Dinamica di pacchetti d'onda in sistemi quantistici

Christian Aoufia

Alessandra Grieco

(Dated: 22 giugno 2021)

Il progetto propone di risolvere numericamente l'equazione di Schrödinger tempo-dipendente per un pacchetto d'onda quantistico, prima libero e poi in presenza di due barriere di potenziale, utilizzando l'algoritmo split-step.

INTRODUZIONE

Obbiettivo del progetto è la risoluzione dell'equazione di Schrödinger 1-D tempo-dipendente

$$\frac{\partial}{\partial t}\psi(x,t) = -\frac{i}{\hbar}\hat{\mathcal{H}}\psi(x,t), \text{ dove } \hat{\mathcal{H}} = \hat{\mathcal{T}} + \hat{\mathcal{V}}, \tag{1}$$

per l'intervallo spaziale $[0, L)$ dotato di condizioni al contorno periodiche. L'evoluzione temporale viene tracciata in un intervallo $[0, T]$. Scrivendo l'equazione nella base delle posizioni si ha:

$$\hat{\mathcal{T}} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial t^2} \quad \text{e} \quad \hat{\mathcal{V}} = V(x), \tag{2}$$

dove m è la massa della particella e $V(x)$, nel caso preso in analisi in questo progetto, è costituito da due barriere centrate in $x_1 = L/3$ e $x_2 = L/2$, rispettivamente di altezza E_1 ed E_2, e di ampiezza l_1 ed l_2, con $l_1, l_2 \ll L$. Ci si propone di risolvere, come ulteriore modo di valutare l'accuratezza dell'integrazione, anche il caso libero con $V(x) = 0$. La condizione iniziale di entrambi i problemi è data da:

$$\psi(x,0) = \frac{1}{\sqrt[4]{2\pi\sigma^2}} \exp\left\{-\frac{(x-x_0)^2}{4\sigma^2}\right\} e^{ik_0 x}, \tag{3}$$

in cui lo stato iniziale è espresso come pacchetto gaussiano con velocità $\hbar k_0/m$ e $\sigma \ll L$.

DEFINIZIONE DEL SISTEMA DA INTEGRARE NUMERICAMENTE E DESCRIZIONE DEGLI ALGORITMI SELEZIONATI

Si consideri la griglia spazio-temporale di $N \times (M+1)$ punti ottenuta rispettivamente dalla discretizzazione di $[0, L)$ e $[0, T]$ con:

$$\delta x = \frac{L}{N}, \quad \delta t = \frac{T}{M}. \tag{4}$$

Partendo dalla soluzione $\psi(x,t)$ ottenuta facendo agire sulla funzione d'onda iniziale l'operatore di evoluzione $\hat{\mathcal{U}}(x,t) = \exp(-\frac{i}{\hbar}\hat{\mathcal{H}}t)$, posti $\hbar = 1,\ m = 1$ e $t = n\delta t$, si ha:

$$\psi(x, t = n\delta t) = \hat{\mathcal{U}}(x, n\delta t)\psi(x,0) = \exp\left\{\left(-i\delta t(\hat{\mathcal{T}} + \hat{\mathcal{V}})\right)^n\right\}\psi(x,0) \tag{5}$$

Utilizzando la formula di Trotter-Suzuki è possibile definire un operatore approssimato:

$$\hat{\mathcal{G}}(x, \delta t) = \hat{\mathcal{U}}(x, \delta t) + \hat{\mathcal{O}}(x, \delta t^3),$$

dove

$$\hat{\mathcal{G}} = \mathbf{G_1}(\delta t/2)\mathbf{G_2}(\delta t)\mathbf{G_1}(\delta/t) \tag{6}$$

e

$$\mathbf{G_1}(\delta t/2) = \exp\left\{-i\frac{\delta t}{2}V(x)\right\}, \quad \mathbf{G_2}(\delta t) = \exp\left\{i\frac{\delta t}{2}\frac{\partial^2}{\partial x^2}\right\}. \tag{7}$$

Il pacchetto gaussiano viene quindi fatto evolvere nel tempo utilizzando il l'operatore approssimato, commettendo un errore che va come la terza potenza della discretizzazione temporale δt. Notiamo inoltre che il propagatore $\mathbf{G_1}$ è diagonale (quindi moltiplicativo) nella base delle posizioni, mentre $\mathbf{G_2}$ lo è nella base duale dei momenti. E' quindi opportuno, prima di applicarlo, effettuare una trasformata di Fourier. L'algoritmo iterativo *split-step* spettrale, applicato all'equazione di Schrödinger dipendente dal tempo, è quindi riassumibile nei seguenti passaggi, effettuati per ogni timestep $j = 0, \dots n-1$:

1. Applicare $\mathbf{G_1}$ alla funzione d'onda in spazio delle posizioni, che agisce moltiplicativamente:

$$\mathbf{G_1}(\delta t/2)\phi(x) = e^{-i\frac{\delta t}{2}V(x)}\phi(x); \tag{8}$$

2. Effettuare la trasformata di Fourier alla funzione d'onda ottenuta al passo 1;

3. Applicare l'operatore $\tilde{\mathbf{G_2}}$, che agisce ora moltiplicativamente su $\phi(k)$:

$$\tilde{\mathbf{G_2}}(\delta t)\phi(k) = e^{-i\frac{\delta t}{2}k^2}\phi(k); \tag{9}$$

4. Effettuare l'antitrasformata della funzione d'onda ottenuta al passo 3 per tornare nello spazio delle posizioni;

5. Applicare nuovamente $\mathbf{G_1}$.

La trasformata di Fourier viene svolta utilizzando il pacchetto Fortran `FFTW`,che implementa l'algoritmo Fast Fourier Transform . Ci si propone di verificare, anche graficamente, l'evoluzione del pacchetto utilizzando `matplotlib` e di andare a studiare l'andamento dell'accuratezza dell'integrazione in funzione di $\delta x/\delta t$. Per valutare questa quantità ci si propone nel caso libero e in presenza di potenziale di andare a monitorare quantità il cui andamento è conosciuto analiticamente (nel caso del potenziale a tratti, si utilizza solamente la conservazione dell'energia e della norma). In particolare, nel caso libero la velocità della particella $\langle p \rangle$ è costante e l'andamento temporale di σ^2 è conosciuto analiticamente:

$$\sigma^2(t) = \sigma^2(0) + \frac{\hbar^2}{4m\sigma^2(0)}t^2. \tag{10}$$

Sia nel caso libero che in presenza del potenziale sono poi conservati norma, che inizialmente è unitaria perché il pacchetto è normalizzato, ed energia. La deviazione rispetto a questi valori analitici è calcolata come massimo della differenza (in valore assoluto) rispetto al valore iniziale.

