Metodo dello Spazio di Krylov e Applicazione al Modello di Ising Quantistico 1D

Manuel Strazzullo
Dipartimento di Fisica, Università di Pisa

5 gennaio 2026

Sommario

In questo lavoro vengono introdotti i metodi algoritmici basati sulla costruzione dello spazio di Krylov, necessari per la trattazione computazionale dei sistemi quantistici interagenti a molti corpi; viene fornita un'implementazione tramite codice Python e ne viene testata l'efficienza. Un'implementazione più complessa di questi metodi in Fortran, basata sull'algoritmo Davidson, viene utilizzata per verificare alcune proprietà dello stato fondamentale per il modello di Ising quantistico in una dimensione: in particolare vengono analizzate le spaziature dei primi livelli energetici e l'andamento del parametro d'ordine vicino alla transizione di fase.

1 Introduzione alla diagonalizzazione esatta

Motivazione

In fisica statistica e dello stato condensato capita spesso di doversi interfacciare con sistemi che, data la loro natura quantistica, sono esponenzialmente complessi da simulare numericamente. Prendiamo come esempio un sistema di N particelle di spin $1/2$ su un reticolo unidimensionale, in cui ogni particella occupa un sito e ha un'orientazione di spin $|\uparrow\rangle$ o $|\downarrow\rangle$; la dimensione dello spazio di Hilbert del problema (e quindi la dimensione della funzione d'onda) sarà

$$Dim(\mathcal{H}) = 2^N$$

mentre una eventuale rappresentazione matriciale degli operatori (l'esempio rilevante è l'operatore Hamiltoniano $\hat{H}$) avrà dimensione

$$Dim(\hat{H}) = 2^N \times 2^N.$$

e sarà spesso una qualche matrice "sparsa" in cui molti elementi sono nulli, a causa della struttura tensoriale dello spazio di Hilbert del problema. Spesso in problemi di questo tipo siamo interessati allo spettro e alle funzioni d'onda dei primi stati dell'Hamiltoniana, che vorremmo quindi portare in forma matriciale e diagonalizzare, processo che richiede tempi e risorse computazionali non indifferenti già per $N = O(10)$. Verifichiamo come esempio i tempi di calcolo che il computer impiega, usando il pacchetto *scipy.linalg*, a diagonalizzare matrici casuali con $N = O(10)$ (tralasciando di trovare gli autovettori) con metodi come la decomposizione di Schur o il $divide - and - conquer$: se per $N = 10$ il tempo necessario τ è $\sim 0.5s$, con $N = 13$ si arriva già a $\tau \sim 130s$. Vorremmo allora trovare dei metodi computazionali che velocizzino il processo di diagonalizzazione, magari usando il fatto che le matrici sono sparse e non necessitano di essere immagazzinate nella loro interezza.

Teoria

Il metodo dello spazio di Krylov è una costruzione algoritmica che permette la diagonalizzazione approssimata di matrici sparse di grandi dimensioni. Lo spazio di Krylov di dimensione D, per la matrice che vogliamo diagonalizzare $\hat{H}$, è definito come

$$\mathcal{K}^D(|\phi_0\rangle) = \text{span}(|\phi_0\rangle, \hat{H}|\phi_0\rangle, ..., \hat{H}^{D-1}|\phi_0\rangle) \tag{1}$$

ed è costruito applicando iterativamente l'operatore $\hat{H}$ a uno stato casuale e normalizzato nello spazio di Hilbert del problema, ottenendo stati che vengono ortonormalizzati rispetto a tutti i precedenti. Si trova che $\hat{H}$ può essere espresso come

$$\hat{H}|\phi_n\rangle = b_n|\phi_{n-1}\rangle + a_n|\phi_n\rangle + b_{n+1}|\phi_{n+1}\rangle \tag{2}$$

ovvero l'Hamiltoniana è tridiagonale sullo spazio di Krylov.

$$\hat{H} = \begin{bmatrix} a_0 & b_1 & 0 & \cdots & 0 \\ b_1 & a_1 & b_2 & \cdots & 0 \\ 0 & b_2 & a_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_D \end{bmatrix}$$

Ciò che è possibile fare a questo punto è diagonalizzare la matrice su questo spazio per ottenere un'approssimazione dei primi autovalori e autovettori; ciò deriva dal fatto che la moltiplicazione ripetuta per la matrice $\hat{H}$ aumenta il peso degli autovalori più grandi (in modulo). L'algoritmo Lanczos utilizza una logica simile, diagonalizzando una matrice su uno spazio a dimensione fissata generato da $(|\phi\rangle, \hat{H}|\phi\rangle)$ e iterando la procedura a partire dall'autovettore trovato corrispondente all'autovalore minore.

Nell'analisi numerica successiva useremo il metodo di Jacobi-Davidson, che si differenzia dal metodo di Krylov nella costruzione del sottospazio approssimativo (che viene costruito a partire da vettori ortogonali agli autovettori approssimati), ma utilizza comunque la stessa logica; per costruire il sottospazio rilevante non è necessario immagazzinare la matrice per intero, ma solo conoscere come essa agisce sui vettori di partenza, risparmiando risorse computazionali ingenti.

Implementazione e verifica

Nonostante il fatto che non useremo il metodo di Krylov, ma uno leggermente più sofisticato, vogliamo comunque verificare l'accuratezza e le capacità del metodo "base" tramite una nostra implementazione.

Il codice Python in appendice implementa la costruzione algoritmica dello spazio di Krylov per una matrice IN, che riempiamo con numeri interi casuali nell'intervallo $[-30, 30]$; il fatto che la matrice non sia sparsa non altera il funzionamento dell'algoritmo, che comunque è preferibile usare nel caso in cui immagazzinare l'intera matrice sia molto più dispendioso che calcolare $\hat{H}|\phi\rangle$ (appunto nel caso di matrici sparse). In ogni caso, il corretto funzionamento dell'algoritmo in questa situazione garantisce il funzionamento atteso nel caso più banale delle matrici sparse. Le matrici IN sono poi rappresentate nella base di Krylov, ottenendo matrici tridiagonali T; infine calcoliamo grazie a *scipy.linalg.eigvalsh* gli autovalori esatti (dalla matrice di partenza IN) e gli autovalori approssimati (dalla matrice tridiagonale T). Gli errori assoluti per diverse dimensioni dello spazio di Krylov sono riportati in Fig. 1.

