del rapporto tra mosse accettate e mosse totali

Per quest'analisi utilizzeremo la classe della prima task e andremo a variare il valore del parametro variazionale (chiamato nella classe e nel codice delta) concludendo con un confronto sull'errore del rapporto tra le mosse accettate e totali nel caso degli elettroni interagenti.

In [6]:
```
#CASO ELETTRONI INTERAGENTI
He = VMC_He()
zeta = 27/16
He.SetParams([zeta,1])
N = 10
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()
error = open ('./Analisi_Errore/delta_0.5.txt','wb')
for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 0.5)
    np.save(error, EnergyList[::passo])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
error.close()
print('Acceptance Rate= %6.2f , varianza= %10.6f' % (acceptance_rate,var))
error = np.load("./Analisi_Errore/delta_0.5.txt")
print("Energia media = %5.4f"%(error.mean()))
```

```
Acceptance Rate=  79.98 , varianza=   0.046526
Energia media = -2.8556
```

In [7]:
```
He = VMC_He()
zeta = 27/16
He.SetParams([zeta,1])
N = 10
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()
error = open ('./Analisi_Errore/delta_1.txt','wb')
for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
```

```
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 1)
    np.save(error, EnergyList[::passo])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
error.close()
print('Acceptance Rate= %6.2f , varianza= %10.6f' % (acceptance_rate,var))
error = np.load("./Analisi_Errore/delta_1.txt")
print("Energia media = %5.4f"%(error.mean()))
```

```
Acceptance Rate=  63.48 , varianza=   0.016244
Energia media = -2.8449
```

In [9]:
```
He = VMC_He()
zeta = 27/16
He.SetParams([zeta,1])
N = 10
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()
error = open ('./Analisi_Errore/delta_1.5.txt','wb')
for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 1.5)
    np.save(error, EnergyList[::passo])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
error.close()
print('Acceptance Rate= %6.2f , varianza= %10.6f' % (acceptance_rate,var))
error = np.load("./Analisi_Errore/delta_1.5.txt")
print("Energia media = %5.4f"%(error.mean()))
```

```
Acceptance Rate=  50.73 , varianza=   0.023098
Energia media = -2.8482
```

In [10]:
```
He = VMC_He()
zeta = 27/16
He.SetParams([zeta,1])
N = 10
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()
error = open ('./Analisi_Errore/delta_2.txt','wb')
for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 2)
    np.save(error, EnergyList[::passo])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
error.close()
print('Acceptance Rate= %6.2f , varianza= %10.6f' % (acceptance_rate,var))
```

```
error = np.load("./Analisi_Errore/delta_2.txt")
print("Energia media = %5.4f"%(error.mean()))
```

```
Acceptance Rate=  41.32 , varianza=   0.030851
Energia media = -2.8427
```

In [11]:
```
He = VMC_He()
zeta = 27/16
He.SetParams([zeta,1])
N = 10
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()
error = open ('./Analisi_Errore/delta_2.5.txt','wb')
for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 2.5)
    np.save(error, EnergyList[::passo])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
error.close()
print('Acceptance Rate= %6.2f , varianza= %10.6f' % (acceptance_rate,var))
error = np.load("./Analisi_Errore/delta_2.5.txt")
print("Energia media = %5.4f"%(error.mean()))
```

```
Acceptance Rate=  34.42 , varianza=   0.026966
Energia media = -2.8508
```

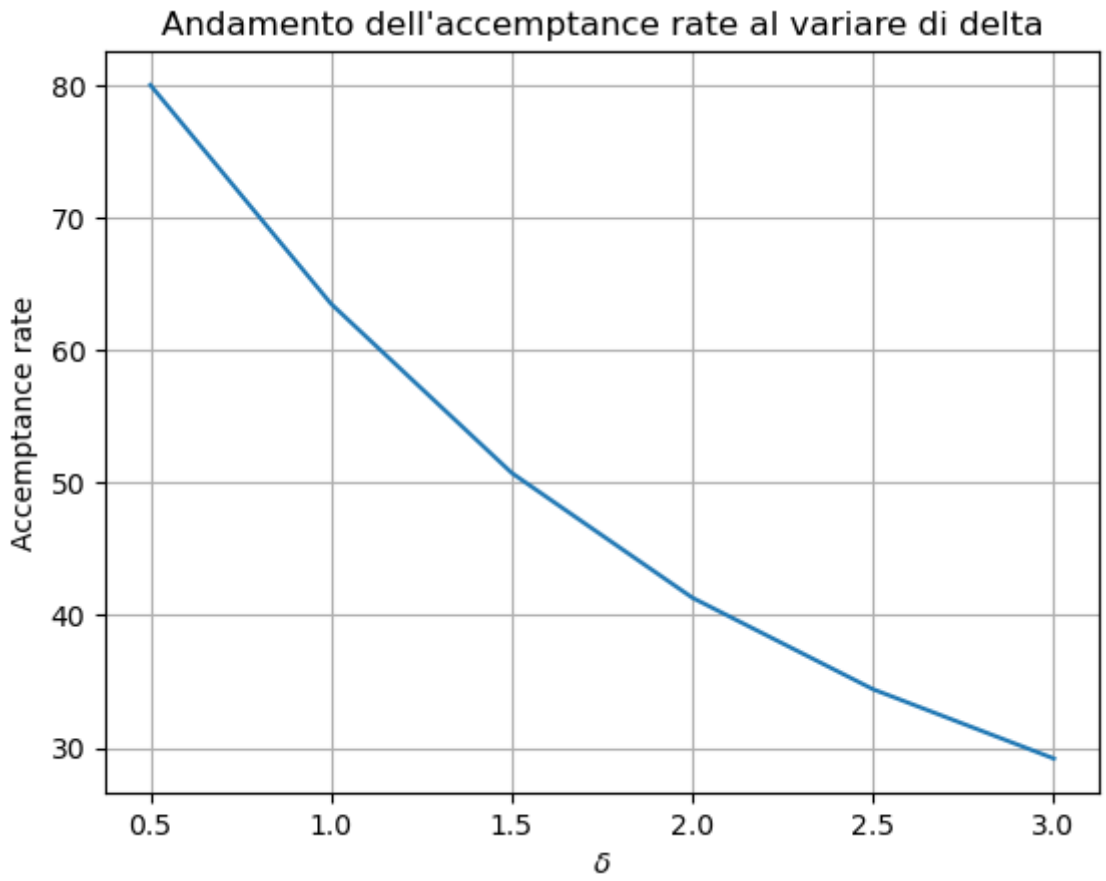
In [12]:
```
He = VMC_He()
zeta = 27/16
He.SetParams([zeta,1])
N = 10
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()
error = open ('./Analisi_Errore/delta_3.txt','wb')
for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 3)
    np.save(error, EnergyList[::passo])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
error.close()
print('Acceptance Rate= %6.2f , varianza= %10.6f' % (acceptance_rate,var))
error = np.load("./Analisi_Errore/delta_3.txt")
print("Energia media = %5.4f"%(error.mean()))
```

```
Acceptance Rate=  29.20 , varianza=   0.073784
Energia media = -2.8322
```

In [15]:
```
delta = [0.5, 1, 1.5, 2, 2.5, 3]
accpt = [79.98, 63.48, 50.73, 41.32, 34.42, 29.20]
plt.plot(delta, accpt)
plt.grid()
plt.xlabel("$\delta$")
```

```
plt.ylabel("Accemptance rate")
plt.title("Andamento dell'accemptance rate al variare di delta")
plt.show()
```



E' possibile notare come, con l'aumentare del delta e della percentuale delle mosse accettate, l'accemptance rate tende a diminuire all'aumentare del delta ed è più evidente per $\delta = 2.5$ e $\delta = 3$ mentre per $\delta = 0.5$ è possibile vedere come l'acceptance rate sia maggiore e ciò deriva dalla somiglianza dei moduli quadri in quelle posizioni perché lo spazio esplorato dagli elettroni risulta minimo. Per δ intorno al valore 1, quindi $\delta = 1$ e $\delta = 1.5$ abbiamo una percentuale di mosse accettate compreso tra il 50% e il 60% consentendoci di usare un valore di δ vicino a 1.5.

Caso Elettroni non interagenti

In [13]:
```
#CASO ELETTRONI NON INTERAGENTI
He = VMC_He()
#Impongo il parametro Z come indicato nella task e
#seleziono il potenziale adeguato per il mio sistema (in
#questo caso seleziono il potenziale per sistema non interagente)
#successivamente genero numeri casuali e inizializzo le coordinate
#in un intervallo [-50, 50]
zeta = 2
He.SetParams([zeta, 0])
N = 100
passo = 1
rg = np.random.default_rng()
R = np.zeros((2,3),float)
rate = np.zeros(N)

#Creo dei file .txt dove salvare le coordinate dei due elettroni
#e le energie locali trovate e calcolate
local_energy = open ('./TaskA/A_non_inter/local_energy.txt','wb')
```

```
r1 = open ('./TaskA/A_non_inter/pos_r1.txt','wb')
r2 = open ('./TaskA/A_non_inter/pos_r2.txt','wb')

#Generazione delle coordinate all'interno dell'intervallo
#che abbiamo imposto e mi salvo i risultati ottenuti sui file appena aperti
for i in range(N):
    R[0,:] = np.array(rg.uniform(-50,50,3))
    R[1,:] = np.array(rg.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 1.618034)

    np.save(local_energy, EnergyList[::passo])
    np.save(r1, CoordsList_r1[::passo,:])
    np.save(r2, CoordsList_r2[::passo,:])
    rate[i] = accpt*100
#Mi calcolo l'accemptance rate e la varianza per poi concludere chiudendo
#i file sugli elettroni e sull'energia locale
acceptance_rate = np.mean(rate)
var = np.var(rate)
local_energy.close()
r1.close()
r2.close()
print('Z= %7.4f , Acceptance Rate= %6.2f  , Varianza= %10.6f' % (zeta,acceptance_r
```

```
Z=  2.0000 , Acceptance Rate=  42.73  , Varianza=   0.041881
```

In [16]:
```
#Carico i dai dati e leggo i risultati: calcolo l'energia e lo
#confronto col valore analitico
energy = np.load("./TaskA/A_non_inter/local_energy.txt")
r1 = np.load("./TaskA/A_non_inter/pos_r1.txt")
r2 = np.load("./TaskA/A_non_inter/pos_r2.txt")
#std = np.std(energy)
print("Energia media = %5.4f"%(energy.mean()))
print("Valore di riferimento analitico = -4.00")
#print("Deviazione standard = %3.2f"%(std))
```

```
Energia media = -4.0000
Valore di riferimento analitico = -4.00
```

Caso Elettroni interagenti

In [14]:
```
#CASO ELETTRONI INTERAGENTI
He = VMC_He()
#Impongo le condizini, come indicato nella task, su Z e
#seleziono il potenziale adeguato per il mio sistema (in
#questo caso seleziono il potenziale per sistema interagente)
#successivamente genero numeri casuali e inizializzo le coordinate
#in un intervallo [-50, 50] da permetterci di avere
#una percentuale di casi accettati del circa 50%
zeta = 27/16
He.SetParams([zeta,1])
N = 100
passo = 1
R = np.zeros((2,3),float)
rate = np.zeros(N)
gen = np.random.default_rng()

local_energy = open ('./TaskA/A_inter/local_energy.txt','wb')
r1 = open ('./TaskA/A_inter/pos_r1.txt','wb')
r2 = open ('./TaskA/A_inter/pos_r2.txt','wb')

for i in range(N):
    R[0,:] = np.array(gen.uniform(-50,50,3))
```

```
    R[1,:] = np.array(gen.uniform(-50,50,3))
    He.SetCoords(R)

    EnergyList,CoordsList_r1,CoordsList_r2,accpt=He.VMC(100000,delta = 1.618034)

    np.save(local_energy, EnergyList[::passo])
    np.save(r1, CoordsList_r1[::passo,:])
    np.save(r2, CoordsList_r2[::passo,:])
    rate[i] = accpt*100
acceptance_rate = np.mean(rate)
var = np.var(rate)
local_energy.close()
r1.close()
r2.close()
print('Z= %7.4f , Acceptance Rate= %6.2f , varianza= %10.6f' % (zeta,acceptance_ra
```