IL CODICE

Il codice viene strutturato in due "shell": il "core" principale costituito da `gen.py`, `main.f90`, `mod1.f90` e opzionalmente `evograph.py`, con file di input `fort_input.txt`, si occupa di risolvere la propagazione del

pacchetto per una fissata griglia spazio-temporale, stampare i risultati su un certo numero di file di output, il cui nome è specificato in quello di input, e di produrre i grafici dell'evoluzione del pacchetto utilizzando `matplotlib`.Un "wrapper" python costituito da `init.py`, con input `input.txt`, si occupa invece di gestire ed eseguire il codice "core" per diverse griglie, scrivendo di volta in volta diversi `fort_input.txt`. I risultati di integrazione vengono infine analizzati e passati a `errgraph.py` che produce i grafici degli errori sulle quantità monitorate. Si può decidere di utilizzare il "core" standalone, senza fare affidamento al wrapper.

A. Descrizione file di input

Il file di input al "core" `fort_input.txt` contiene informazioni sulla griglia, una variabile binaria `free` che ci dice se svolgere i calcoli per una particella libera (`free` = 1) oppure in presenza delle due barriere (`free` = 0), la cui altezza è specificata dalle variabili E1 ed E2. E' specificato inoltre il "tempo di acquisizione" ovvero ogni quanto immagazzinare l'informazione sullo stato del pacchetto (per evitare output che occupino troppa memoria, si memorizza l'evoluzione solo a una frequenza fissata). Sono infine presenti i nomi di file su cui stampare i risultati e il nome del file da cui leggere il pacchetto iniziale.

```
2048 512 0 ! N, M, free
500 60 1 2 2 ! Lunghezza spaziale, lunghezza temporale, tempo acquisizione, E1, E2
'monitor.txt' 'evo.txt' 'packet.txt' 'output.txt' ! nomi file di interesse
```

Listing 1 File di input fort_input.txt.

Il file input al "wrapper" `input.txt` contiene le stesse informazioni del precedente file, ma abbiamo importanti cambiamenti. E' inclusa una "modalità di operazione", che può assumere valori 0 o 1, il cui significato verrà spiegato successivamente. Inoltre, è presente una lista di griglie finali o alternativamente un range di valori da cui creare griglie.

```
0 'output.txt' !modalit  di operazione, file di output
500 60 1 2 2 ! Lunghezza spaziale, lunghezza temporale, tempo acquisizione, E1, E2
'monitor.txt' 'evo.txt' 'packet.txt' ! nomi file di interesse
2048 512 0 ! lista di valori N, M, free
2048 768 0
2048 1024 0
```

Listing 2 File di input input.txt, con liste di valori

```
1 'output.txt' !modalit  di operazione, file di output
500 60 1 2 2 ! Lunghezza spaziale, lunghezza temporale, tempo acquisizione, E1, E2
'monitor.txt' 'evo.txt' 'packet.txt' ! nomi file di interesse
1024 2048 64 512 512 128 0 !Nstart, Nstop, Nstep, Mstart, Mstop, Mstep, free
```

Listing 3 File di input input.txt, con range di valori

E' infine presente anche un altro file di input a `gen.py` (oltre a `fort_input.txt`), chiamato `genparams.txt`, contenente k_0 e σ^2 del pacchetto iniziale. Il suo nome, non modificabile, viene utilizzato anche nel modulo `mod1.f90`:

```
-2 5 !k0, sigma
```

Listing 4 File di input genparams.txt

B. File `gen.py`

Il file python `gen.py` si occupa di generare il pacchetto gaussiano iniziale, fornito in input il numero di punti della griglia spaziale e la sua lunghezza. Queste informazioni sono ovviamente estratte da `fort_input.txt`. Il pacchetto viene stampato in un file a scelta, che di default viene chiamato `packet.txt`, utilizzando una formattazione che permetta al codice Fortran di interpretarlo correttamente come array di valori complessi. I parametri del pacchetto gaussiano, k_0 e σ, sono letti da `genparams.txt`, mentre i parametri della griglia sono letti dallo stesso input del core Fortran, `fort_input.txt`. Il pacchetto generato, in fase di debug, viene anche graficato con `matplotlib`in `debugpacket.jpg`, per avere controllo diretto sul corretto campionamento.

C. Modulo `mod1.f90`

Il modulo contiene le utilities necessarie al programma principale. La `subroutine grids` prende in ingresso la lunghezza dell'intervallo spaziale e il suo numero di punti N, quella dell'intervallo temporale e il suo numero di intervallini M; restituisce in output le griglie nello spazio reale x, nello spazio reciproco k (che sono prodotte secondo la convenzione della libreria `FFTW3`, ovvero con la prima metà che contiene i vettori d'onda positivi e la seconda metà che contiene quelli negativi in ordine decrescente in modulo) e la griglia temporale t, e i relativi parametri δx, δk e δt.

La `subroutine makeV` si occupa di calcolare l'operatore di evoluzione del potenziale secondo (8) sia nel caso libero, in cui è l'identità, sia nel caso in presenza di potenziale. La subroutine prende in ingresso la griglia spaziale, le spaziature dt e dx, il parametro binario `free` e l'altezza delle barriere, restituendo il potenziale V e l'operatore `propV`. Per convenzione, la larghezza "nominale" delle barriere è presa fissa, con $l = l_1 = l_2$ e pari a $L/137$, rispettando la condizione $l \ll L$. Per come è strutturata la subroutine, invece, la larghezza reale dipende da δx ed è almeno una volta questa quantità. Ciò è dovuto alla necessità di discretizzare anche il potenziale, quindi non c'è modo di rimuovere la correlazione tra larghezza delle barriere e griglia.