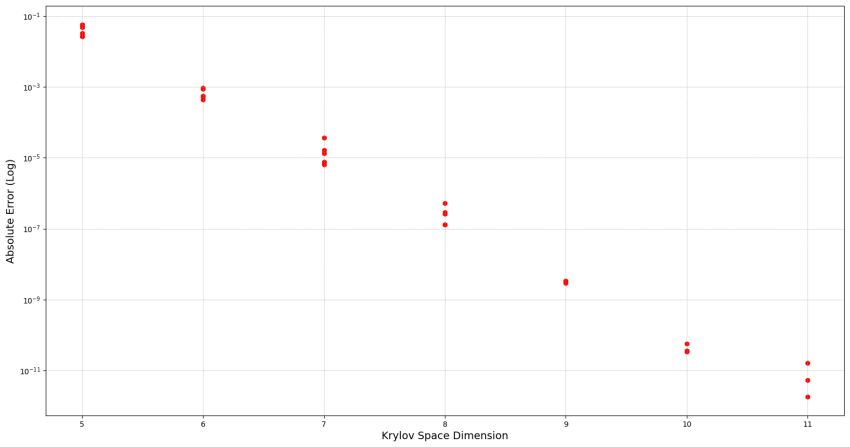

Figura 1: Errore assoluto sullo stato fondamentale di matrici casuali al variare della dimensione dello spazio di Krylov, calcolato con il codice in appendice; l'andamento medio è chiaramente esponenziale.

Per ogni dimensione D dello spazio di Krylov sono state effettuate da due a quattro prove, la cui media ha andamento esponenzialmente decrescente, con una precisione eccellente già per dimensione dello spazio di Krylov $D = 10$. L'autovalore minimo, dato l'intervallo scelto sopra per la generazione delle matrici, vale in media $\approx -4160 \pm 20$ e il tempo di calcolo totale per ciascuno dei processi è $\sim 20s$, con il $\sim 97\%$ ($\sim 19.5s$) del tempo impiegato per diagonalizzare la matrice iniziale contro $\sim 0.5s$ per diagonalizzare la matrice approssimata.

Notiamo il fatto che le implementazioni usate in genere per l'analisi numerica sono ulteriormente ottimizzate dagli algoritmi scelti per diagonalizzare la matrice tridiagonale, che sfruttano ulteriori simmetrie delle matrici da

diagonalizzare; tuttavia la nostra verifica mostra l'accuratezza del metodo di Krylov, che riguarda la costruzione del sottospazio approssimato più che la diagonalizzazione delle matrici in sè.

2 Analisi numerica della catena di spin quantistica

Tramite l'algoritmo Davidson, vogliamo ora effettuare alcune misure sul modello di Ising 1D, in cui a ogni sito di una catena è associata una particella quantistica di spin 1/2; l'Hamiltoniana del modello è

$$\hat{H}(J,h,g) = -J \sum_{<i,j>} \hat{\sigma}_i^z \hat{\sigma}_j^z - h \sum_i \hat{\sigma}_i^z - g \sum_i \hat{\sigma}_i^x \tag{3}$$

in cui $\hat{\sigma}_i^{k=x,y,z}$ è la matrice di Pauli corrispondente al sito i-esimo, $< i,j >$ indica la somma sui primi vicini e J, g, h quantificano rispettivamente l'accoppiamento tra due siti (J) e l'interazione con campi esterni con direzione ortogonale (g) e parallela (h) alla direzione di accoppiamento degli spin (in questo caso fissata lungo $\hat{z}$).

Come verifica dell'utilità dei metodi di diagonalizzazione esposti sopra, riportiamo i tempi di calcolo necessari alla diagonalizzazione numerica dell'Hamiltoniana tramite metodo di Davidson per diverse lunghezze L del sistema, avendo imposto $h = 0$, $g = 0.75$ e condizioni al bordo periodiche; l'andamento esponenziale è riportato in Fig. 2.

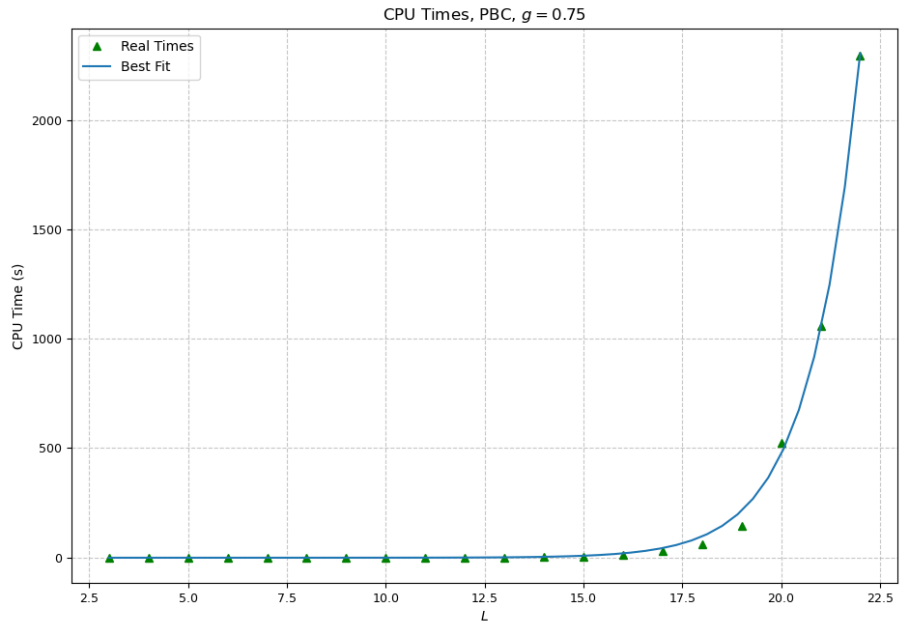

Figura 2: Tempi di calcolo al variare della lunghezza L della catena; l'andamento è $\sim e^{\frac{L}{R}}$, con $1/R \simeq 8$.

Assumendo che l'andamento temporale esponenziale sia lo stesso che presentano gli algoritmi di diagonalizzazione completa, possiamo stimare che per $N = 20$ i tempi di calcolo sarebbero stati $\sim 500s \cdot 40 \approx 33$ min per una sola misura, il che mostra la comodità degli algoritmi basati sul metodo di Krylov e simili.

Spaziature dei primi livelli energetici

In questa sottosezione poniamo $h = 0$, ovvero campo longitudinale nullo.