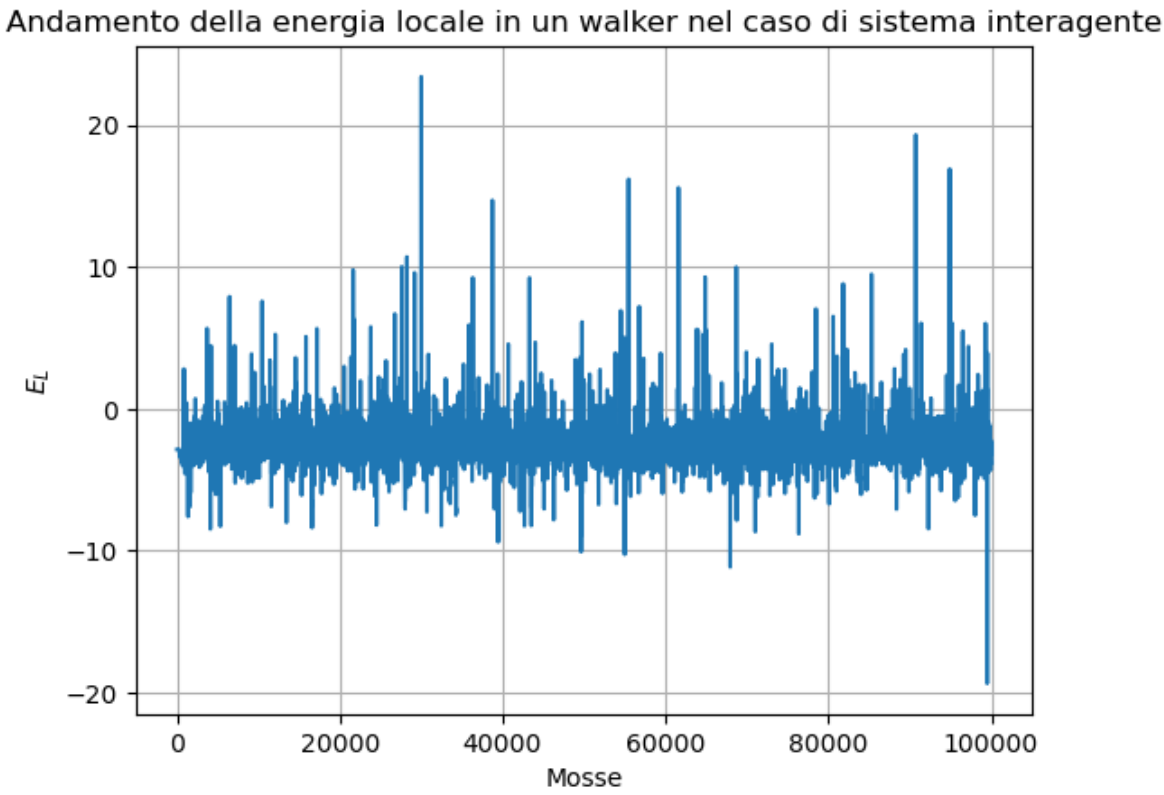
```
Z=  1.6875 , Acceptance Rate=  48.24 , varianza=   0.045467
```

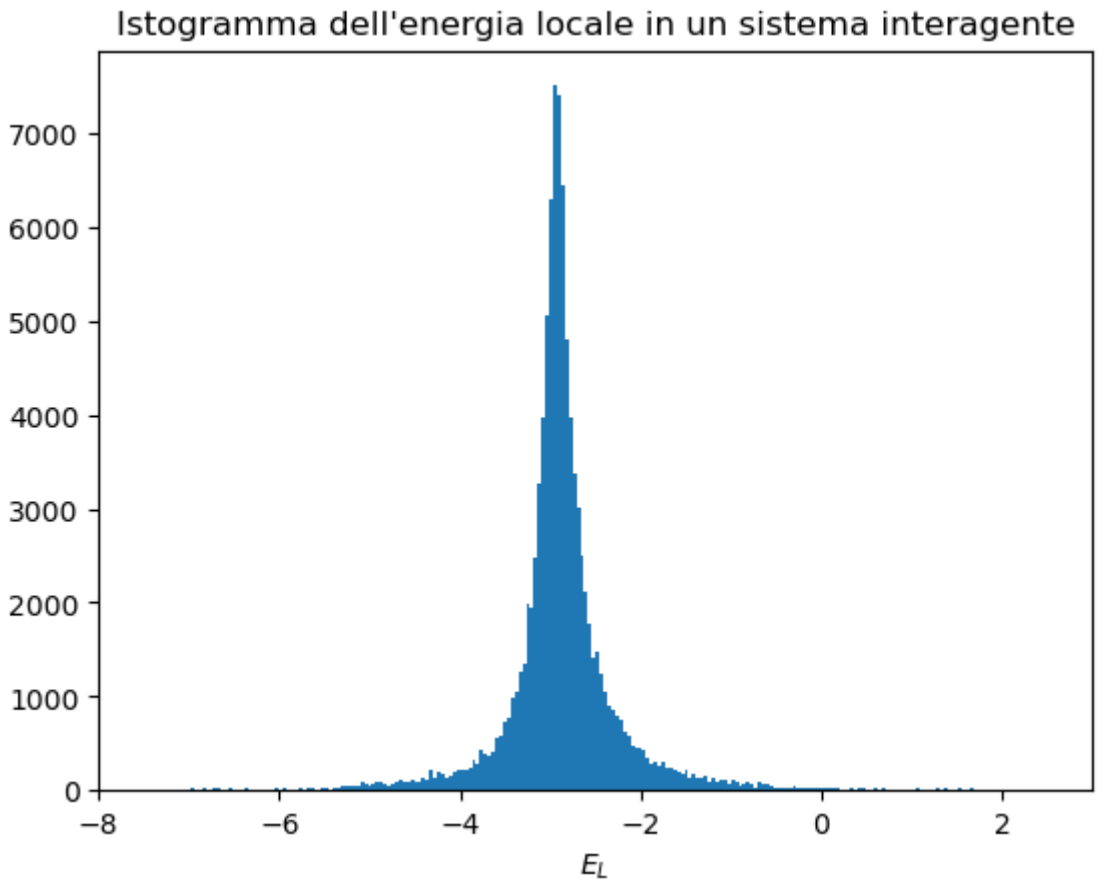
In [17]:
```
energy = np.load("./TaskA/A_inter/local_energy.txt")
r1 = np.load("./TaskA/A_inter/pos_r1.txt")
r2 = np.load("./TaskA/A_inter/pos_r2.txt")
print("Energia media = %5.4f"%(energy.mean()))
print("Valore di riferimento analitico = -2.8477")
```

```
Energia media = -2.8630
Valore di riferimento analitico = -2.8477
```

Seguono, adesso, grafici dell'andamento dell'energia locale e il suo istogramma e uno scatter plot per visualizzare la densità di probabilità nelle due dimensioni (r_1, r_2)

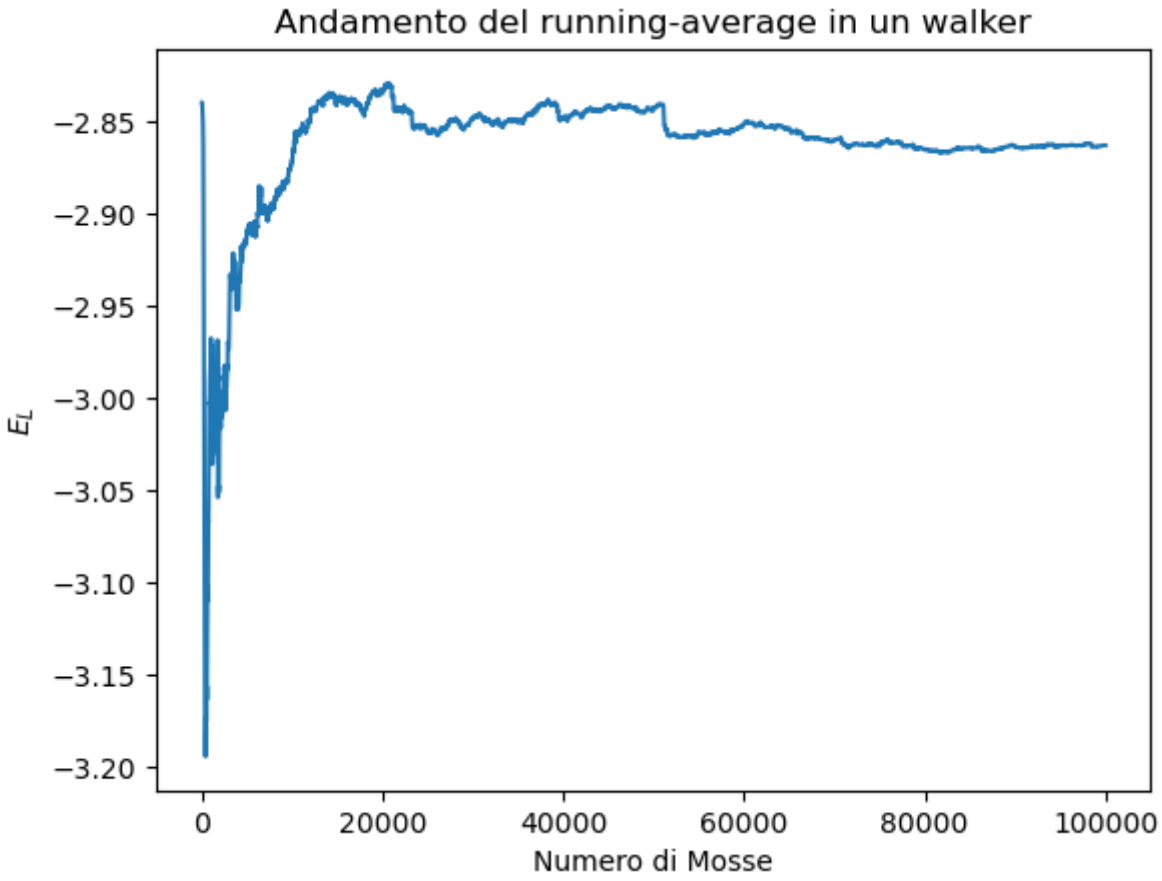
In [20]:
```
with open('./TaskA/A_inter/local_energy.txt', 'rb') as file:
    coeff = np.load(file)
    energyList = np.load(file)
plt.plot(energyList)
plt.title("Andamento della energia locale in un walker nel caso di sistema interage
plt.grid ()
plt.xlabel("Mosse")
plt.ylabel("$E_L$")
plt.show()
#
plt.hist(energyList,bins=1000)
plt.title("Istogramma dell'energia locale in un sistema interagente")
plt.xlabel("$E_L$")
plt.xlim (-8, 3)
plt.show()
```