La `function makeT` si occupa di calcolare l'operatore di evoluzione dell'energia cinetica `propT` secondo (9), prendendo come input la griglia k. L'operatore è quindi calcolato nella base dei momenti, in modo da essere moltiplicativo per la trasformata di Fourier del pacchetto gaussiano.

La `subroutine fourier` si occupa di effettuare trasformate e antitrasformate di Fourier, a seconda che il parametro `info` sia 1 o 0 rispettivamente, utilizzando la libreria `FFTW3`. La trasformata viene anche normalizzata.

La `subroutine monitor` prende in ingresso il pacchetto ad un particolare passo di iterazione, le griglie e il potenziale e calcola le quantità che devono essere monitorate durante il moto: il valore d'aspettazione dell'energia, la normalizzazione, il valore medio della posizione, la velocità e la σ del pacchetto gaussiano. Le ultime tre quantità sono particolarmente utili, come verrà discusso in seguito, quando andremo a studiare l'accuratezza dell'integrazione nel caso libero. Tutti i valori vengono calcolati integrando sulla griglia x (o k, per l'energia cinetica e la velocità) con la formula trapezoidale e immagazzinati in un vettore nel programma

principale.

```
        U = 0 !VALORE ASPETTAZIONE ENERGIA POTENZIALE
        DO j = 1, N-1
            U = U + L/(2*dble(N)) * (V(j-1)*conjg(packet(j-1))*packet(j-1) + V(j)*conjg(packet
    (j))*packet(j))
        END DO
        U=U/norm

        CALL fourier(packet,N,trpacket,0)

        normtr = 0  !NORMALIZZAZIONE TRASFORMATA
        DO j = 1, N-1
            normtr = normtr + (4*atan(1.d0)/L) * (conjg(trpacket(j-1))*trpacket(j-1) + conjg(
    trpacket(j))*trpacket(j))
        END DO

        T = 0 !VALORE ASPETTAZIONE ENERGIA CINETICA
        DO j = 1, N-1
            T = T +  (4*atan(1.d0)/L) * (k(j-1)**2/2*conjg(trpacket(j-1))*trpacket(j-1) + k(j)
    **2/2*conjg(trpacket(j))*trpacket(j))
        END DO
        T = T/normtr
```

Listing 5 alcuni esempi di integrazione. Non è strettamente necessario dividere T e U per la norma del pacchetto (o della sua trasformata) perché questi dovrebbero essere già normalizzati; tuttavia è comunque necessario controllare la normalizzazione ad ogni passaggio, che dà informazioni sulla corretta evoluzione.

La `subroutine mintegrals` si occupa valutare l'accuratezza dell'integrazione. Essa prende in ingresso i vettori delle quantità acquisite durante il moto, le griglie ed il valore di `free` e restituisce in output gli "errori" associati a ciascuna delle quantità. Questi vengono calcolati come massimo della differenza, in valore assoluto, rispetto al valore nell'istante iniziale, letto da `genparams.txt` o calcolato, alla legge analitica nel caso di $\sigma^2(t)$ e all'unità nel caso della normalizzazione. Se si sta risolvendo il problema libero (`free` = 1), allora si vanno a calcolare gli errori su $v(t)$, $\sigma^2(t)$, energia e normalizzazione. In caso contrario si calcolano solamente le ultime due quantità.

```
!CONTROLLO INTEGRALI DEL MOTO
        !CALCOLO ERRORE SU ENERGIA
        stddevE = maxval(abs(Ei-E(1:counter-1)))

        !CALCOLO ERRORE SU NORMA
        normdiff = maxval(abs(1-norm(1:counter-1)))

        IF (free == 1) THEN

            !CALCOLO ERRORE SU VELOCITA'
            veldiff = maxval( abs(vi-vel(1:counter-1) ))
```

```
13              !CALCOLO ERRORE SU SIGMA^2
                sigmadiff = maxval( abs( sigma(1:counter-1) - sigmai**2 - tsample(1:counter-1)**2/
    (4*sigmai**2) ) )
15
            ENDIF
```

Listing 6 subroutine mintegrals

D. Il programma `main.f90`

Il programma principale si occupa in primo luogo di leggere `fort_input.txt` e di allocare in modo adeguato gli array. L'esecuzione procede poi leggendo pacchetto nell'istante iniziale dal file specificato in input, che nel nostro caso è chiamato `packet.txt`. Questo viene immagazzinato nella matrice `evolution` che alla fine del programma conterrà l'evoluzione del pacchetto ogni `phi` passi temporali. Viene successivamente applicato iterativamente l'algoritmo *split-step* per la propagazione del pacchetto:

```
!PASSO 0
2 evolution(0,:) = packet(:)
CALL monitor(N,L,k,x,packet,V,E(0),sigma(0),xbar(0),norm(0))
4 tsample(0) = 0
counter = 1
6
!ITERAZIONE
8 DO i = 1,M
    !AZIONE 1 IN SPAZIO REALE PROPV
10  packet(:) = propV(:)*packet(:)
    !AZIONE 2 IN SPAZIO K PROPT
12  CALL fourier(packet,N,trpacket,0)
    trpacket(:) = propT(:)*trpacket(:)
14  CALL fourier(trpacket,N,packet,1)
    !AZIONE 3 IN SPAZIO REALE PROPV
16  packet(:) = propV(:)*packet(:)
    !CONTROLLO SE E' TEMPO DI MONITORARE
18  IF ( t(i) >= phi*counter ) THEN
        tsample(counter) = t(i)
20      CALL monitor(N,L,k,x,packet,V,E(counter),sigma(counter),xbar(counter),norm(counter))
        evolution(counter,:) = packet(:)
22      counter = counter + 1
    ENDIF
24
END DO
```

Listing 7 Iterazione split-step

Alla fine del ciclo di iterazione vengono calcolati gli errori sugli integrali del moto e vengono stampati in output i risultati.