Vogliamo a questo punto usare i metodi di diagonalizzazione introdotti in precedenza per verificare alcune proprietà dei primi livelli energetici del modello di Ising. Avendo posto il campo longitudinale uguale a zero, il modello è esattamente risolvibile tramite la trasformazione di Jordan-Wigner, che permette di mappare il modello in un modello fermionico con Hamiltoniana

$$\hat{H} = \frac{1}{2} \sum_j \epsilon_j \hat{\gamma}_j^\dagger \hat{\gamma}_j + \text{cost.} \tag{4}$$

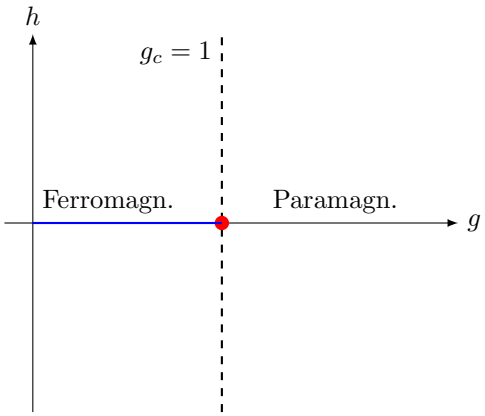

Figura 3: Diagramma di fase per il modello di Ising quantistico; avendo posto $h = 0$, il passaggio per il punto $g_c = 1$ implica una transizione di fase, che si manifesta con diversi andamenti delle spaziature energetiche nelle due regioni $g \gtrless 1$.

che decompone il sistema in modi normali $\hat{\gamma}_j^\dagger$, i quali si comportano come fermioni (quasiparticelle di Bogoliubov). Da questa Hamiltoniana è possibile ricavare una relazione di dispersione per le ϵ_j

$$\epsilon_j = \sqrt{1 + g^2 - 2g\cos\left(\frac{2\pi j}{L}\right)} \tag{5}$$

che racchiude l'andamento degli autovalori del sistema iniziale.

Vogliamo osservare l'andamento delle spaziature energetiche

$$\Delta_0 = E_1 - E_0 \qquad \Delta_1 = E_2 - E_0$$

in funzione del campo trasverso g e al crescere della dimensione L, per verificare che il sistema approcci l'andamento noto per il limite termodinamico $L \to \infty$.

Riportiamo in Fig. 3 il diagramma di fase del modello, che fornisce una spiegazione quantitativa del fatto che nelle regioni $g \gtrless 1$ si osservano andamenti diversi per le spaziature: per $g < 1$ siamo nella fase ferromagnetica (ordinata), mentre per $g > 1$ siamo nella fase paramagnetica (disordinata).

La scelta delle condizioni al bordo (periodiche, PBC, o aperte, OBC) diventa rilevante a questo punto, poichè le diverse simmetrie del sistema risultano in andamenti diversi per le energie nel limite termodinamico; in seguito (Fig. 4 per OBC e Fig. 5 per PBC) riportiamo i risultati per entrambe le scelte.

Partiamo dal commentare le spaziature Δ_0, che nella regione $g < 1$ assumono andamenti simili per entrambe le condizioni al bordo

$$\Delta_{0,PBC} \sim \frac{g^L}{\sqrt{L}}\sqrt{1-g^2} \qquad \Delta_{0,OBC} \sim g^L(1-g^2) \tag{6}$$

ovvero le spaziature sono esponenzialmente soppresse in L quando $g \to 0$; per $g > 1$ si può ricavare dall'Hamiltoniana in Eq. 4 che, per entrambe le scelte di condizioni al bordo,

$$\Delta_0 \approx 2(g-1). \tag{7}$$

E' anche noto il fatto che, sempre per per $g > 1$, anche la spaziatura Δ_1 segue lo stesso andamento asintotico (sia con OBC che con PBC), poichè vale la stessa relazione di dispersione.

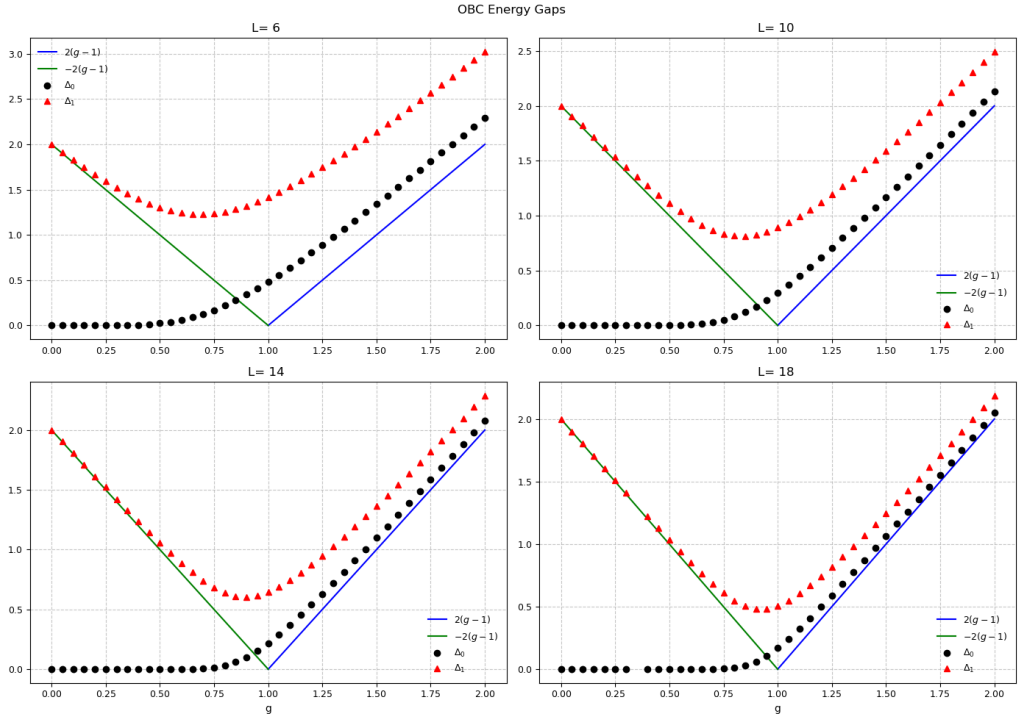

Figura 4: Prime spaziature energetiche $\Delta_0 = E_1 - E_0$, $\Delta_1 = E_2 - E_0$, per varie lunghezze L della catena con condizioni al bordo aperte, al variare del campo trasverso g; i rispettivi andamenti nel limite termodinamico (notare che Δ_0 e Δ_1 hanno lo stesso asintoto per $g > 1$) sono riportati in legenda.