In [22]:
```
passo = 1
with open('./TaskA/A_inter/local_energy.txt', 'rb') as file:
    energyList = np.load(file)
    M = len(energyList)
run_avg=np.zeros(M)
mosse = np.arange(0,M*passo,1*passo)
for t in range(M):
    run_avg[t]=np.mean(energyList[:t])
plt.plot(mosse,run_avg)
```

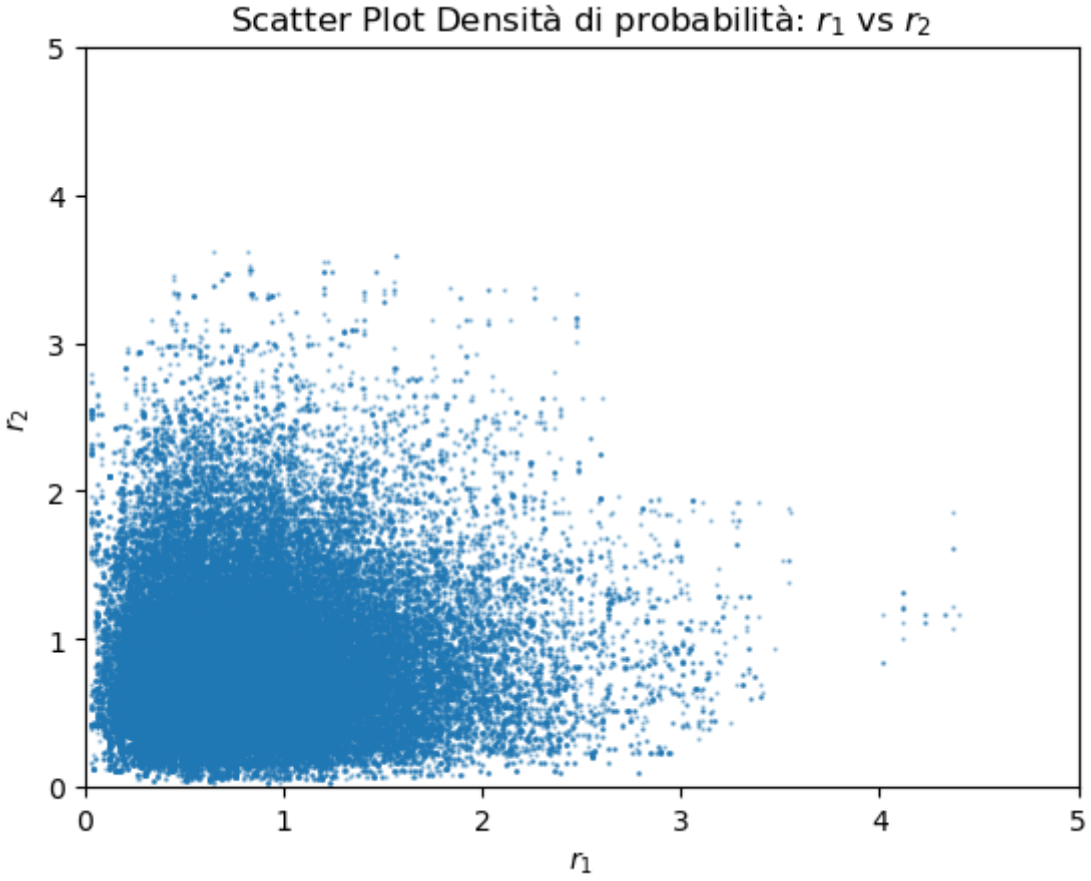
```
plt.xlabel("Numero di Mosse")
plt.ylabel("$E_L$")
plt.title("Andamento del running-average in un walker")
plt.show()
```



Scatter Plot

In [23]:
```
with open('./TaskA/A_inter/pos_r1.txt', 'rb') as file:
    r1_vector = np.load(file)
    r1 = np.linalg.norm(r1_vector[1000:],axis=1)
with open('./TaskA/A_inter/pos_r2.txt', 'rb') as file:
    r2_vector = np.load(file)
    r2 = np.linalg.norm(r2_vector[1000:],axis=1)
plt.scatter(r1,r2, s=0.5, alpha=0.4)
plt.xlabel("$r_1$")
plt.ylabel("$r_2$")
plt.xlim(0,5)
plt.ylim(0,5)
plt.title("Scatter Plot Densità di probabilità: $r_1$ vs $r_2$")
plt.plot()
```

Out[23]:
```
[]
```



Task B.

In [12]:

```python
class He_VMC_B:
    def __init__(self):
        self.coords = np.zeros((2,3))
        self.rg = np.random.default_rng()

    def SetParams(self,params):
        self.params=params.copy()

    def SetCoords(self,coords):
        self.coords=coords.copy()

    def WaveFunction(self,coords):
        alpha=self.params[0]
        r1 = np.linalg.norm(coords[0,:])
        r2 = np.linalg.norm(coords[1,:])
        r12 = np.linalg.norm(coords[0,:]-coords[1,:])
        return np.exp(-2.*(r1+r2)+(0.5*r12)/(1+alpha*r12))


    def LocalEnergy(self,coords):
        KE = -0.5*self.LaplacianPsiOverPsi(coords)
        V  = self.Potential(coords)
        return V+KE

    def Potential(self,coords):
        #Utilizzo solo il potenziale per il sistema con elettroni interagenti
        #e non ho più bisogno di segliere tra i due sistemi
        r1 = np.linalg.norm(coords[0,:])
        r2 = np.linalg.norm(coords[1,:])
        r12 = np.linalg.norm(coords[0,:]-coords[1,:])
        return (-2*r12*(r1 + r2) + r1*r2)/(r1*r2*r12)
```

```
    def LaplacianPsiOverPsi(self,coords,delta=0.0001):
        total=0.0
        tempVal3=self.WaveFunction(coords)
        for i in range(0,len(coords)):
            for j in range(0,len(coords[0])):
                coords[i,j]=coords[i,j]+delta
                tempVal=self.WaveFunction(coords)
                coords[i,j]=coords[i,j]-2*delta
                tempVal2=self.WaveFunction(coords)
                coords[i,j]=coords[i,j]+delta
                total +=(tempVal+tempVal2)-2.0*tempVal3
        return total/(delta*delta*tempVal3)

    def VMC(self,numSteps=1000,delta=1.618034):
        EnergyList=np.zeros(numSteps)
        CoordsList_r1=np.zeros((numSteps,3))
        CoordsList_r2=np.zeros((numSteps,3))
        movesAttempted=0.0
        movesAccepted=0.0
        Psi=self.WaveFunction(self.coords)
        energy=self.LocalEnergy(self.coords)
        for step in range(numSteps):
            nu1 = np.random.randint(2)
            nu2 = np.random.randint(3)
            RT = self.coords.copy()
            RT[nu1,nu2] = self.coords[nu1,nu2] + self.rg.uniform(-delta,delta,1)
            newPsi=self.WaveFunction(RT)
            if ( newPsi**2/Psi**2 > self.rg.random() ):
                self.coords=RT.copy()
                Psi=newPsi
                movesAccepted+=1.
                energy=self.LocalEnergy(self.coords)
            movesAttempted+=1.
            EnergyList[step] = energy
            CoordsList_r1[step,:] = self.coords[0,:]
            CoordsList_r2[step,:] = self.coords[1,:]
        return EnergyList,CoordsList_r1,CoordsList_r2,movesAccepted/movesAttempted

    #Rispetto alla classe precedente abbiamo aggiunto la parte di ottimizzazione
    #da usare come assieme al VMC sopra. Essa consiste in un'interpolazione su grig
    #che ci permette di determinare l'andamento di una variabile, nel nostro caso l
    #di punti equamente spaziati partendo da valori discreti.
    def Griglia(self,start,stop,step,M,N,delta,passo=1):
        #Inizializzo le posizioni dei due elettroni nelle tre coordinate come dei r
        R=np.zeros((2,3),float)
        n = int(0)
        #Ciclo for che ci permette di selezionare delle beta da un valore iniziale
        #con un certo step; per il caso in esame si è scelto di utilizzare delle be
        #in quanto sono molto più favorevoli delle configurazioni con gli elettron
        for beta in np.arange(start,stop,step):
            n = n + 1
            self.SetParams([beta])
            #Creo dei file .txt dove salvare le coordinate e le energie locali trov
            error = open ('./Analisi_Errore/delta_0.5'+str(n)+'.txt','wb')
            energy = open('./TaskB/local_energy'+str(n)+'.txt','wb')
            r1 = open('./TaskB/r1'+str(n)+'.txt','wb')
            r2 = open('./TaskB/r2'+str(n)+'.txt','wb')
            np.save(delta, energy, beta)
            np.save(energy,beta)
            np.save(r1,beta)
            np.save(r2,beta)
            rate = np.zeros(N)
            for i in range(N):
```

```
                ##Generazione delle coordinate all'interno dell'intervallo
                #che abbiamo imposto
                R[0,:] = np.array(np.random.uniform(-50,50,3))
                R[1,:] = np.array(np.random.uniform(-50,50,3))
                self.SetCoords(R)
                #richiamo l'algoritmo metropolis e mi salvo i risultati nei file ap
                energyList,coordsList_r1,coordsList_r2,accpt=self.VMC(100000,delta
                np.save(energy,energyList[::passo])
                np.save(r1,coordsList_r1[::passo,:])
                np.save(r2,coordsList_r2[::passo,:])
                rate[i] = accpt*100
            #Mi calcolo l'acceptance rate e la varianza per poi concludere chiuder
            #i file
            rate_m = np.mean(rate)
            rate_var = np.var(rate)
            energy.close()
            r1.close()
            r2.close()
            print('beta= %7.4f , accpt= %6.2f , var= %10.6f' % (beta,rate_m,rate_va
```