E. Descrizione dei file di output

Abbiamo complessivamente tre file di output dal codice di "core": nel nostro caso vengono chiamati `'monitor.txt'`, `'evo.txt'` e `'output.txt'`. Il primo di questi contiene tutte le quantità che sono state precedente monitorate durante il moto:

```
1   Energy(t)              Sigma(t)              xbar(t)                ...
    12.501543358324138     1358.9514893506660    1365.3332236349820     ...
3   12.501543358324144     1358.9521944520143    1362.9332236288676     ...
    12.501543358324133     1358.9543218540498    1360.5332236227571     ...
5   12.501543358324144     1358.9578715567416    1358.1332236166429     ...
    ...
```

Listing 8 monitor.txt, il file contiene anche vel(t) e norm(t), che non sono inserite per motivi di spazio

`evo.txt` contiene l'evoluzione della densità di probabilità del pacchetto, acquisita ogni `phi` passi. Nella prima e seconda riga sono inoltre stampate rispettivamente la griglia e il potenziale:

```
    0.0000000000000000          1.0000000000000000          2.0000000000000000     ...
2   0.0000000000000000          0.0000000000000000          0.0000000000000000     ... (caso libero)
    00000013E-010    5.0000000000000013E-010    8.9999999999999999E-010 ...
4   ....
```

Listing 9 evo.txt

`output.txt` contiene informazioni sull'accuratezza dell'integrazione eseguita. Il file viene aperto in modalità `append` così che sia possibile eseguire più volte il "core" e tracciare l'andamento dell'integrazione in funzione dei parametri di griglia:

```
    free     N      L      M      LT        stddevE       veldiff        sigmadiff      normdiff
2   0        1024   500    512    60.       ...
```

Listing 10 una riga di esempio di output.txt, per motivi di spazio non tutti i valori sono riportati.

F. Il file `evograph.py`

Il file `evograph.py` prende in ingresso `evo.txt` e traccia, per ogni frame, il grafico del pacchetto e del potenziale a cui è soggetto, utilizzando `matplotlib`. E' inoltre prodotta una GIF `evolution_ani.gif` formata unendo i precedenti frames.

G. Il file `init.py`

`init.py` è il primo dei file del "wrapper": si occupa di prendere in ingresso `input.txt` e di iterare il codice fortran sotto opportune condizioni. Il programma (compila ed) esegue solamente `gen.py` e `main.f90`, mentre l'esecuzione di `evograph.py` è opzionale e in grado di produrre il grafico solo per l'ultima iterazione. Il wrapper opera in due possibili modalità. La modalità 0 o manuale prende in ingresso una lista di triplette $N, M, free$, insieme a tutti gli altri parametri in `input.txt` e produce, per ognuna di queste, un file `fort_input.txt`, a cui segue l'esecuzione del "core".

H. Il file `errgraph.py`

Il file `errgraph.py` si occupa di prendere in ingresso `output.txt` e di fare il grafico degli andamenti delle quantità che rappresentano l'accuratezza dell'integrazione. Se la particella è libera vengono graficati $stddevE$, $xdiff$, $sigmadiff$, e $normdiff$, mentre se non lo è solamente la prima e l'ultima quantità.

ESPERIMENTI NUMERICI

A. Considerazioni preliminari

Notiamo prima di tutto che nella generazione del pacchetto esiste un limite superiore a δx, al di sopra del quale l'implementazione di (3) non genera il pacchetto corretto. Facendo alcuni tentativi, si vede che modificando il parametro di `gen.py` k_0 e con diversi valori di L ed N, il limite si trova quando $k_0/2\pi$ è circa la metà di $1/\delta x$. Siccome quello che viene fatto per implementare (3) numericamente non è nient'altro che un campionamento, il limite sulla larghezza di δx è dovuto ad un fenomeno tipo-aliasing ed è in accordo con il teorema del campionamento di Shannon. Infatti, facendo la trasformata di Fourier del pacchetto gaussiano, la componente di frequenza maggiore è proprio k_0; è quindi è necessario prestare attenzione quando si scelgono i parametri della griglia spaziale e k_0, altrimenti non verrà generato il pacchetto corretto. Infine, durante l'esecuzione degli esperimenti numerici, lasciamo fissi $L = 200$, $LT = 50$, $k_0 = -2$ e $\sigma^2(0) = 25$, mentre per variare δx e δt andiamo a cambiare rispettivamente N ed M. Ad M fissato scegliamo $N \in [160, 2048]$ con passo $\delta N = 32$. Per N fissato scegliamo invece $M \in [16, 1536]$ con passo $\delta M = 32$. L'intervallo temporale è scelto in modo che il pacchetto (libero) non attraversi il bordo dell'intervallo $[0, L)$, in quanto ai bordi il calcolo di alcuni integrali (e di conseguenza il grafico degli errori) perde di significato. Infatti, le condizioni periodiche al contorno fanno sì che il pacchetto, arrivato al bordo della griglia spaziale, riemerga dall'altro estremo, e ciò si ripercuote sul calcolo di σ (che non è altro che $\sqrt{\langle (x - \langle x \rangle)^2 \rangle}$). In linea di principio, se si ignora il calcolo errato di questa quantità, l'evoluzione avviene correttamente, con le condizioni periodiche considerate, anche se si scelgono parametri per l'intervallo temporale e la velocità del pacchetto tali per cui esso attraversa i bordi.

B. Pacchetto libero

Eseguiamo inizialmente il programma a N fissato (per diversi valori di N) e variando M, ottenendo i risultati in figura [6-9], che riguardano l'errore dell'integrazione sulle quantità monitorate in funzione di $\delta x/\delta t$. Vediamo che errori su energia e velocità rimangono piccoli, dell'ordine rispettivamente di 10^{-3} e 10^{-7} ed oscillano al variare del rapporto, mentre l'errore su σ^2 generalmente rimane dell'ordine di 10^{-2}. Anche l'errore sulla normalizzazione presenta una variabilità molto piccola sul range testato, dell'ordine di 10^{-6}. Nessun andamento caratteristico ricorrente è individuabile.