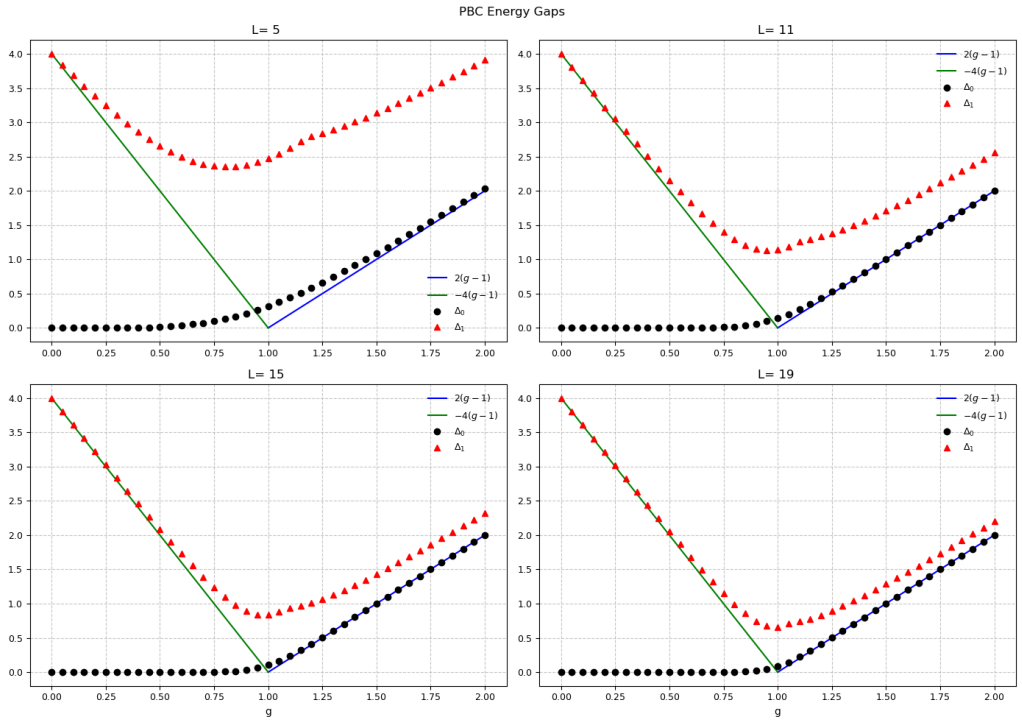

Figura 5: Prime spaziature energetiche $\Delta_0 = E_1 - E_0$, $\Delta_1 = E_2 - E_0$, per varie lunghezze L della catena con condizioni al bordo periodiche, al variare del campo trasverso g; come nel caso OBC, le spaziature tendono al valore atteso per $L \to \infty$.

Per quanto riguarda Δ_1 nella regione ferromagnetica, a causa delle diverse simmetrie l'andamento va differenziato tra OBC e PBC, trovando

$$\Delta_{1,PBC} \approx -4(g-1) \qquad \Delta_{1,OBC} \approx -2(g-1). \tag{8}$$

E' visibile a questo punto come in tutti i casi, per $g = 1$ (punto critico), le spaziature tendano a zero nel limite termodinamico. Le energie dei primi stati dello spettro sono state calcolate con un codice Fortran (riportato in appendice) che fa uso dell'algoritmo Davidson e che, fissata la lunghezza L e il valore di g, consente di trovare le informazioni rilevanti utilizzando risorse computazionali ridotte, a patto di saper codificare $\hat{H}\,|\psi_{in}\rangle = |\psi_{out}\rangle$; in appendice è inoltre riportato un codice Bash utilizzato per automatizzare alcune misure.

Gli andamenti osservati in Fig 4. e 5. mostrano come, al crescere di L, le spaziature tendano ai risultati attesi nel limite termodinamico.

Un'ultima verifica che possiamo fare sulle spaziature riguarda ancora Δ_0: in Fig. 6 riportiamo Δ_0, al variare di L e per valori di g fissati, in modo da esplorare le fasi ordinate, disordinate e il punto critico $g = 1$.

Vorremmo osservare il decadimento esponenziale in L delle spaziature, ovvero

$$\Delta_{0,g<1} \sim e^{-L/\xi(g)} \qquad \Delta_{0,g>1} \sim 2(g-1) + e^{-L/\xi(g)} \tag{9}$$

dove $\xi(g)$ è la lunghezza di correlazione del sistema, per la quale vale

$$\xi(g) \sim |g - g_c|^{-\nu} \tag{10}$$

e che rimane finita lontano dal punto critico $g = g_c$. Per $g = g_c = 1$ possiamo usare un *guess*

$$\Delta_0(g) \sim L^{-z} \sim \xi(g) \sim |g - g_c|^{-\nu} \tag{11}$$

per cui dovrebbe valere, mappando il modello quantistico 1D sul modello classico 2D, $z = \nu = 1$. Un fit per la legge di potenza su $L \geq 13$, al valore critico $g = 1$, restituisce $z = 1.004(2)$ in accordo con la nostra ipotesi; la precisione del fit è deducibile dal fatto che esso rimane stabile non considerando i punti $L = 14, 15, 16$.