In [317…

```
He2=He_VMC_B()
#Definisco l'intervallo di lavoro, in particolare ho scelto
#di partire da 0.05 e arrivare a 0.5 con step di 0.005
#la scelta di questo intervallo è dovuto al grosso costo computazionale
#delle operazioni effettuate e sapendo che il parametro beta
#più è basso più il sistema si trova energicamente più favorevole
start=0.05
stop=0.5
step=0.005
delta=1.5
N = 10 #numero walker
He2.Griglia(start,stop,step,100000,N,delta)
```

```
beta=  0.0500 , accpt=  50.48 , var=   0.021707
beta=  0.0550 , accpt=  50.45 , var=   0.062966
beta=  0.0600 , accpt=  50.18 , var=   0.023328
beta=  0.0650 , accpt=  50.10 , var=   0.053866
beta=  0.0700 , accpt=  50.19 , var=   0.031119
beta=  0.0750 , accpt=  50.03 , var=   0.048154
beta=  0.0800 , accpt=  50.00 , var=   0.027127
beta=  0.0850 , accpt=  49.85 , var=   0.037361
beta=  0.0900 , accpt=  49.81 , var=   0.029477
beta=  0.0950 , accpt=  49.68 , var=   0.036349
beta=  0.1000 , accpt=  49.80 , var=   0.031818
beta=  0.1050 , accpt=  49.69 , var=   0.018242
beta=  0.1100 , accpt=  49.58 , var=   0.036472
beta=  0.1150 , accpt=  49.53 , var=   0.029293
beta=  0.1200 , accpt=  49.58 , var=   0.026474
beta=  0.1250 , accpt=  49.42 , var=   0.016697
beta=  0.1300 , accpt=  49.34 , var=   0.020779
beta=  0.1350 , accpt=  49.27 , var=   0.014009
beta=  0.1400 , accpt=  49.33 , var=   0.034509
beta=  0.1450 , accpt=  49.17 , var=   0.061625
beta=  0.1500 , accpt=  49.28 , var=   0.026638
beta=  0.1550 , accpt=  49.10 , var=   0.055822
beta=  0.1600 , accpt=  49.11 , var=   0.020323
beta=  0.1650 , accpt=  49.01 , var=   0.014840
beta=  0.1700 , accpt=  48.95 , var=   0.010664
beta=  0.1750 , accpt=  49.03 , var=   0.019366
beta=  0.1800 , accpt=  48.84 , var=   0.053854
beta=  0.1850 , accpt=  48.86 , var=   0.024378
beta=  0.1900 , accpt=  48.85 , var=   0.025822
beta=  0.1950 , accpt=  48.73 , var=   0.038110
beta=  0.2000 , accpt=  48.76 , var=   0.066999
beta=  0.2050 , accpt=  48.75 , var=   0.028632
beta=  0.2100 , accpt=  48.70 , var=   0.062515
beta=  0.2150 , accpt=  48.70 , var=   0.052200
beta=  0.2200 , accpt=  48.64 , var=   0.050070
beta=  0.2250 , accpt=  48.68 , var=   0.008153
beta=  0.2300 , accpt=  48.57 , var=   0.024222
beta=  0.2350 , accpt=  48.56 , var=   0.029717
beta=  0.2400 , accpt=  48.50 , var=   0.031387
beta=  0.2450 , accpt=  48.59 , var=   0.053042
beta=  0.2500 , accpt=  48.40 , var=   0.014753
beta=  0.2550 , accpt=  48.40 , var=   0.047198
beta=  0.2600 , accpt=  48.45 , var=   0.027659
beta=  0.2650 , accpt=  48.29 , var=   0.071070
beta=  0.2700 , accpt=  48.19 , var=   0.081598
beta=  0.2750 , accpt=  48.20 , var=   0.036894
beta=  0.2800 , accpt=  48.22 , var=   0.023964
beta=  0.2850 , accpt=  48.36 , var=   0.066674
beta=  0.2900 , accpt=  48.37 , var=   0.083900
beta=  0.2950 , accpt=  48.20 , var=   0.073756
beta=  0.3000 , accpt=  48.07 , var=   0.060946
beta=  0.3050 , accpt=  48.19 , var=   0.058425
beta=  0.3100 , accpt=  48.21 , var=   0.036949
beta=  0.3150 , accpt=  48.09 , var=   0.070158
beta=  0.3200 , accpt=  47.98 , var=   0.061882
beta=  0.3250 , accpt=  48.07 , var=   0.063077
beta=  0.3300 , accpt=  48.11 , var=   0.044748
beta=  0.3350 , accpt=  48.10 , var=   0.002491
beta=  0.3400 , accpt=  47.92 , var=   0.053625
beta=  0.3450 , accpt=  48.00 , var=   0.050986
beta=  0.3500 , accpt=  48.10 , var=   0.041467
beta=  0.3550 , accpt=  47.83 , var=   0.014197
beta=  0.3600 , accpt=  48.02 , var=   0.038298
beta=  0.3650 , accpt=  47.85 , var=   0.042502
```

```
beta=  0.3700 , accpt=  47.82 , var=   0.042461
beta=  0.3750 , accpt=  47.85 , var=   0.047918
beta=  0.3800 , accpt=  47.71 , var=   0.075818
beta=  0.3850 , accpt=  47.83 , var=   0.050016
beta=  0.3900 , accpt=  47.64 , var=   0.022790
beta=  0.3950 , accpt=  47.69 , var=   0.033182
beta=  0.4000 , accpt=  47.66 , var=   0.056389
beta=  0.4050 , accpt=  47.64 , var=   0.040364
beta=  0.4100 , accpt=  47.75 , var=   0.042623
beta=  0.4150 , accpt=  47.65 , var=   0.016910
beta=  0.4200 , accpt=  47.52 , var=   0.030444
beta=  0.4250 , accpt=  47.69 , var=   0.065662
beta=  0.4300 , accpt=  47.52 , var=   0.030810
beta=  0.4350 , accpt=  47.54 , var=   0.040358
beta=  0.4400 , accpt=  47.55 , var=   0.028503
beta=  0.4450 , accpt=  47.56 , var=   0.040346
beta=  0.4500 , accpt=  47.64 , var=   0.018341
beta=  0.4550 , accpt=  47.48 , var=   0.017322
beta=  0.4600 , accpt=  47.53 , var=   0.023173
beta=  0.4650 , accpt=  47.56 , var=   0.061090
beta=  0.4700 , accpt=  47.49 , var=   0.043990
beta=  0.4750 , accpt=  47.43 , var=   0.062883
beta=  0.4800 , accpt=  47.49 , var=   0.073326
beta=  0.4850 , accpt=  47.31 , var=   0.029626
beta=  0.4900 , accpt=  47.18 , var=   0.090065
beta=  0.4950 , accpt=  47.44 , var=   0.107839
```

In [318…

```python
#Imposto l'intervallo e il numero dei walk
start=0.05
stop=0.5
step=0.005
N = 10
#trovo il numero dei beta e utilizzo il ciclo for per aprire il file
#relativo all'energia e utilizzo i valori trovati prima per calcolare l'energia
num_beta = int((stop-start)//step)
for i in range(1,num_beta+1):
    energy = np.zeros(N)
    with open('./TaskB/local_energy'+str(i)+'.txt', 'rb') as file:
        coeff = np.load(file)
        for j in range(N):
            energyList = np.load(file)
            energy[j] = np.mean(energyList[40000//passo:])
    result = np.mean(energy)
    std = np.std(energy) #calcolo la deviazione standard
    print('beta= %7.4f , energy= %10.6f , std= %10.6f' % (coeff,result,std))
```

```
beta=  0.0500 , energy=  -2.873591 , std=   0.004615
beta=  0.0550 , energy=  -2.872892 , std=   0.004499
beta=  0.0600 , energy=  -2.870443 , std=   0.005097
beta=  0.0650 , energy=  -2.873970 , std=   0.005339
beta=  0.0700 , energy=  -2.874396 , std=   0.003694
beta=  0.0750 , energy=  -2.877870 , std=   0.005240
beta=  0.0800 , energy=  -2.875036 , std=   0.005752
beta=  0.0850 , energy=  -2.877666 , std=   0.007082
beta=  0.0900 , energy=  -2.879705 , std=   0.006392
beta=  0.0950 , energy=  -2.875773 , std=   0.004637
beta=  0.1000 , energy=  -2.876906 , std=   0.006509
beta=  0.1050 , energy=  -2.877616 , std=   0.004717
beta=  0.1100 , energy=  -2.876282 , std=   0.003703
beta=  0.1150 , energy=  -2.878553 , std=   0.002744
beta=  0.1200 , energy=  -2.877136 , std=   0.004217
beta=  0.1250 , energy=  -2.878601 , std=   0.003350
beta=  0.1300 , energy=  -2.878518 , std=   0.005234
beta=  0.1350 , energy=  -2.877116 , std=   0.002064
beta=  0.1400 , energy=  -2.877425 , std=   0.003401
beta=  0.1450 , energy=  -2.877599 , std=   0.006522
beta=  0.1500 , energy=  -2.880777 , std=   0.004151
beta=  0.1550 , energy=  -2.880675 , std=   0.005692
beta=  0.1600 , energy=  -2.875303 , std=   0.005662
beta=  0.1650 , energy=  -2.879150 , std=   0.006369
beta=  0.1700 , energy=  -2.875801 , std=   0.005991
beta=  0.1750 , energy=  -2.876816 , std=   0.005575
beta=  0.1800 , energy=  -2.878412 , std=   0.005782
beta=  0.1850 , energy=  -2.876911 , std=   0.005074
beta=  0.1900 , energy=  -2.876931 , std=   0.005906
beta=  0.1950 , energy=  -2.878524 , std=   0.002908
beta=  0.2000 , energy=  -2.876364 , std=   0.006468
beta=  0.2050 , energy=  -2.879802 , std=   0.004472
beta=  0.2100 , energy=  -2.873565 , std=   0.006003
beta=  0.2150 , energy=  -2.877520 , std=   0.006226
beta=  0.2200 , energy=  -2.877764 , std=   0.005171
beta=  0.2250 , energy=  -2.876730 , std=   0.003571
beta=  0.2300 , energy=  -2.870305 , std=   0.004654
beta=  0.2350 , energy=  -2.874272 , std=   0.008611
beta=  0.2400 , energy=  -2.872733 , std=   0.006574
beta=  0.2450 , energy=  -2.877833 , std=   0.005374
beta=  0.2500 , energy=  -2.874617 , std=   0.004099
beta=  0.2550 , energy=  -2.874608 , std=   0.007055
beta=  0.2600 , energy=  -2.872899 , std=   0.006274
beta=  0.2650 , energy=  -2.870877 , std=   0.005924
beta=  0.2700 , energy=  -2.871856 , std=   0.006100
beta=  0.2750 , energy=  -2.870982 , std=   0.006432
beta=  0.2800 , energy=  -2.872681 , std=   0.005029
beta=  0.2850 , energy=  -2.874240 , std=   0.007250
beta=  0.2900 , energy=  -2.874450 , std=   0.006896
beta=  0.2950 , energy=  -2.874230 , std=   0.006696
beta=  0.3000 , energy=  -2.867902 , std=   0.007312
beta=  0.3050 , energy=  -2.870838 , std=   0.004457
beta=  0.3100 , energy=  -2.867981 , std=   0.006104
beta=  0.3150 , energy=  -2.873068 , std=   0.009124
beta=  0.3200 , energy=  -2.869008 , std=   0.003925
beta=  0.3250 , energy=  -2.873317 , std=   0.006286
beta=  0.3300 , energy=  -2.873218 , std=   0.005028
beta=  0.3350 , energy=  -2.865654 , std=   0.003263
beta=  0.3400 , energy=  -2.867755 , std=   0.005807
beta=  0.3450 , energy=  -2.869832 , std=   0.005738
beta=  0.3500 , energy=  -2.870615 , std=   0.004147
beta=  0.3550 , energy=  -2.865985 , std=   0.005087
beta=  0.3600 , energy=  -2.866909 , std=   0.006126
beta=  0.3650 , energy=  -2.865880 , std=   0.008282
```

```
beta=  0.3700 , energy=  -2.868584 , std=   0.005862
beta=  0.3750 , energy=  -2.864702 , std=   0.008837
beta=  0.3800 , energy=  -2.865361 , std=   0.005311
beta=  0.3850 , energy=  -2.865146 , std=   0.008247
beta=  0.3900 , energy=  -2.863031 , std=   0.003280
beta=  0.3950 , energy=  -2.863587 , std=   0.005605
beta=  0.4000 , energy=  -2.865582 , std=   0.007237
beta=  0.4050 , energy=  -2.864403 , std=   0.008606
beta=  0.4100 , energy=  -2.861586 , std=   0.006224
beta=  0.4150 , energy=  -2.863254 , std=   0.003364
beta=  0.4200 , energy=  -2.862561 , std=   0.008332
beta=  0.4250 , energy=  -2.865758 , std=   0.006683
beta=  0.4300 , energy=  -2.859579 , std=   0.004732
beta=  0.4350 , energy=  -2.862335 , std=   0.004873
beta=  0.4400 , energy=  -2.861210 , std=   0.006480
beta=  0.4450 , energy=  -2.861642 , std=   0.008554
beta=  0.4500 , energy=  -2.862886 , std=   0.005380
beta=  0.4550 , energy=  -2.860295 , std=   0.006094
beta=  0.4600 , energy=  -2.861267 , std=   0.005285
beta=  0.4650 , energy=  -2.861002 , std=   0.008807
beta=  0.4700 , energy=  -2.858092 , std=   0.007242
beta=  0.4750 , energy=  -2.856356 , std=   0.004356
beta=  0.4800 , energy=  -2.855645 , std=   0.006779
beta=  0.4850 , energy=  -2.859010 , std=   0.009496
beta=  0.4900 , energy=  -2.853618 , std=   0.008887
beta=  0.4950 , energy=  -2.859048 , std=   0.006120
```