Andiamo poi studiare a M fissato (per diversi valori di M) e variando N, ottenendo i risultati in figura [2-5]. Notiamo come gli errori su velocità, normalizzazione ed energia rimangano piccoli ed oscillino attorno ad un certo valore. Questo andamento è inoltre invariante rispetto al valore di M e rimane sempre lo stesso indipendentemente dal range di lavoro. L'andamento di $\delta(\sigma^2)$ presenta inoltre una decrescenza. In generale,

a parte l'ultimo caso analizzato, gli errori sono quindi oscillanti rispetto ad un valore centrale e molto piccoli, indipendentemente dal rapporto $\delta x/\delta t$.

C. Pacchetto in presenza di potenziale

In presenza di un potenziale andiamo a monitorare solamente energia e normalizzazione, sia a N fissato che M fissato. Otteniamo, in tutti i casi, come in figura [10-13], che al di sotto di un certo valore del rapporto, al variare di M, per N fissato a diversi valori, è presente un aumento dell'errore sull'energia. Questo si riscontra anche sulla normalizzazione. Il valore di $\delta x/\delta t$ in cui l'errore sale è dell'ordine dell'unità e si sposta sensibilmente verso sinistra al diminuire di M (o N). Andando ad usare `evograph.py` in questi casi è possibile vedere che sovrapposta al pacchetto, una volta che questo incontra le barriere di potenziale, è presente del rumore non fisico che va a disturbarne l'evoluzione numerica.

IMMAGINI ESPERIMENTI NUMERICI

In alcune delle immagini i valori sull'asse delle ordinate sono incorrettamente tagliati da `matplotlib`. Ciò che importa, in ogni caso, è l'ordine di grandezza delle quantità in gioco, il quale è invece ben visibile.

A. Pacchetto libero - M fissato

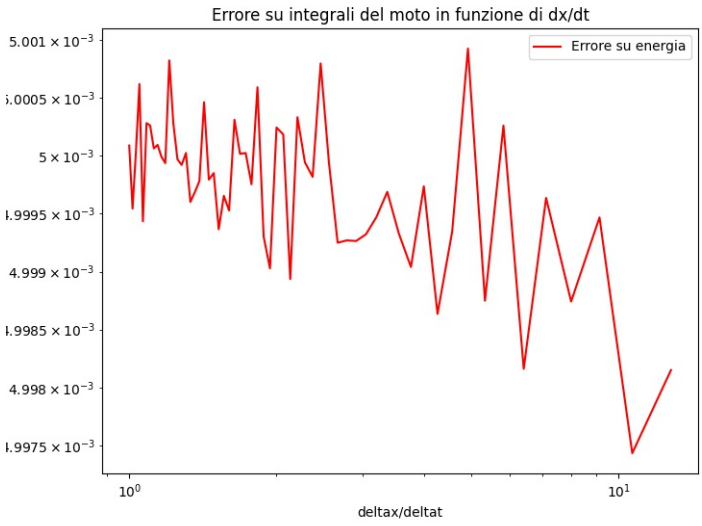

Figura 2 Esempio di andamento dell'errore sull'energia nel caso libero per $M = 512$.

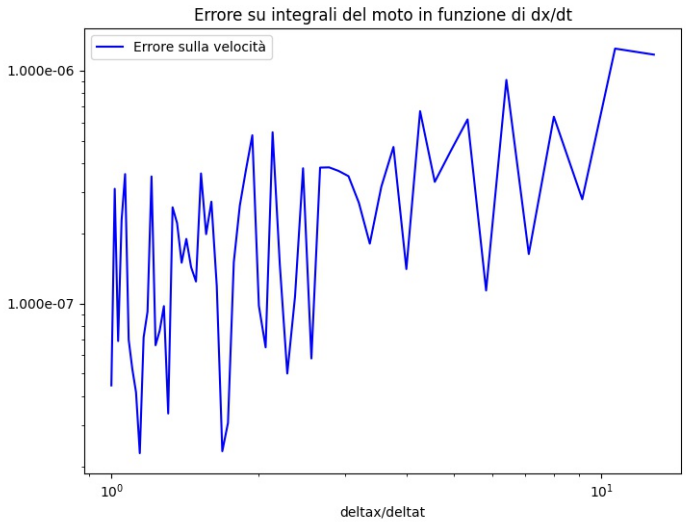

Figura 3 Esempio di andamento dell'errore sulla velocità nel caso libero per $M = 512$.

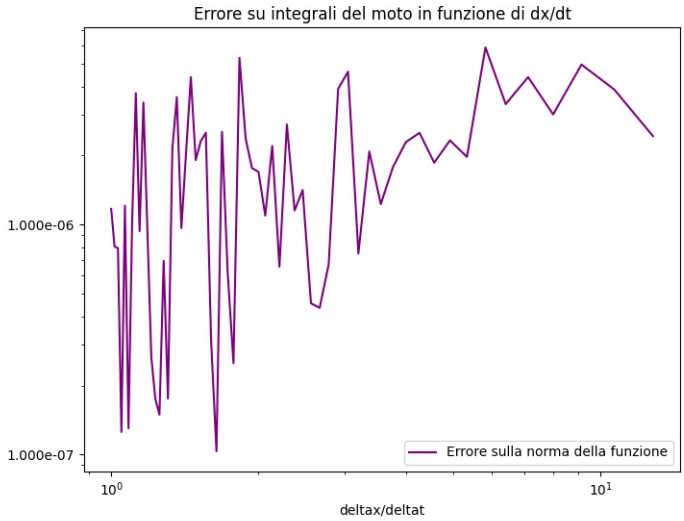

Figura 4 Esempio di andamento dell'errore sulla normalizzazione nel caso libero per $M = 512$.