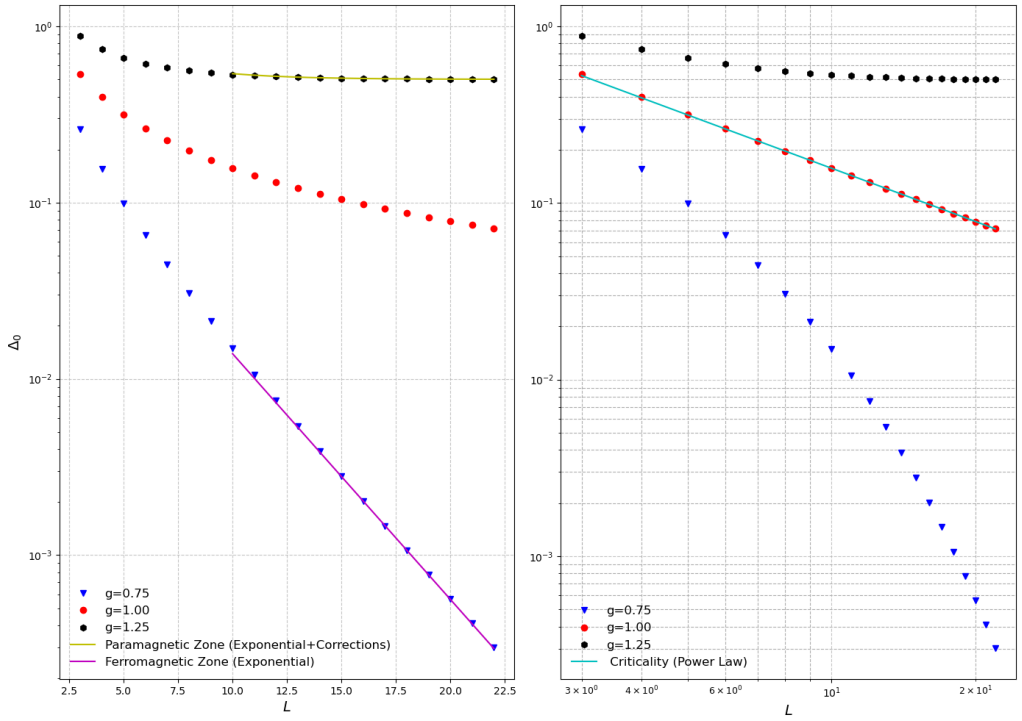

Figura 6: Prima spaziatura Δ_0 per valori del campo trasverso g nelle tre zone del diagramma di fase, in cui a sinistra la scala è *lin-log*, in cui sono chiaramente visibili gli esponenziali, mentre a destra è *log-log*, in cui è chiaramente visibile la legge di potenza.; il fit al punto critico $g = 1$ restituisce esponente critico $z = 1.004(2)$, mentre lontano dal punto critico l'andamento approssima quello aspettato nel limite termodinamico.

In questa sottosezione abbiamo dimostrato l'affidabilità e la precisione dei metodi basati sullo spazio di Krylov, almeno per quanto riguarda la parte di calcolo degli autovalori.

Parametro d'ordine Ψ, pseudo-parametro d'ordine $\bar{\Psi}$

Vogliamo ora verificare l'affidabilità dell'algoritmo per quanto riguarda la ricerca degli autovettori dei primi stati dello spettro; come già fatto sopra, useremo un aspetto del modello di Ising per testare i nostri metodi.

Nel modello di Ising la fase ferromagnetica è caratterizzata dalla rottura della simmetria $\mathbb{Z}_2$ di inversione (lungo la direzione di accoppiamento $\hat{z}$) di tutti gli spin; la rottura della simmetria può essere quantificata attraverso un parametro d'ordine Ψ. Il parametro d'ordine, per una catena composta da siti che indichiamo con l'indice j, è dato dalla magnetizzazione longitudinale calcolata sullo stato fondamentale Ψ_0

$$\Psi \equiv \langle \Psi_0 | \hat{m}_z | \Psi_0 \rangle = \frac{1}{L} \sum_j \langle \Psi_0 | \hat{\sigma}_z^j | \Psi_0 \rangle \tag{12}$$

che, nel limite termodinamico e nella fase $g < 1$, assume valori non nulli a causa della rottura della simmetria $\mathbb{Z}_2$.

Solitamente, a causa delle possibili simmetrie (in particolare della simmetria generale dell'Hamiltoniana $\mathbb{Z}_2$ in assenza di campo trasverso), sono presenti delle sottigliezze nella scelta e nel calcolo del parametro d'ordine; ma, ponendo a questo punto $g = 1$, la magnetizzazione longitudinale definita come in Eq. 12 quantifica correttamente la rottura di simmetria del sistema. Riferendosi al diagramma di fase in Fig. 3, si ha che il sistema attraversa la linea di transizione per $h \to 0$, con andamento teorico atteso

$$\Psi(g = 1) \sim h^{1/\delta}$$

dove $\delta = 15$ è un esponente critico della classe di universalità Ising classico 2D .

In Fig. 7 è riportato il parametro d'ordine, calcolato come in Eq. 12, al variare del campo trasverso h, sullo stato fondamentale trovato grazie al codice che implementa il metodo di Davidson; i dati mostrano che al crescere di L il sistema si avvicina all'andamento teorico nel limite termodinamico.

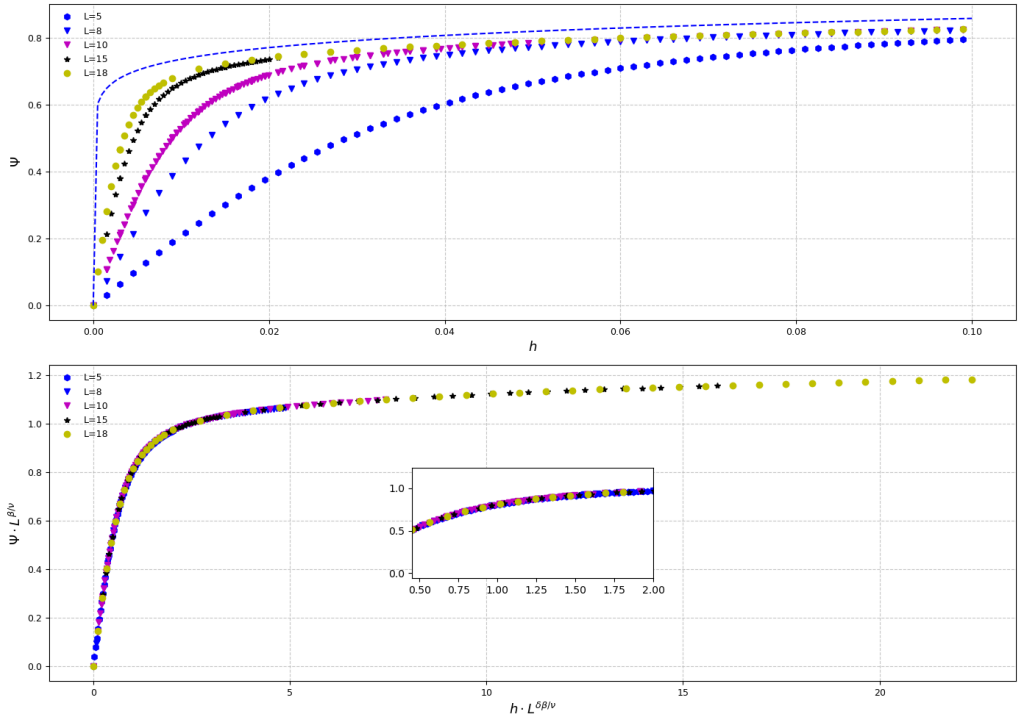

Figura 7: Sopra, parametro d'ordine Ψ per il valore critico $g = 1$, al variare del campo longitudinale h, per diversi valori di L e andamento teorico (linea blu tratteggiata); in basso, FSS con gli esponenti critici noti per classe di universalità del modello di Ising classico 2D $\beta = 1/8$, $\nu = 1$, $\delta = 15$.