In [224…

```python
He2=He_VMC_B()
#Definisco l'intervallo di lavoro, in particolare ho scelto
#di partire da 0.5 e arrivare a 1 con step di 0.005
#per completare l'analisi dei beta
start=0.5
stop=1
step=0.005
delta=1.5
N = 10 #numero walker
He2.Griglia(start,stop,step,100000,N,delta)
```

```
beta=  0.5000 , accpt=  47.26 , var=   0.020064
beta=  0.5050 , accpt=  47.48 , var=   0.053418
beta=  0.5100 , accpt=  47.33 , var=   0.026128
beta=  0.5150 , accpt=  47.40 , var=   0.039049
beta=  0.5200 , accpt=  47.34 , var=   0.051952
beta=  0.5250 , accpt=  47.35 , var=   0.075182
beta=  0.5300 , accpt=  47.17 , var=   0.013344
beta=  0.5350 , accpt=  47.13 , var=   0.032757
beta=  0.5400 , accpt=  47.21 , var=   0.040954
beta=  0.5450 , accpt=  47.19 , var=   0.043070
beta=  0.5500 , accpt=  47.26 , var=   0.071399
beta=  0.5550 , accpt=  47.10 , var=   0.061321
beta=  0.5600 , accpt=  47.23 , var=   0.028441
beta=  0.5650 , accpt=  47.21 , var=   0.039759
beta=  0.5700 , accpt=  47.08 , var=   0.024532
beta=  0.5750 , accpt=  47.22 , var=   0.023765
beta=  0.5800 , accpt=  47.12 , var=   0.028903
beta=  0.5850 , accpt=  47.06 , var=   0.066502
beta=  0.5900 , accpt=  47.05 , var=   0.018948
beta=  0.5950 , accpt=  47.30 , var=   0.056835
beta=  0.6000 , accpt=  47.04 , var=   0.025407
beta=  0.6050 , accpt=  47.05 , var=   0.061963
beta=  0.6100 , accpt=  47.14 , var=   0.052204
beta=  0.6150 , accpt=  47.12 , var=   0.038302
beta=  0.6200 , accpt=  47.06 , var=   0.072963
beta=  0.6250 , accpt=  47.08 , var=   0.053434
beta=  0.6300 , accpt=  47.03 , var=   0.023796
beta=  0.6350 , accpt=  46.97 , var=   0.042210
beta=  0.6400 , accpt=  47.11 , var=   0.027557
beta=  0.6450 , accpt=  47.01 , var=   0.029917
beta=  0.6500 , accpt=  47.01 , var=   0.079394
beta=  0.6550 , accpt=  47.08 , var=   0.063957
beta=  0.6600 , accpt=  46.85 , var=   0.056285
beta=  0.6650 , accpt=  46.88 , var=   0.015985
beta=  0.6700 , accpt=  46.94 , var=   0.046051
beta=  0.6750 , accpt=  46.83 , var=   0.019883
beta=  0.6800 , accpt=  46.87 , var=   0.076534
beta=  0.6850 , accpt=  46.76 , var=   0.016786
beta=  0.6900 , accpt=  46.80 , var=   0.023292
beta=  0.6950 , accpt=  46.93 , var=   0.074874
beta=  0.7000 , accpt=  46.96 , var=   0.043222
beta=  0.7050 , accpt=  46.84 , var=   0.045620
beta=  0.7100 , accpt=  46.79 , var=   0.010517
beta=  0.7150 , accpt=  46.85 , var=   0.013255
beta=  0.7200 , accpt=  46.73 , var=   0.038251
beta=  0.7250 , accpt=  46.77 , var=   0.087975
beta=  0.7300 , accpt=  46.88 , var=   0.068705
beta=  0.7350 , accpt=  46.74 , var=   0.047153
beta=  0.7400 , accpt=  46.81 , var=   0.026804
beta=  0.7450 , accpt=  46.84 , var=   0.053182
beta=  0.7500 , accpt=  46.78 , var=   0.018961
beta=  0.7550 , accpt=  46.74 , var=   0.029844
beta=  0.7600 , accpt=  46.85 , var=   0.073318
beta=  0.7650 , accpt=  46.76 , var=   0.022720
beta=  0.7700 , accpt=  46.73 , var=   0.026414
beta=  0.7750 , accpt=  46.78 , var=   0.030303
beta=  0.7800 , accpt=  46.78 , var=   0.027952
beta=  0.7850 , accpt=  46.67 , var=   0.042239
beta=  0.7900 , accpt=  46.73 , var=   0.062730
beta=  0.7950 , accpt=  46.68 , var=   0.063076
beta=  0.8000 , accpt=  46.70 , var=   0.020340
beta=  0.8050 , accpt=  46.53 , var=   0.032792
beta=  0.8100 , accpt=  46.76 , var=   0.038903
beta=  0.8150 , accpt=  46.59 , var=   0.062126
```

```
beta=  0.8200 , accpt=  46.56 , var=   0.063018
beta=  0.8250 , accpt=  46.62 , var=   0.048808
beta=  0.8300 , accpt=  46.61 , var=   0.058456
beta=  0.8350 , accpt=  46.62 , var=   0.017886
beta=  0.8400 , accpt=  46.68 , var=   0.048230
beta=  0.8450 , accpt=  46.69 , var=   0.040965
beta=  0.8500 , accpt=  46.65 , var=   0.059692
beta=  0.8550 , accpt=  46.61 , var=   0.018121
beta=  0.8600 , accpt=  46.54 , var=   0.025319
beta=  0.8650 , accpt=  46.58 , var=   0.042394
beta=  0.8700 , accpt=  46.58 , var=   0.037359
beta=  0.8750 , accpt=  46.61 , var=   0.054781
beta=  0.8800 , accpt=  46.55 , var=   0.010819
beta=  0.8850 , accpt=  46.61 , var=   0.047763
beta=  0.8900 , accpt=  46.56 , var=   0.136068
beta=  0.8950 , accpt=  46.57 , var=   0.042823
beta=  0.9000 , accpt=  46.59 , var=   0.045864
beta=  0.9050 , accpt=  46.63 , var=   0.023559
beta=  0.9100 , accpt=  46.49 , var=   0.033579
beta=  0.9150 , accpt=  46.68 , var=   0.022150
beta=  0.9200 , accpt=  46.50 , var=   0.066472
beta=  0.9250 , accpt=  46.44 , var=   0.031516
beta=  0.9300 , accpt=  46.64 , var=   0.021245
beta=  0.9350 , accpt=  46.41 , var=   0.025201
beta=  0.9400 , accpt=  46.44 , var=   0.036798
beta=  0.9450 , accpt=  46.59 , var=   0.047248
beta=  0.9500 , accpt=  46.52 , var=   0.027846
beta=  0.9550 , accpt=  46.42 , var=   0.020588
beta=  0.9600 , accpt=  46.45 , var=   0.042038
beta=  0.9650 , accpt=  46.51 , var=   0.075964
beta=  0.9700 , accpt=  46.46 , var=   0.085080
beta=  0.9750 , accpt=  46.41 , var=   0.056792
beta=  0.9800 , accpt=  46.47 , var=   0.057885
beta=  0.9850 , accpt=  46.39 , var=   0.058953
beta=  0.9900 , accpt=  46.48 , var=   0.025125
beta=  0.9950 , accpt=  46.30 , var=   0.026245
```