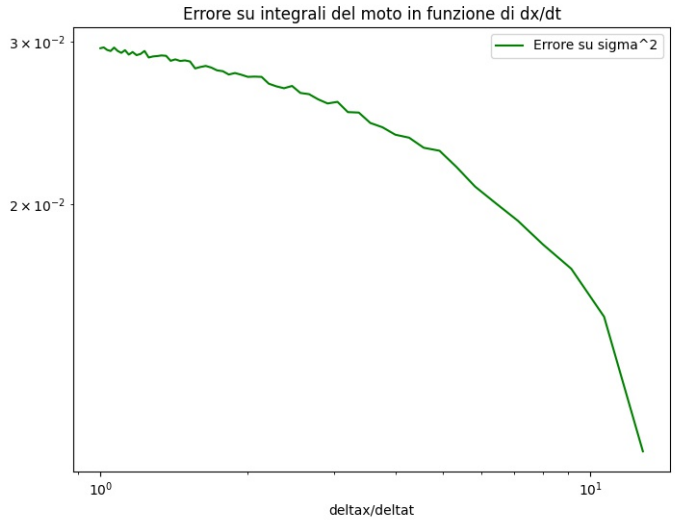

Figura 5 Esempio di andamento dell'errore su $\sigma^2(t)$ nel caso libero per $M = 512$.

B. Pacchetto libero - N fissato

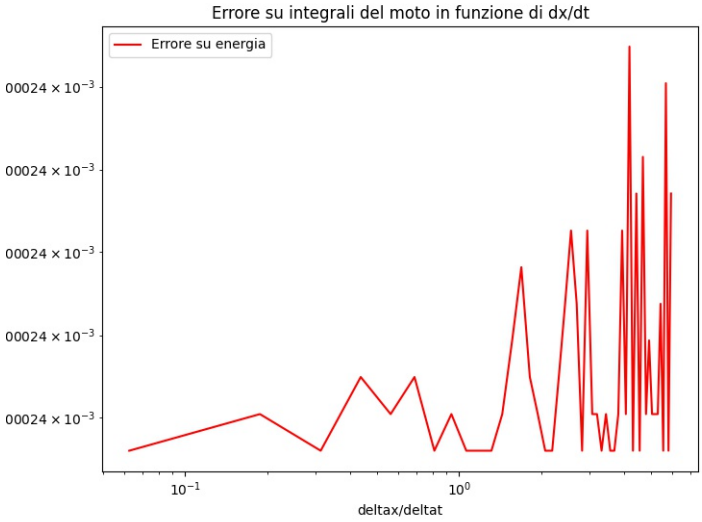

Figura 6 Esempio di andamento dell'errore sull'energia nel caso libero per $N = 1024$.

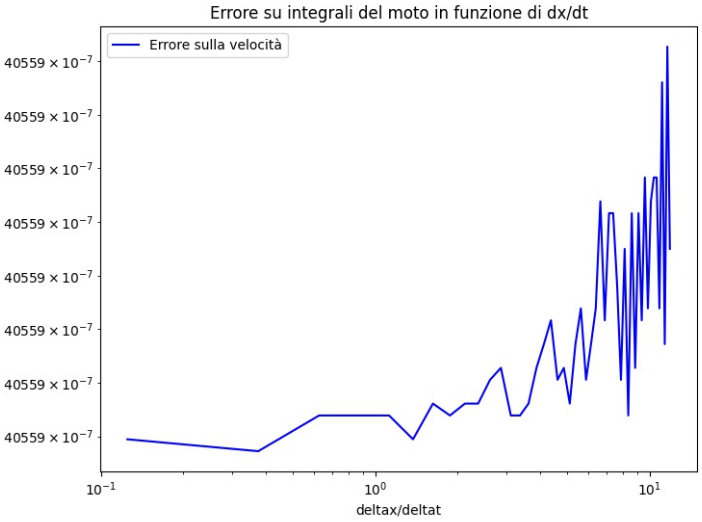

Figura 7 Esempio di andamento dell'errore sulla velocità nel caso libero per $N = 512$.

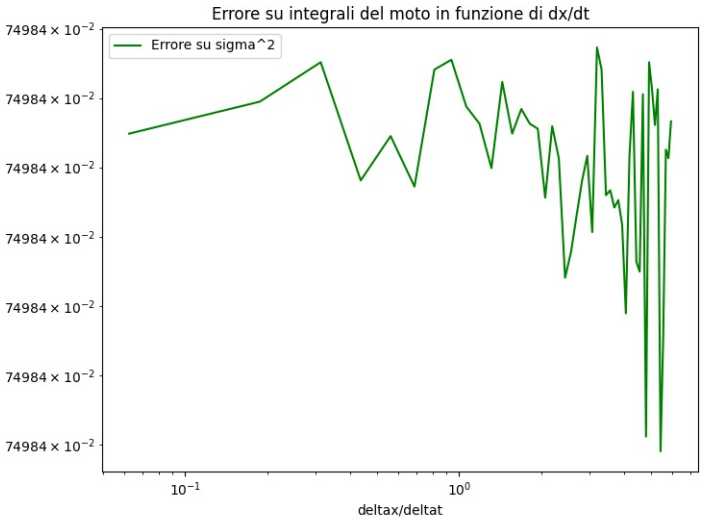

Figura 8 Esempio di andamento dell'errore su $\sigma^2(t)$ nel caso libero per $N = 1024$.

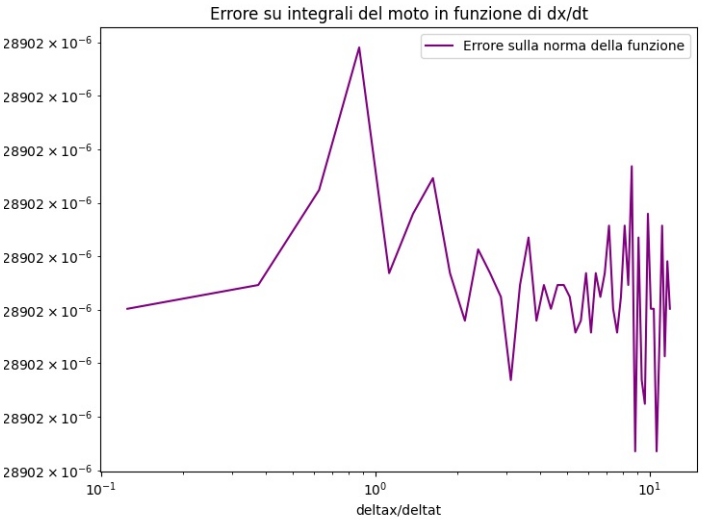

Figura 9 Esempio di andamento dell'errore sulla normalizzazione nel caso libero per $N = 512$.