Come ulteriore verifica, osserviamo il *finite size scaling* del parametro d'ordine con gli esponenti critici noti per il modello: dalla teoria sappiamo che, vicino al punto critico,

$$\Psi \approx L^{-\beta/\nu} \Psi_u(hL^{\delta\beta/\nu}) \tag{13}$$

a meno di correzioni dipendenti dalla dimensione del sistema e dove Ψ_u è una funzione nota. Riscalando opportunamente i dati per diversi L, dovremmo osservare un andamento universale del parametro d'ordine ('collasso' dei punti in un'unica curva); il collasso è riportato in Fig. 7 e le correzioni non sono visibili a occhio.

Poniamo ora $h = 0$ (campo longitudinale nullo), andando a ristabilire la simmetria $\mathbb{Z}_2$ del modello. Come riportato sopra, il parametro d'ordine Ψ definito come in Eq. 12 e calcolato sullo stato fondamentale è identicamente nullo, ma definendo uno pseudo-parametro d'ordine

$$\bar{\Psi} = \sum_k |a_k|^2 | \langle k | \hat{m}_z | k \rangle | \tag{14}$$

calcolato sullo stato fondamentale decomposto nella base computazionale come $|\Psi_0\rangle = \sum_k a_k |k\rangle$, possiamo ottenere l'andamento corretto per la magnetizzazione al variare di g, almeno in prossimità del punto critico $g_c = 1$. In Fig. 8 è riportato l'andamento di $\bar{\Psi}$ al variare di g, fissato $h = 0$ (muovendosi quindi l'asse delle ascisse nel diagramma in Fig. 3), e il FSS con gli esponenti critici noti per il modello di Ising classico 2D; tutto è stato calcolato con lo stato fondamentale trovato con l'algoritmo Davidson. L'andamento atteso, nel limite termodinamico e nelle vicinanze del punto critico, è

$$\Psi(g) \sim (g_c - g)^{\beta} \qquad g < g_c$$

e $\Psi(g) = 0$ per $g > g_c$.

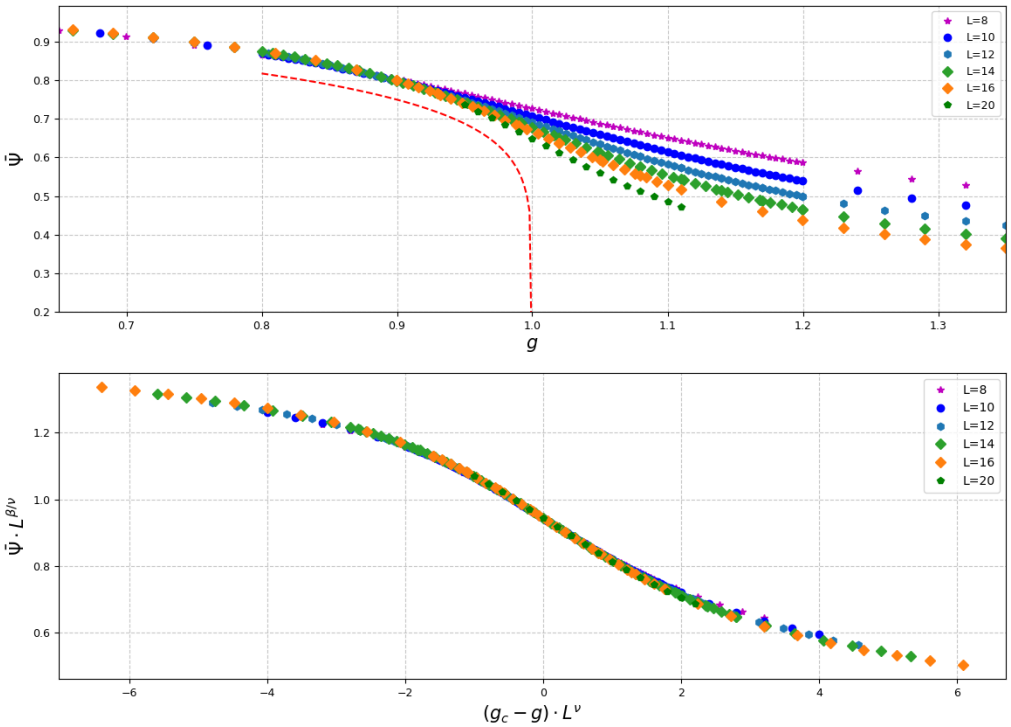

Figura 8: Sopra, parametro d'ordine $\bar{\Psi}$ per il valore $h = 0$, al variare del campo trasverso g, per diversi valori di L e andamento teorico (linea rossa tratteggiata); in basso, FSS con gli esponenti critici noti per classe di universalità del modello di Ising classico 2D. Il collasso dei punti nel FSS è ottimo, e l'andamento atteso di parametro nullo nella regione paramagnetica e legge di potenza approcciando il punto critico da sinistra è visibile all'aumentare della taglia del sistemmicra.

Abbiamo dunque verificato che il metodo Davidson restituisce la funzione d'onda corretta per lo stato fondamentale, dimostrandosi in generale un ottimo strumento di calcolo nella diagonalizzazione esatta dei sistemi quantistici a molti corpi.