In [315…

```python
#Imposto l'intervallo e il numero dei walk
start=0.05
stop=0.5
step=0.005
N = 10
#trovo il numero dei beta e utilizzo il ciclo for per aprire il file
#relativo all'energia e utilizzo i valori trovati prima per calcolare l'energia
num_beta = int((stop-start)//step)
for i in range(1,num_beta+1):
    energy = np.zeros(N)
    with open('./TaskB/local_energy'+str(i)+'.txt', 'rb') as file:
        coeff = np.load(file)
        for j in range(N):
            energyList = np.load(file)
            energy[j] = np.mean(energyList[40000//passo:])
    result = np.mean(energy)
    std = np.std(energy) #calcolo la deviazione standard
    print('beta= %7.4f , energy= %10.6f , std= %10.6f' % (coeff,result,std))
```

```
beta=  0.5000 , energy=  -2.853399 , std=   0.006747
beta=  0.5050 , energy=  -2.855785 , std=   0.007686
beta=  0.5100 , energy=  -2.855647 , std=   0.007702
beta=  0.5150 , energy=  -2.857526 , std=   0.007462
beta=  0.5200 , energy=  -2.856457 , std=   0.006522
beta=  0.5250 , energy=  -2.854251 , std=   0.009846
beta=  0.5300 , energy=  -2.852071 , std=   0.007663
beta=  0.5350 , energy=  -2.851612 , std=   0.007578
beta=  0.5400 , energy=  -2.855360 , std=   0.006078
beta=  0.5450 , energy=  -2.851282 , std=   0.007208
beta=  0.5500 , energy=  -2.854665 , std=   0.008765
beta=  0.5550 , energy=  -2.849421 , std=   0.008777
beta=  0.5600 , energy=  -2.856035 , std=   0.006055
beta=  0.5650 , energy=  -2.855090 , std=   0.007768
beta=  0.5700 , energy=  -2.847521 , std=   0.007935
beta=  0.5750 , energy=  -2.854467 , std=   0.007930
beta=  0.5800 , energy=  -2.852881 , std=   0.005863
beta=  0.5850 , energy=  -2.850852 , std=   0.006394
beta=  0.5900 , energy=  -2.846539 , std=   0.006355
beta=  0.5950 , energy=  -2.855381 , std=   0.006465
beta=  0.6000 , energy=  -2.847116 , std=   0.004300
beta=  0.6050 , energy=  -2.844733 , std=   0.007678
beta=  0.6100 , energy=  -2.849927 , std=   0.008312
beta=  0.6150 , energy=  -2.852033 , std=   0.008077
beta=  0.6200 , energy=  -2.848925 , std=   0.006637
beta=  0.6250 , energy=  -2.845944 , std=   0.009586
beta=  0.6300 , energy=  -2.846859 , std=   0.007050
beta=  0.6350 , energy=  -2.842257 , std=   0.006056
beta=  0.6400 , energy=  -2.848130 , std=   0.007083
beta=  0.6450 , energy=  -2.844871 , std=   0.007947
beta=  0.6500 , energy=  -2.846834 , std=   0.006491
beta=  0.6550 , energy=  -2.846690 , std=   0.012261
beta=  0.6600 , energy=  -2.843849 , std=   0.008490
beta=  0.6650 , energy=  -2.840639 , std=   0.010654
beta=  0.6700 , energy=  -2.845365 , std=   0.010058
beta=  0.6750 , energy=  -2.843263 , std=   0.006629
beta=  0.6800 , energy=  -2.848394 , std=   0.004816
beta=  0.6850 , energy=  -2.842086 , std=   0.009610
beta=  0.6900 , energy=  -2.838477 , std=   0.009334
beta=  0.6950 , energy=  -2.842590 , std=   0.009657
beta=  0.7000 , energy=  -2.841499 , std=   0.009464
beta=  0.7050 , energy=  -2.838047 , std=   0.013418
beta=  0.7100 , energy=  -2.841313 , std=   0.008850
beta=  0.7150 , energy=  -2.844565 , std=   0.008149
beta=  0.7200 , energy=  -2.838596 , std=   0.010881
beta=  0.7250 , energy=  -2.838451 , std=   0.008790
beta=  0.7300 , energy=  -2.844706 , std=   0.007908
beta=  0.7350 , energy=  -2.837678 , std=   0.007730
beta=  0.7400 , energy=  -2.842484 , std=   0.007132
beta=  0.7450 , energy=  -2.842277 , std=   0.004600
beta=  0.7500 , energy=  -2.840643 , std=   0.010306
beta=  0.7550 , energy=  -2.835985 , std=   0.011128
beta=  0.7600 , energy=  -2.840486 , std=   0.006957
beta=  0.7650 , energy=  -2.838833 , std=   0.009754
beta=  0.7700 , energy=  -2.834973 , std=   0.009948
beta=  0.7750 , energy=  -2.839071 , std=   0.009681
beta=  0.7800 , energy=  -2.837338 , std=   0.010119
beta=  0.7850 , energy=  -2.839082 , std=   0.010324
beta=  0.7900 , energy=  -2.836006 , std=   0.006539
beta=  0.7950 , energy=  -2.832875 , std=   0.006405
beta=  0.8000 , energy=  -2.841596 , std=   0.006976
beta=  0.8050 , energy=  -2.831672 , std=   0.010139
beta=  0.8100 , energy=  -2.836169 , std=   0.006825
beta=  0.8150 , energy=  -2.832055 , std=   0.007246
```

```
beta=  0.8200 , energy=  -2.834368 , std=   0.008853
beta=  0.8250 , energy=  -2.835617 , std=   0.008683
beta=  0.8300 , energy=  -2.832614 , std=   0.014311
beta=  0.8350 , energy=  -2.830993 , std=   0.012115
beta=  0.8400 , energy=  -2.833525 , std=   0.008138
beta=  0.8450 , energy=  -2.834799 , std=   0.009132
beta=  0.8500 , energy=  -2.835752 , std=   0.008339
beta=  0.8550 , energy=  -2.834129 , std=   0.008289
beta=  0.8600 , energy=  -2.830866 , std=   0.008557
beta=  0.8650 , energy=  -2.833469 , std=   0.006278
beta=  0.8700 , energy=  -2.832029 , std=   0.010811
beta=  0.8750 , energy=  -2.832167 , std=   0.007220
beta=  0.8800 , energy=  -2.828693 , std=   0.008926
beta=  0.8850 , energy=  -2.832595 , std=   0.009564
beta=  0.8900 , energy=  -2.834340 , std=   0.010062
beta=  0.8950 , energy=  -2.830936 , std=   0.007870
beta=  0.9000 , energy=  -2.830057 , std=   0.010101
beta=  0.9050 , energy=  -2.831640 , std=   0.008142
beta=  0.9100 , energy=  -2.826576 , std=   0.009145
beta=  0.9150 , energy=  -2.833873 , std=   0.007955
beta=  0.9200 , energy=  -2.830900 , std=   0.005378
beta=  0.9250 , energy=  -2.827181 , std=   0.006625
beta=  0.9300 , energy=  -2.830676 , std=   0.011088
beta=  0.9350 , energy=  -2.824059 , std=   0.007826
beta=  0.9400 , energy=  -2.825955 , std=   0.010978
beta=  0.9450 , energy=  -2.830211 , std=   0.010510
```

Il valore migliore di β grazie alla quale l'energia è minore è $\beta = 0.15$ con energia $\langle E_L \rangle = (-2.88077 \pm 0.004151)$.

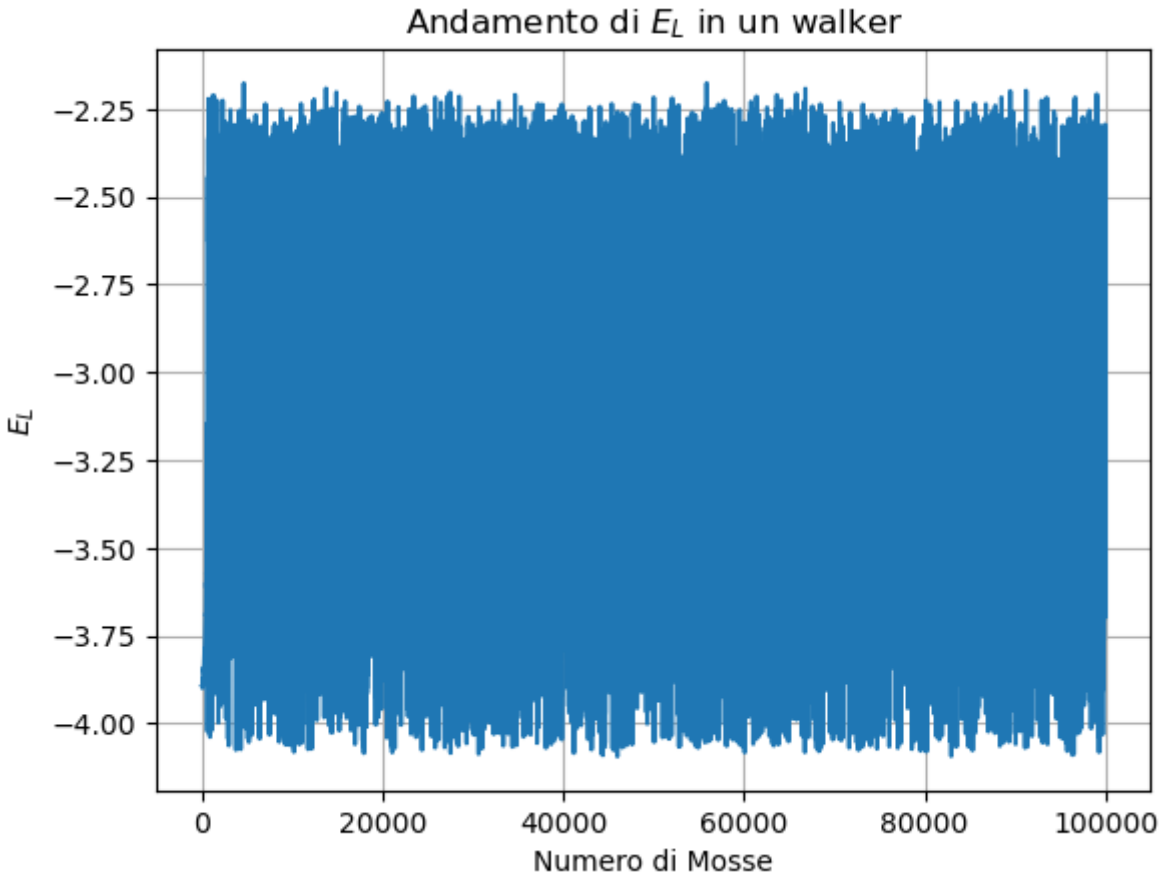
Grafici

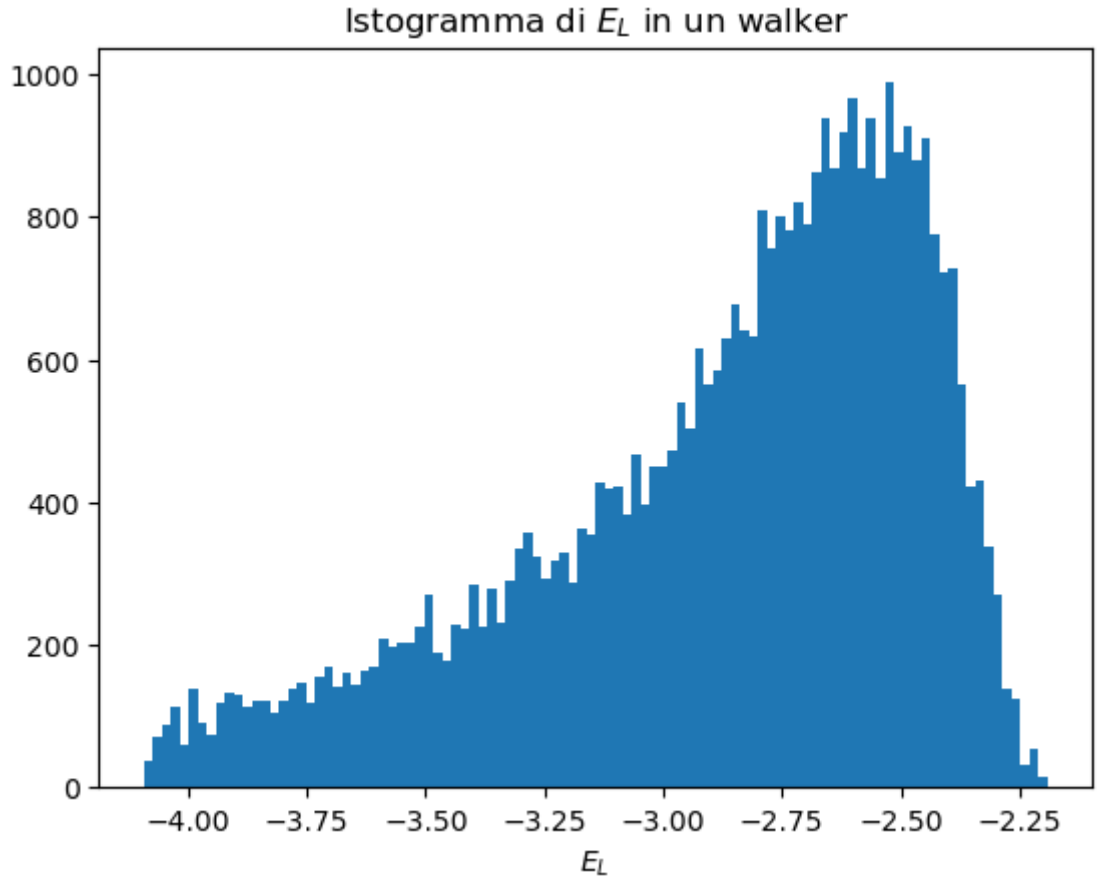
Per comodità e leggibilità si è deciso di di visualizzare i grafici relativi ai file 1 e 51 delle energie {local_energy1.txt, local_energy51.txt} e visualizzarne gli andamanti, istogrammi e running average. Inoltre è possibile visualizzare gli scatter plot relativi alle densità di probabilità delle posizioni relative {r11.txt, r21.txt, r151.txt, r251.txt}.