C. Pacchetto in presenza di potenziale

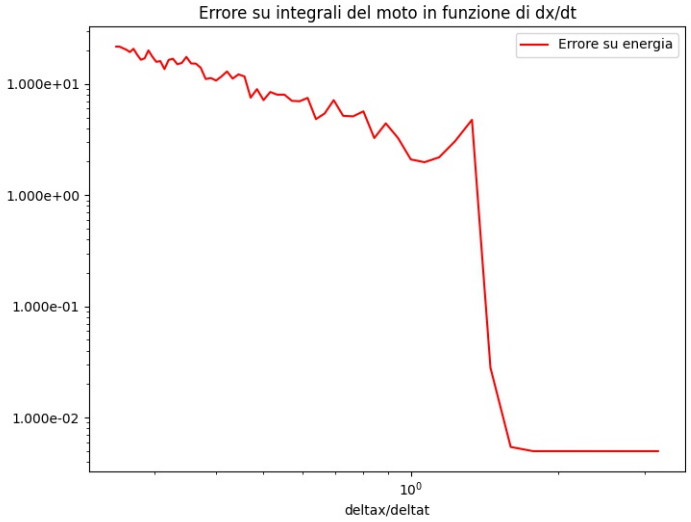

Figura 10 Esempio di andamento dell'errore sull'energia nel caso non libero per $M = 128$.

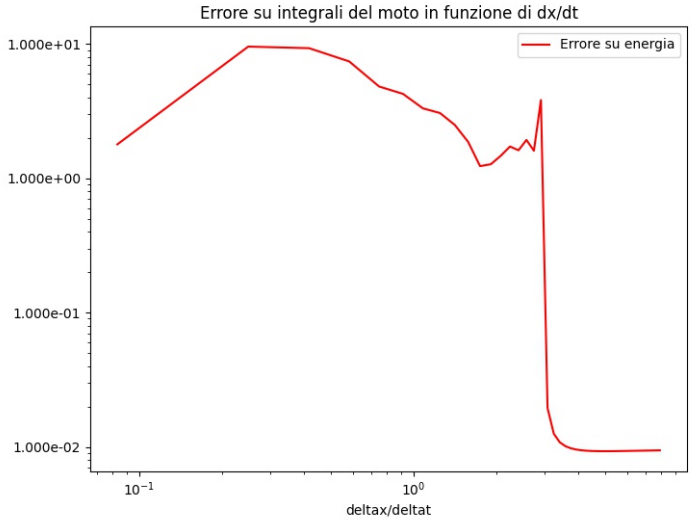

Figura 11 Esempio di andamento dell'errore sull'energia nel caso non libero per $N = 768$.

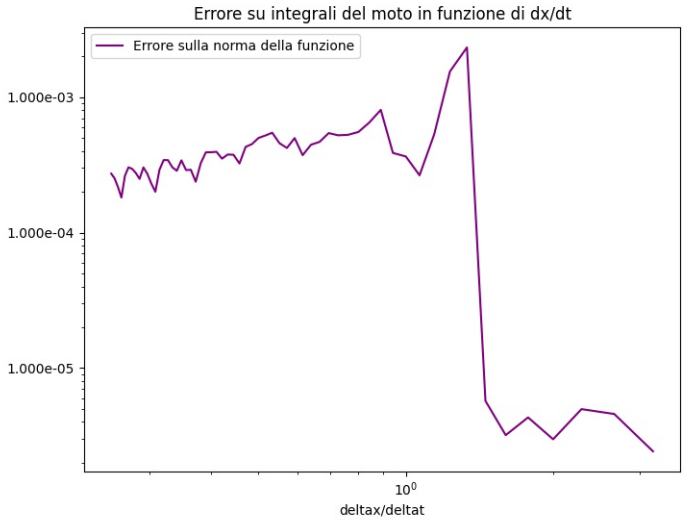

Figura 12 Esempio di andamento dell'errore sulla normalizzazione nel caso non libero per $M = 128$.

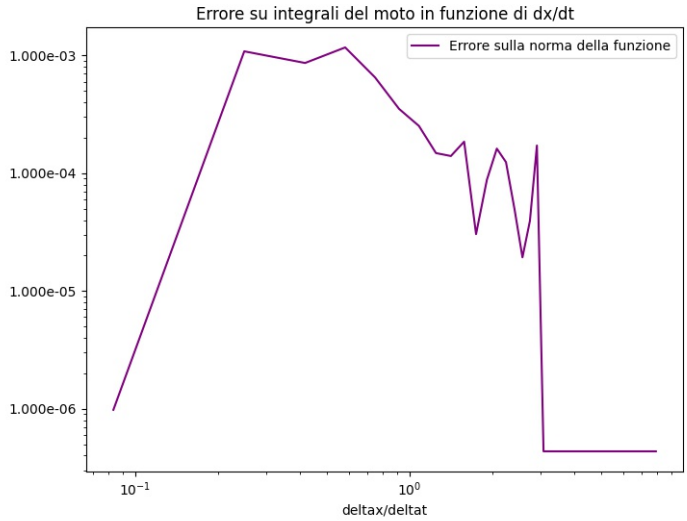

Figura 13 Esempio di andamento dell'errore sulla normalizzazione nel caso non libero per $N = 768$.