Appendice: codici

```
1
2  import scipy as sp
3  import numpy as np
4
5  dim_hilbert = 2**10
6  dim_krylov = 80
7
8  def lanczos(IN, v, k):
9      """
10     Algoritmo Lanczos-Like (per matrici simmetriche)
11
```

```
    Parametri
    ----------
    IN : (n, n) matrice reale simmetrica

    v : (n,) vettore di partenza

    k : intero, dimensione dello spazio di Krylov

    Ritorna
    -------
    B : (n, k) base ortonormale

    T : (k, k) matrice tridiagonale che approssima la matrice di partenza

    """
    n = IN.shape[0]
    B = np.zeros((n, k), dtype=IN.dtype)
    T = np.zeros((k, k), dtype=IN.dtype)

    v = v / np.linalg.norm(v)
    B[:, 0] = v
    w = IN @ v
    a = np.vdot(v, w)
    T[0, 0] = a
    w = w - a * v
    b = np.linalg.norm(w)

    for j in range(1, k):
        if b < 1e-12:            #se converge prima si ferma
            return B[:, :j], T[:j, :j]
        v_next = w / b
        B[:, j] = v_next
        w = IN @ v_next - b * B[:, j - 1]
        a = np.vdot(v_next, w)
        w = w - a * v_next
        T[j, j] = a
        T[j, j - 1] = T[j - 1, j] = b
        b = np.linalg.norm(w)

    return B, T


#inizializzo la matrice (simmetrica) e il vettore iniziale (la funzione lo normalizzao)
start_mat = np.random.randint(-3,3, (dim_hilbert, dim_hilbert))
start_mat = (start_mat + start_mat.T)
start_mat = start_mat.astype(float)
start_vec = np.random.randint(-3,3, size=dim_hilbert)

##verifica (conviene solo in dimensioni basse)
#print("Starting matrix:\n", start_mat)
#print("Starting vector:\n", start_vec)

krylov_basis, tridiag = lanczos(start_mat, start_vec, dim_krylov)

##verifica (conviene solo in dimensioni basse)
#print("Orthonormal Krylov basis B:\n", krylov_basis)
#print("\nTridiagonal matrix T:\n", tridiag)

#autovalori esatti e approssimati
approx_eigenvals = np.sort(sp.linalg.eigvalsh(tridiag))
exact_eigenvals = np.sort(sp.linalg.eigvalsh(start_mat))
diff = exact_eigenvals[0]-approx_eigenvals[0]
print("\nExact ground state:\n", exact_eigenvals[0])
print("\nApproximate ground state:\n", approx_eigenvals[0])
print("\nError:\n", diff)
print("\nKrylov space dimension:\n", dim_krylov)
```

Listing 1: Codice Python per l'implementazione del metodo di Krylov e verifica della sua efficienza.

```
module parameters      !!non posso usare contains(subroutines), problemi nel compilare
  logical  :: periodic
  integer  :: lenght, dim_hilbert
  integer, dimension(:,:), allocatable :: spin_states
  real(8) :: g, lambda, lambdaAmul
end module parameters

program energy_gaps      !!programma per l'analisi dei gap energetici in ising quantistico 1D
```

```
  use parameters
  implicit none

  integer :: i, j, itemp
  integer :: kmax          !!parametro davidson, autovalore massimo che la routine trova
  real(8) :: gap0, gap1, gap2, cputime
  logical :: save_time, lenght_is_fixed
  real(8), dimension(:), allocatable :: en
  double complex, dimension(:), allocatable :: ground_state

    lenght_is_fixed=.false.              !!se true devo mandare in input il campo, altrimenti
    la lunghezza
  save_time = .false.          !!printa il tempo di calcolo o no
  periodic = .false.             !!condizioni al bordo
  lambda = 0.d0              !!campo longitudinale (0 per questo progetto)

  if (lenght_is_fixed) then            !!inizializzo i parameteri a seconda dello scopo
      lenght = 10                !!lunghezza catena (costante)
      read *, g                !!campo trasverso (da mandare con script shell)
  else
      read *, lenght                   !!lunghezza catena (da mandare con script shell)
    g = 1.25d0                   !!campo trasverso (costante)
    end if

  dim_hilbert = 2**lenght  !!inizializzo i registri di spin
  allocate(spin_states(dim_hilbert, lenght))
  spin_states = 0

  do i=1,dim_hilbert       !!codifica dei registri in binario: in ordine crescente
     itemp = i-1         !!con codifica convenzionale da dx
     do j=1,lenght
       spin_states(i,lenght+1-j)=mod(itemp, 2)
       itemp = itemp/2
     end do
  !print*, i,  spin_states(i, :)      !!verifica stati
  end do

  kmax = 4
  allocate(ground_state(dim_hilbert))
  allocate(en(kmax))

  call davidson(dim_hilbert, kmax, en, ground_state)    !!trovo il ground state (funzione d'
    onda)
  if (lenght_is_fixed) then                             !!e i primi kmax livelli energetici
        print *, g, en(1:kmax)
  else
      print *, L, en(1:kmax)
  end if

  if (save_time) then
      call cpu_time(cputime)
        print *,'Total CPU time:  ', cputime
    end if

    deallocate(spin_states, ground_state, en)

end program energy_gaps

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine davidson(n, Kmax, Eigenvalue, EigenVector) !!non toccare
  use parameters

  implicit none
  logical           UseGuess,Wanted
  integer           kmax,jmax,jmin,maxstep,method,m,l,maxnmv,order,testspace,j,lwork,istate,ii
    ,n,kmaxuser
  real(8)           tol,lock,targetEn,Norm,emin,etemp
  real(8), dimension(Kmax)                    :: EigenValue
  double complex,   dimension(n)                :: EigenVector
  double complex,   dimension(:),   allocatable :: alpha,beta,tmp,residu
  double complex,   dimension(:,:), allocatable :: eivec,zwork

    !!  INIZIALIZATION OF PARAMETERS  !!
  Useguess = .false.
  KMaxUser = KMax
```

```
  targetEn = -5.d0*lenght
  tol = 1.d-9    ! Tolerance of the eigensolutions: $\Vert \beta H_{SB} x - \alpha x \vert$
  maxnmv = 100    ! Maximum number of matvecs in cgstab or gmres (very problem dependent;
    typically choose in [5-100])
  wanted = .true. ! If wanted=.true. then computes the converged eigenvectors
  order = -1      ! Selection criterion for Ritz values:  0 (nearest to target);  -1 (smallest
     real part)
  if(order == 0)  testspace = 3 ! put 3 if a reasonable value for target is known, else take 2
  if(order /= 0)  testspace = 2

  if (3*KmaxUser <= 20) jmax=20          ! Maximum size of the search space:
  if (3*KmaxUser >  20) jmax=3*KmaxUser
  jmin=2*KmaxUser                        ! Minimum size of the search space
  maxstep = 1000                         ! Maximum number of Jacobi-Davidson iterations
  lock = 1.d-12                          ! Tracking parameter
  method = 2                             ! Method for the linear equation solver  1: gmres(m)
     2: cgstab(l)
  m = 30                                 ! Maximum dimension of searchspace for gmres(m):
  l= 2                                   ! Degree of gmres-polynomial in bi-cgstab(l):
  if (method == 1) lwork =  4 +  m  + 5*jmax + 3*KmaxUser  ! Size of workspace
  if (method == 2) lwork = 10 + 6*l + 5*jmax + 3*KmaxUser  !KmaxUser is used since Kmax = 1
    gives problems ...!
  !!  END OF INIZIALIZATION  !!

  allocate (alpha(jmax), beta(jmax), eivec(n,Kmax))
  Alpha=0.d0
  Beta=0.d0
  EiVec=0.d0
  allocate (tmp(n), residu(n), zwork(n,lwork))
  tmp=0.d0
  residu=0.d0
  zwork=0.d0

  call JDQZ(ALPHA, BETA, EIVEC, wanted, n, targetEn, tol, Kmax, jmax, jmin, method, m, l,
    maxnmv, maxstep, &
            lock, order, testspace, zwork, lwork, UseGuess )

  !     Computes the norms of the residuals:
  do j = 1, Kmax
     call AMUL  ( n, eivec(1,j), residu )
     call ZSCAL ( n, beta(j), residu, 1 )
     call BMUL  ( n, eivec(1,j), tmp )
     call ZAXPY ( n, -alpha(j), tmp, 1, residu, 1 )
  end do
  deallocate (zwork,tmp,residu)
  Eigenvalue(1:Kmax) = dReal(alpha(1:Kmax)/beta(1:Kmax))

  !     Calculates the smallest eigenvalue (ground state)
  emin=eigenvalue(1)
  istate = 1
  do ii=2,Kmax
     if (eigenvalue(ii) < emin) then
        emin=eigenvalue(ii)
        istate = ii
     end if
  end do
  if (istate /= 1) then
     etemp=eigenvalue(1)
     eigenvalue(1)=eigenvalue(istate)
     eigenvalue(istate)=etemp
  end if
  deallocate (alpha,beta)

!  print *,'istate',istate
!  Chooses the eigenvector corresponding to the selected eigenvalue
  EigenVector = eivec(:,istate)
  Norm = Sum(dConjg(EigenVector)*(EigenVector))
  EigenVector = EigenVector/(Norm**0.5d0)
  deallocate (eivec)

end subroutine davidson

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine amul(n, psiIn, psiOut)     !!ausiliaria a Davidson, calcola |psiOut> = H*|psiIn>
  use parameters                      !!unica cosa da modificare
```

```
  implicit none
  integer  :: n, i, j, jtemp, temp_amul
  double complex, dimension(dim_hilbert), intent(in)  :: psiIn
  double complex, dimension(dim_hilbert), intent(out) :: psiOut

  psiOut = (0.d0,0.d0)

  do i=1,dim_hilbert           !!interazione -sigma_z*sigma_z
    do j=1,lenght-1
      if (spin_states(i,j) == spin_states(i,j+1))    psiOut(i) = psiOut(i) - psiIn(i)
      if (spin_states(i,j) /= spin_states(i,j+1))    psiOut(i) = psiOut(i) + psiIn(i)
      end do
  end do

  if (periodic) then           !!condizioni periodiche
      do i=1,dim_hilbert
          if (spin_states(i,lenght) == spin_states(i,1))    psiOut(i) = psiOut(i) - psiIn(i)
          if (spin_states(i,lenght) /= spin_states(i,1))    psiOut(i) = psiOut(i) + psiIn(i)
      end do
  end if

  do i=1,dim_hilbert           !!campo longitudinale -sigma(z)
       do j=1,lenght
          if (spin_states(i,j) == 1)   psiOut(i) = psiOut(i) - lambdaAmul*psiIn(i)
          if (spin_states(i,j) == 0)   psiOut(i) = psiOut(i) + lambdaAmul*psiIn(i)
       end do
  end do

  do i = 1,dim_hilbert         !!campo longitudinale -sigma(x)
      do j=1,lenght
          jtemp = lenght +1 -j
          if (spin_states(i,jtemp) == 1)   temp_amul = i -2**(j-1)
          if (spin_states(i,jtemp) == 0)   temp_amul = i +2**(j-1)
        psiOut(temp_amul) = psiOut(temp_amul) + g*psiIn(i)
      end do
  end do

end subroutine amul

subroutine bmul(neq, q, r )       !!tre subroutine ausiliarie qua sotto, servono per compilare
  implicit none
  integer :: neq
  double complex :: q(neq),r(neq)

  r=q

end subroutine bmul

subroutine Guess(N,X)
  implicit none
  integer :: n
  double complex :: X( * )

  X(1:n) = X(1:n)!PsiGuess(1:n) ! NOT IMPLEMENTED

end subroutine Guess

subroutine precon (neq,psi)
  !     This subroutine computes $\psi = K \psi$, where $K$ is a matrix which mimics the
    action of
  !     $(H - \tau \mathbb{1})^{-1}$. Here H is the Hamiltonian to be diagonalized, $\tau$ is
    the target
  !     of the Davidson method, namely the value near which the eigenvalues are sought.
  !     A zeroth order approximation is typically used: $K_{i,j} = \delta_{i,j} (H_{i,i}-\tau)
    ^{-1}$
  implicit none
  integer :: neq
  double complex :: psi(neq)

  psi=psi

end subroutine precon

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Listing 2: Script .f90 per la diagonalizzazione dell'Hamiltoniana.

```
#!/bin/bash
export LC_NUMERIC="en_US.UTF-8"      #sistema le virgole dei decimali
                                     #i file vanno ottenuti compilando almeno una volta le
    librerie

gfortran gaps.f90 davidson.guess.lib2.o lapack_tebd.lib.f -O2 -o gaps.x   #compila linkando i
    file necessari

start=0.00
end=0.05
spacing=0.20

output_file="gaps_3.txt"    #mettere L prima di ogni run (gaps_L.txt), segnarsi se obc, pbc
: > "$output_file"                                  #pulisce il file

for g in $(seq $start $spacing $end); do            #loop per i valori del campo g desiderati
    echo "Sending input g= $g to simulation"
    echo "$g" | ./gaps.x >> "$output_file"          #pipe
done
echo "Done simulating"
```

Listing 3: Script .sh per automatizzare le misure a L fissato; lo script per le misure a campo trasverso g fissato segue essenzialmente la stessa logica.