In [428…

```
with open('./TaskB/local_energy1.txt', 'rb') as file:
    coeff = np.load(file)
    energyList = np.load(file)
plt.plot(mosse,energyList)
plt.grid()
plt.xlabel("Numero di Mosse")
plt.ylabel("$E_L$")
plt.title("Andamento di $E_L$ in un walker")
plt.show()

#istogramma
plt.hist(energyList[60000//passo:],bins=100)
plt.title("Istogramma di $E_L$ in un walker")
plt.xlabel("$E_L$")
plt.show()
```



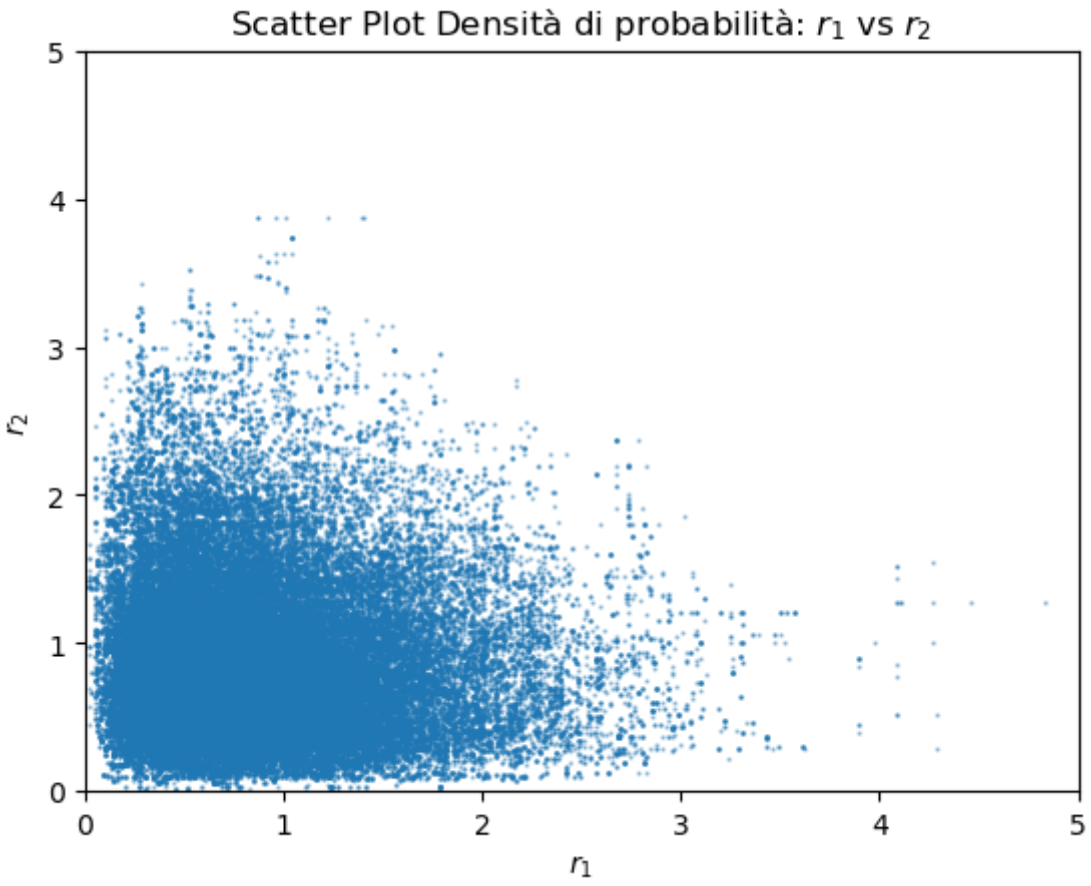


In [337…

```
with open('./TaskB/r11.txt', 'rb') as file:
    coeff = np.load(file)
    r1_vector = np.load(file)
    r1 = np.linalg.norm(r1_vector[1000:],axis=1)
with open('./TaskB/r21.txt', 'rb') as file:
    coeff = np.load(file)
```

```
    r2_vector = np.load(file)
    r2 = np.linalg.norm(r2_vector[1000:],axis=1)
plt.scatter(r1,r2, s=0.5, alpha=0.4)
plt.xlabel("$r_1$")
plt.ylabel("$r_2$")
plt.xlim(0,5)
plt.ylim(0,5)
plt.title("Scatter Plot Densità di probabilità: $r_1$ vs $r_2$")
plt.plot()
```

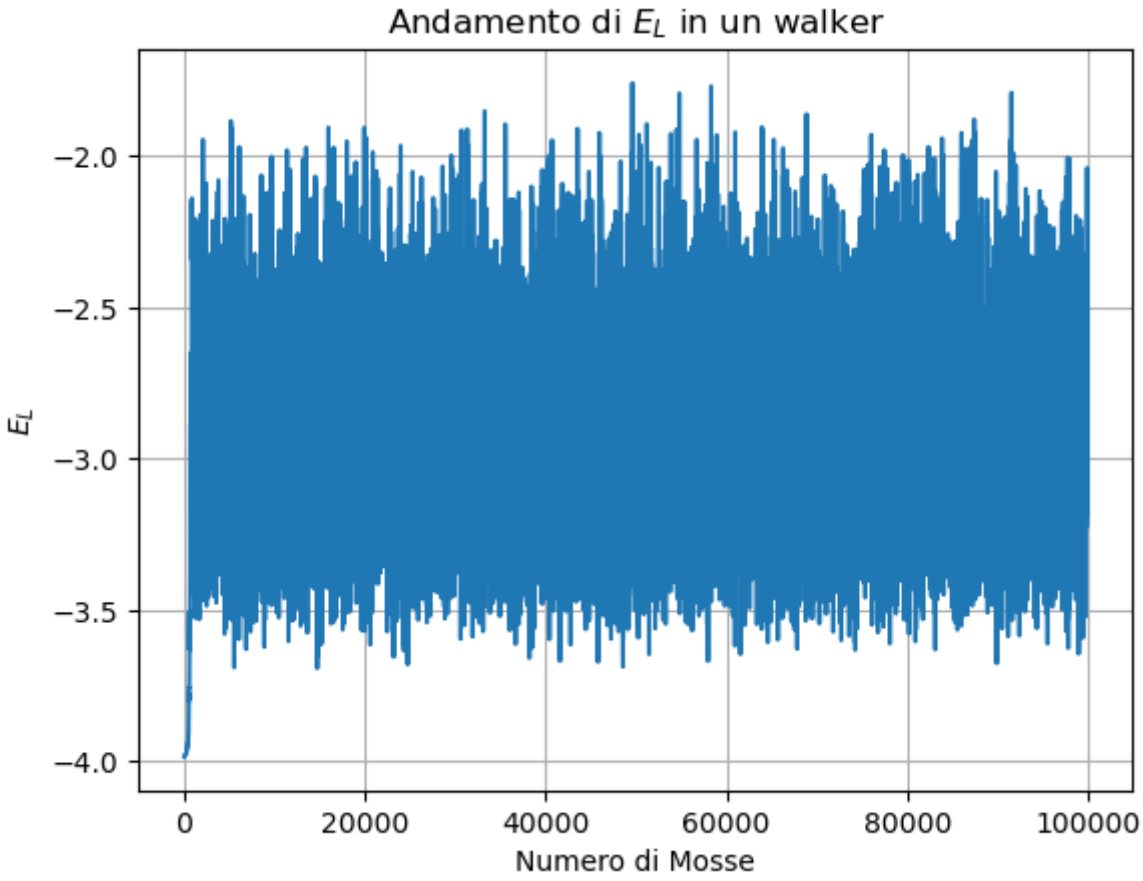
Out[337]: []

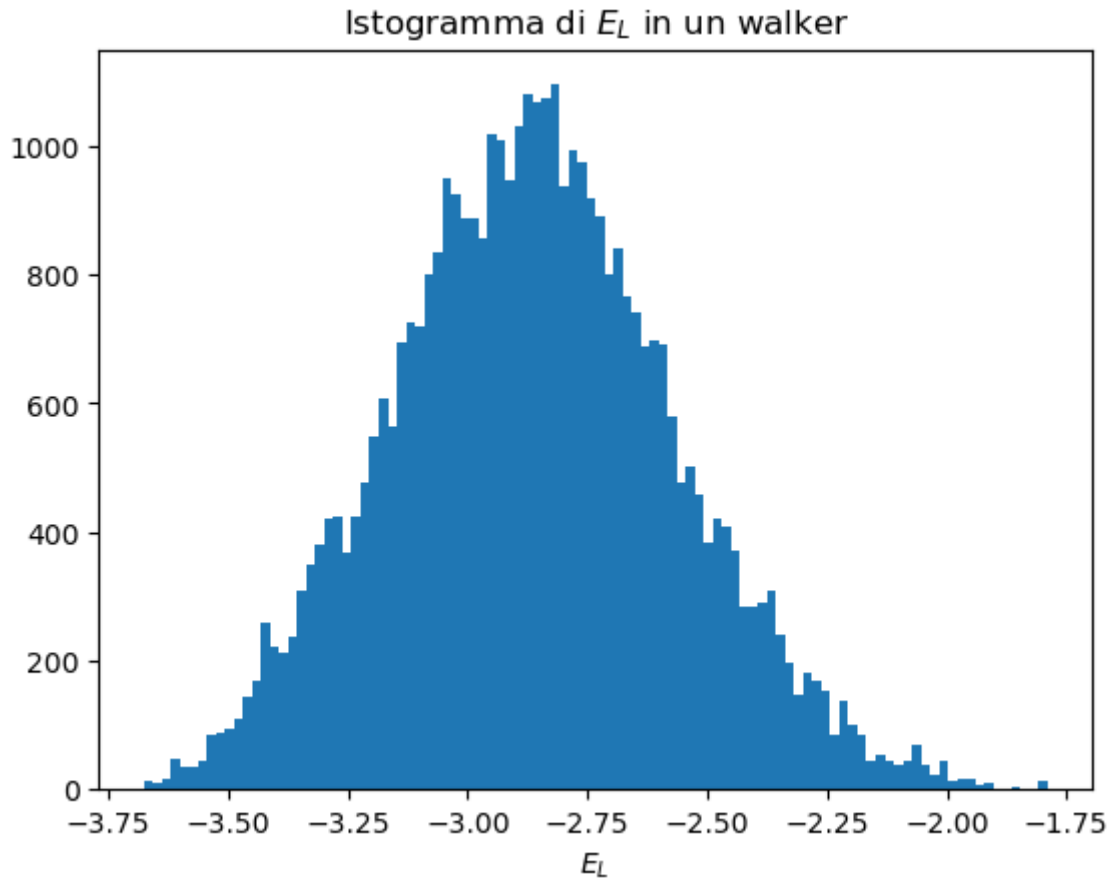


In [429…

```
with open('./TaskB/local_energy51.txt', 'rb') as file:
    coeff = np.load(file)
    energyList = np.load(file)
plt.plot(mosse,energyList)
plt.grid()
plt.xlabel("Numero di Mosse")
plt.ylabel("$E_L$")
plt.title("Andamento di $E_L$ in un walker")
plt.show()

#istogramma
plt.hist(energyList[60000//passo:],bins=100)
plt.title("Istogramma di $E_L$ in un walker")
plt.xlabel("$E_L$")
plt.show()
```