DISCUSSIONE DEI RISULTATI E CONCLUSIONI

Abbiamo visto, nel caso libero, che per M fissato otteniamo dei risultati indipendenti dal range di lavoro $\delta x/\delta t$ ed invarianti rispetto ad M stesso. Ciò è aspettato in quanto, per il pacchetto libero, l'operatore di evoluzione non è approssimato con la formula di Trotter-Suzuki al terzo ordine ma è esattamente quello implementato, poiché all'esponente c'è solo l'operatore cinetico. Di conseguenza, la discretizzazione temporale gioca un ruolo debole sull'errore globale, che dipende più pesantemente da δx, il quale varia sempre nello stesso range durante questi specifici esperimenti numerici. Ciò è supportato anche dal fatto che i grafici degli errori a N fissato siano essenzialmente oscillanti attorno ad un valore medio.

Per quanto riguarda il caso in presenza di un potenziale, notiamo prima di tutto che lo spessore delle barriere l, nonostante nominalmente fisso, varia con δx, in quanto il potenziale viene necessariamente costruito su una griglia. Per questo motivo, riteniamo significanti quei risultati che prevedono $\delta x < l$. Inoltre, in luce dell'andamento analizzato nella precedente sezione, ipotizziamo che l'errore possa essere dovuto ai seguenti motivi:

- la formula di Trotter-Suzuki è applicabile solo a δt infinitesimi, quindi fallisce per valori troppo grandi;

- se interpretiamo $\delta x/\delta t$ come una "velocità caratteristica" dell'integrazione, essa deve essere molto maggiore della velocità del pacchetto data da k_0, in modo che esso percorra una distanza pari a δx in un tempo molto maggiore di δt. Infatti, l'errore si riduce drasticamente quando il rapporto è almeno dell'ordine di k_0 (in valore assoluto). Inoltre, $\delta x/\delta t$ deve essere anche tale che l'interazione con la barriera venga ben rappresentata, quindi δt deve essere molto minore del tempo impiegato dal centro del pacchetto (nell'ipotesi in cui non venga riflesso) ad attraversare lo spessore della barriera. Ciò spiegherebbe lo spostamento del "treshold" sul rapporto per diversi valori di N, al variare di M, e viceversa, in quanto la larghezza della barriera è legata, nel codice utilizzato, a δx.

- potrebbero esserci anche andamenti dovuti alla non corretta rappresentazione del coefficiente di riflessione e trasmissione, ma sono oltre gli scopi di questa analisi.

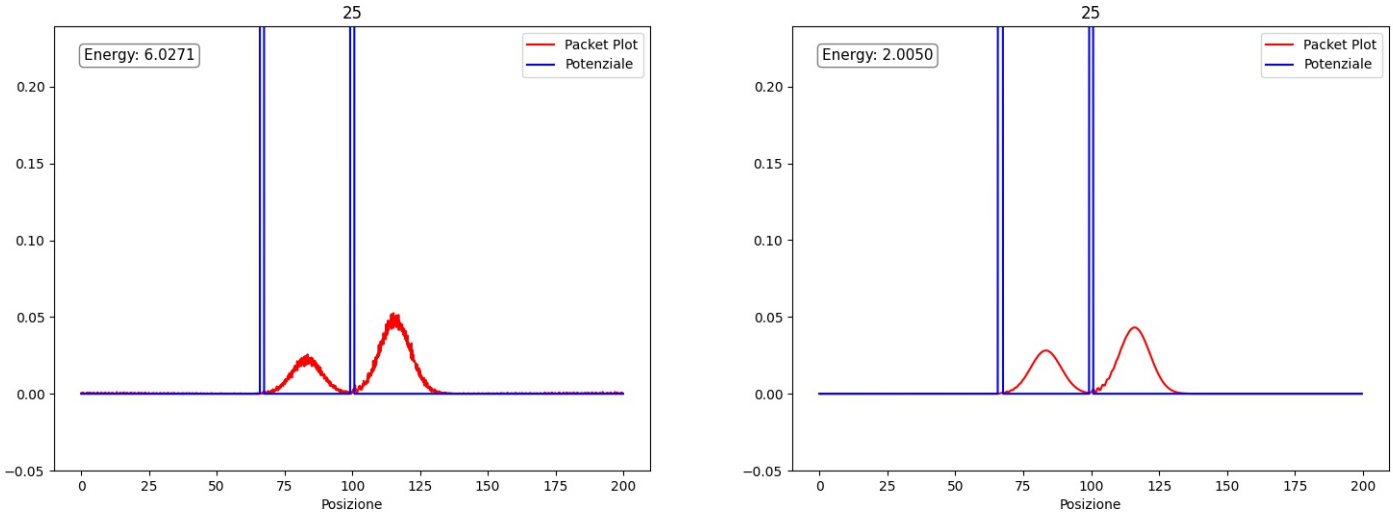

Figura 1 Snapshots della differenza tra l'evoluzione della densità di probabilità per un valore di $\delta x/\delta t$ compatibile e uno non compatibile con le condizioni sopracitate.

Alleghiamo a questo documento due GIF rappresentanti l'evoluzione della densità di probabilità, costruite con le immagini prodotte da `evograph.py`. Una GIF rappresenta l'evoluzione corretta, l'altra quella per un valore di $\delta x/\delta t$ troppo piccolo, per cui si nota bene come l'energia non si conservi.

Sottolineiamo infine che il rapporto, nei grafici, fra l'altezza delle barriere e il picco della densità di probabilità non è indicativo: le barriere sono molto alte perché sono rappresentate sulla stessa scala di un pacchetto normalizzato. Le quantità da confrontare sono l'energia totale del pacchetto (nel box in alto) e l'altezza delle barriere, che nelle nostre simulazioni è fissata a 2, ma variabile nel file di input per studiare, eventualmente, come cambiano i coefficienti di riflessione e trasmissione.

In conclusione, il programma sviluppato attraverso l'algoritmo *split-step* è in grado di predire correttamente l'evoluzione di un pacchetto gaussiano secondo l'equazione di Schr̈oedinger; tuttavia è necessario fare attenzione ai parametri scelti per la discretizzazione temporale e spaziale, specialmente, nel caso di barriere di potenziale "sottili", per quanto riguarda il rapporto $\delta x/\delta t$, e in generale la discretizzazione spaziale deve rispettare i criteri del teorema del campionamento di Shannon.