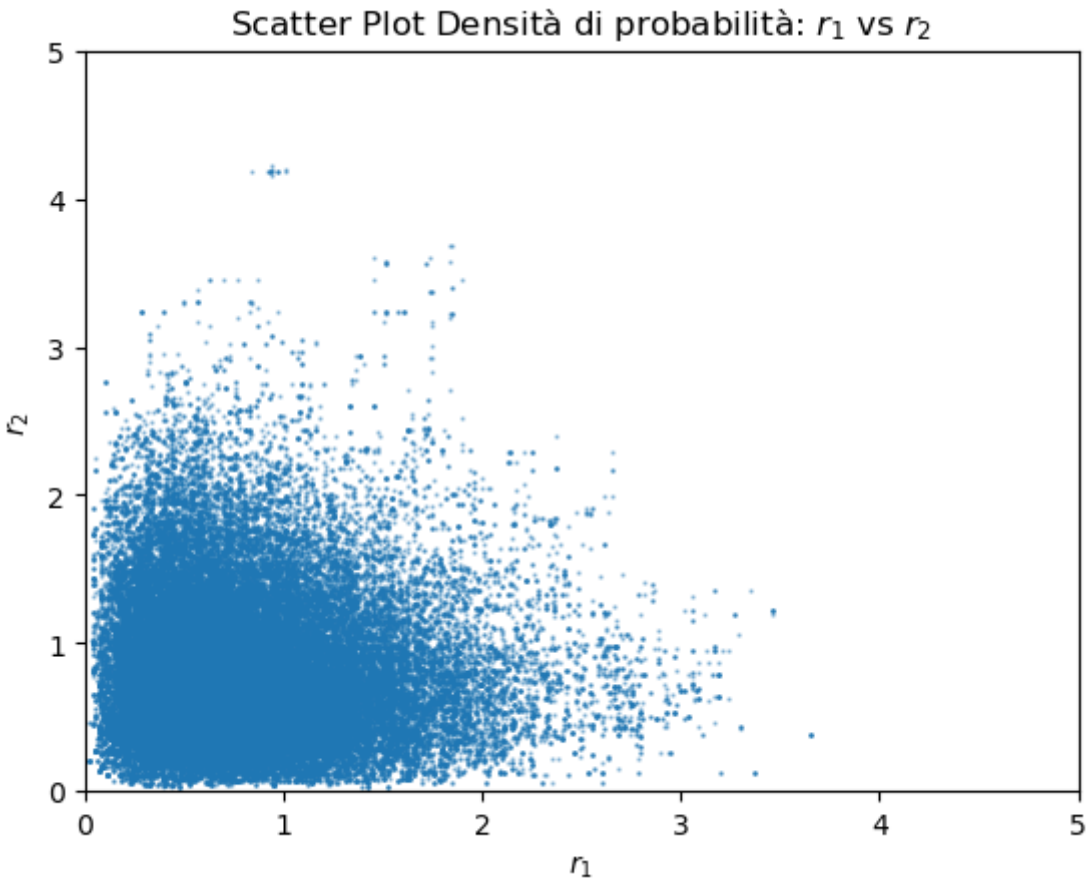


In [338…

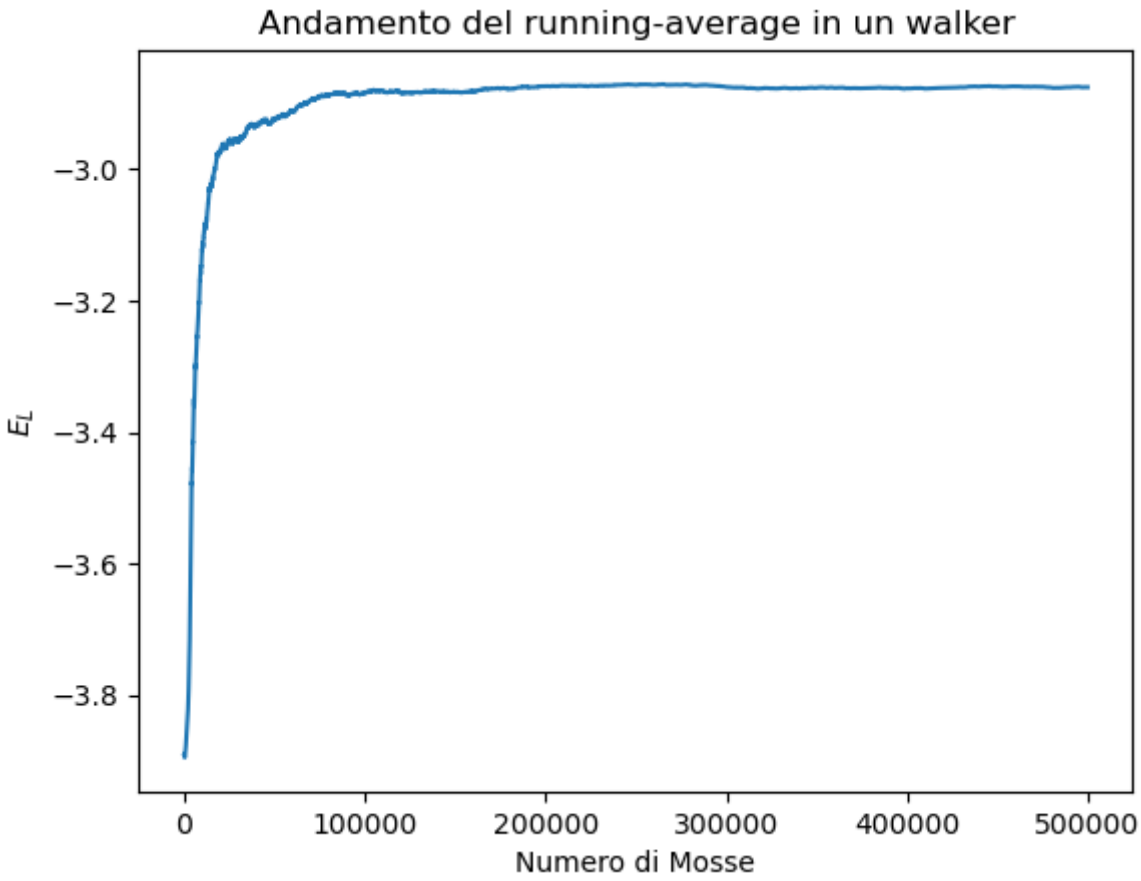
```
with open('./TaskB/r151.txt', 'rb') as file:
    coeff = np.load(file)
    r1_vector = np.load(file)
    r1 = np.linalg.norm(r1_vector[1000:],axis=1)
with open('./TaskB/r251.txt', 'rb') as file:
    coeff = np.load(file)
```

```
    r2_vector = np.load(file)
    r2 = np.linalg.norm(r2_vector[1000:],axis=1)
plt.scatter(r1,r2, s=0.5, alpha=0.4)
plt.xlabel("$r_1$")
plt.ylabel("$r_2$")
plt.xlim(0,5)
plt.ylim(0,5)
plt.title("Scatter Plot Densità di probabilità: $r_1$ vs $r_2$")
plt.plot()
```

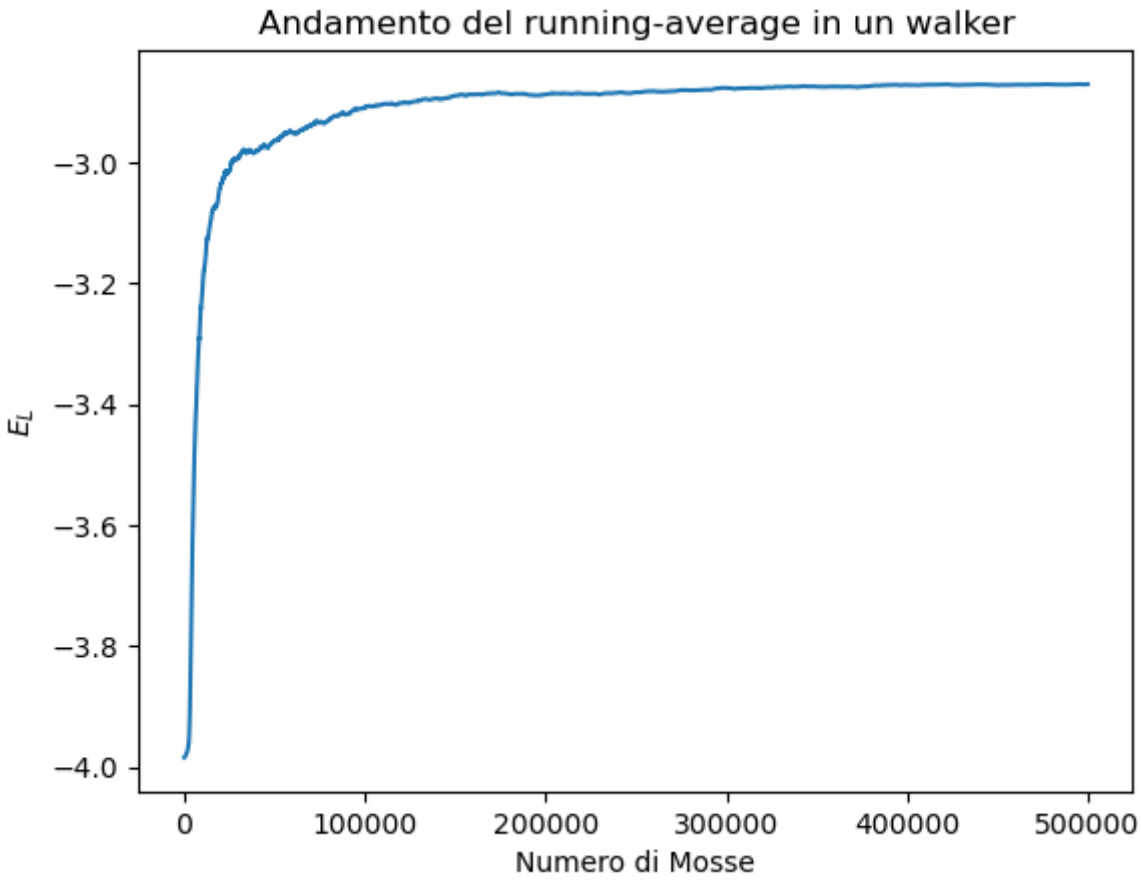
Out[338]:
```
[]
```



In [87]:
```
passo = 5
with open('./TaskB/local_energy1.txt', 'rb') as file:
    coeff = np.load(file)
    energyList = np.load(file)
    M = len(energyList)
run_avg=np.zeros(M)
mosse = np.arange(0,M*passo,1*passo)
for t in range(M):
    run_avg[t]=np.mean(energyList[:t])
plt.plot(mosse,run_avg)
plt.xlabel("Numero di Mosse")
plt.ylabel("$E_L$")
plt.title("Andamento del running-average in un walker")
plt.show()
```



In [88]:
```
passo = 5
with open('./TaskB/local_energy51.txt', 'rb') as file:
    coeff = np.load(file)
    energyList = np.load(file)
    M = len(energyList)
run_avg=np.zeros(M)
mosse = np.arange(0,M*passo,1*passo)
for t in range(M):
    run_avg[t]=np.mean(energyList[:t])
plt.plot(mosse,run_avg)
plt.xlabel("Numero di Mosse")
plt.ylabel("$E_L$")
plt.title("Andamento del running-average in un walker")
plt.show()
```



Conclusioni

E' possibile notare come, in entrambe le task, l'energia trovata non corrisponde completamente al valore misurato analiticamente pari a -2.904 anche se, per quanto riguarda il task B, grazie all'ottimizzazione utilizzata (interpolazione su griglia uniforme) ci si avvicina molto trovando un'energia pari a -2.880 e questo rispecchia i risultati attesi in quanto ci si aspettava un valore meno preciso dalla TaskA (abbiamo ottenuto infatti -2.863) dato che la funzione era relativa ad un sistema di elettroni non interagenti.

Grazie al confronto tra i grafici relativi alla running average dell'energia all'interno di un walker possiamo vedere come, nella prima Task, il numero di mosse per arrivare alla condizione stazionaria è di molto inferiore rispetto al quelle della seconda Task.

Per quanto riguarda il valore di β atteso esso risulta ragionevole in quanto con β piccoli il sistema è energicamente più favorevole, sono preferite delle configurazioni in cui i due elettroni sono distanti tra di loro.

In [ ]: