Universitá degli Studi di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di laurea in Fisica

Relazione Laboratorio di Fisica Computazionale

Simulazione numerica della propagazione di un pacchetto d'onda

Gruppo Braidi-Tommasi

Federico Braidi

Anno Accademico 2022-2023

Sommario

Si studia la propagazione di un pacchetto d'onda gaussiano attraverso barriere di potenziale mediante una simulazione numerica sviluppata in Python.

Introduzione[2]

Teoria

Ci poniamo l'obiettivo di risolvere l'equazione di Schroedinger dipendente dal tempo espressa dalla formula:

$$\frac{\partial}{\partial t}\psi(x,t) = -\frac{\imath}{\hbar}\widehat{\mathscr{H}}\,\psi(x,t) \tag{1}$$

La $\widehat{\mathscr{H}}$ si puó scrivere nelle sue due componenti:

$$\widehat{\mathscr{H}} = \widehat{\mathscr{K}} + \widehat{\mathscr{V}} \tag{2}$$

Dove

$$\widehat{\mathscr{K}} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} \quad , \quad \widehat{\mathscr{V}} = V(x) \tag{3}$$

Prendiamo come stato iniziale un pacchetto d'onda gaussiano con $\sigma << L$ (dimensione dello spazio simulato) che si muove con velocitá iniziale $v_i = \frac{\hbar k_0}{m}$. La funzione che lo descrive é:

$$\psi(x,0) = \frac{1}{\sqrt[4]{2\pi\sigma^2}}\exp\left\{-\frac{(x-x_0)^2}{4\sigma^2}\right\}e^{\imath k_0 x} \tag{4}$$

Per studiarne la propagazione vogliamo applicare l'operatore di evoluzione temporale che é dato da:

$$\widehat{\mathscr{U}}(x,t) = e^{-\frac{\imath}{\hbar}\widehat{\mathscr{H}}t} \tag{5}$$

Dobbiamo ora considerare la natura quantizzata della nostra analisi e mettere in evidenza gli step temporali a cui siamo soggetti. Scegliamo step omogenei $\delta t = \frac{T}{M}$ e utilizziamo l'operatore di evoluzione che ci porta dal timestep corrente al successivo:

$$\psi(x, t+\delta t) = e^{-\frac{\imath}{\hbar}\widehat{\mathscr{H}}\delta t}\,\psi(x,t) \tag{6}$$

Usiamo la formula di Trotter-Suzuki per definire un propagatore approssimato $\widehat{\mathscr{G}}$ che si discosta da $\widehat{\mathscr{U}}$ per un termine che dipende da δt^3 ([5]):

$$\widehat{\mathscr{G}}(x,\delta t) = G_3\left(\frac{\delta t}{2}\right)\cdot G_2\left(\delta t\right)\cdot G_1\left(\frac{\delta t}{2}\right) \tag{7}$$

Le espressioni di questi G_1, G_2, G_3, se scritti in unitá di misura naturali ([6]), sono :

$$\begin{aligned} G_1\left(\frac{\delta t}{2}\right) = G_3\left(\frac{\delta t}{2}\right) &= \exp\left\{-\imath\frac{\delta t}{2}V(x)\right\} \\ G_2 &= \exp\left\{\imath\frac{\delta t}{2}\frac{\partial^2}{\partial x^2}\right\} \end{aligned} \tag{8}$$

Quindi per evolvere la funzione d'onda da un istante al successivo bisogna applicarle nell'ordine G_1, G_2 e infine G_3. Una volta determinato il potenziale applicare G_1 e G_3 é piuttosto semplice. Ad un fissato tempo si ha:

$$G_1\left(x, \frac{\delta t}{2}\right)\psi(x) = e^{-\imath\frac{\delta t}{2}V(x)}\psi(x) \tag{9}$$

ed il calcolo é analogo per G_3 $(= G_1)$.

Il problema é leggermente piú complesso per G_2. Invece di applicare la derivata doppia si puó trasferire il problema nello spazio delle fasi $(p = \hbar k)$ nel quale la rappresentazione di G_2 diventa:

$$G_2(\delta t) = \exp\left\{\imath\frac{\delta t}{2}\frac{\partial^2}{\partial x^2}\right\} \implies \widetilde{G_2}(\delta t) = \exp\left\{-\imath\frac{\delta t}{2}k^2\right\} \tag{10}$$

Dunque calcolare la trasformata di Fourier della $\psi(x,t)$ definita come:

$$\widetilde{\psi(k,t)} = \int_{-\infty}^{\infty} e^{-\imath kx}\,\psi(x,t)\,dx \tag{11}$$

Applicare $\widetilde{G_2}$ a $\widetilde{\psi(k,t)}$:

$$\widetilde{G_2}(\delta t)\,\widetilde{\psi(k)} = e^{\imath\frac{\delta t}{2}k^2}\widetilde{\psi}(k) \tag{12}$$

E riportare il problema nello spazio delle coordinate da cui si era partiti.

$$\psi(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{\imath kx}\,\widetilde{\psi(x,t)}\,dx \tag{13}$$

Si puó allora sintetizzare il procedimento a ogni timestep temporale con questi passi:

- Applicazione di G_1 alla funzione corrente secondo la formula riportata in precendenza (9)
- Calcolo della trasformata di Fourier del risultato del punto precendente (11)
- Applicazione di $\widetilde{G_2}$ alla trasformata appena calcolata (12)
- Calcolo dell'antitrasformata di Fourier per tornare nello spazio delle coordinate (13)
- Applicazione di G_3, analoga a quella di G_1 (9)

Il Codice[1]

Seguendo le indicazioni della teoria viene sviluppato un codice Python cosí strutturato:

- Una classe **Onda** che contiene gli attributi e le azioni che possiamo compiere sull'onda, oggetto della nostra simulazione
- Una classe **Animazione** responsabile di creare i frame del grafico (animazione) a partire dai dati ricavati dalla simulazione
- Un file principale **MainOnda**:
  - Definisce i parametri della simulazione
  - Utilizzando i metodi di **Onda** porta a termine i calcoli per la simulazione
  - Usando i metodi di **Animazione** genera l'animazione relativa

I file completi si possono trovare nel repository GitHub dedicato [1]. Qui riportiamo una versione commentata del codice:

Classe Onda

La classe **Onda** é composta da 11 attributi e 2 metodi (uno dei quali é il costruttore della classe). Riportiamo di seguito una breve descrizione dei loro scopi:

- Attributi:
  - **L**: contiene la lunghezza dell'intervallo spaziale in cui vogliamo avvenga la propagazione.
  - **T**: contiene la lunghezza dell'intervallo temporale in cui vogliamo avvenga la propagazione.
  - **N**: numero di step spaziali in cui é divisa la lunghezza **L** (come detto nella teoria dividiamo lo spazio per passare dal dominio continuo a quello discreto nel quale siamo in grado di calcolare l'evoluzione numerica del sistema).
  - **M**: numero di step temporali in cui é diviso il tempo **T** (vedi punto precedente).
  - **V**: é un vettore di dimensione **N** che contiene, per ogni punto di griglia spaziale, il valore del potenziale in cui si deve propagare l'onda. Viene passato dal main al momento della creazione dell'oggetto.
  - **norma**: é un vettore di dimensione **M** in cui viene inserito, durante l'evoluzione dell'onda, il suo modulo quadro ad ogni timestep. Serve per controllare che la norma si conservi durante il processo.
  - **k**: é un vettore di dimensione **N** che contiene i vettori d'onda ottenuti secondo la formula $\frac{2\pi\omega N}{L}$ in cui gli ω sono il risultato dalla funzione *fft.fftfreq* di numpy.[4]

- **x**: é un vettore di dimensione **N** che contiene i valori di x a cui si trovano i punti della griglia creata dalla quantizzazione.
- **dx** : contiene il valore δx tra un punto di griglia spaziale e l'altro.
- **dt** : contiene il valore δt tra un punto di griglia temporale e l'altro.
- **traj** : é una matrice di dimensioni (**N**,**M**) che viene piano piano riempita durante l'evoluzione. Ogni colonna é relativa ad un punto di griglia temporale e contiene i valori della funzione a quel tempo nei vari punti della griglia spaziale.

- Metodi:
  - Costruttore (__init__):

```
def __init__ (self, L,T,N,M,V):
        """
        Costruttore:
        L = lunghezza intervallo spaziale
        T = lunghezza intervallo temporale
        N = step spaziali
        M = step temporali

        Return:
        traj = matrice di N*(M+1)
        """
        self.L = L
        self.T = T
        self.N = N
        self.M = M+1
        self.V = V
        self.norma = np.zeros(self.M)

        self.k = np.fft.fftfreq(self.N)
        self.k = self.k*((2.*np.pi)*N/self.L)
        self.x = np.linspace(0.,L,N,endpoint=False)
        self.dx = self.x[1]
        self.dt = T/M

        self.traj = np.zeros((self.N, self.M),dtype = "complex_")
```

Listing 1: Costruttore della classe **Onda**

    - Input: **L**,**T**,**N**,**M**,**V**.
    - Compito: Assegna i valori in input alle variabili di istanza, calcola **k** come spiegato in precendenza, **x** , **dx** e **dt**. Inizializza a 0 i vettori **norma** e **traj**.
    - Output: non ha output, i valori assegnati alle variabili di istanza possono essere trovati nelle variabili stesse.

– Evoluzione:

```
def evoluzione(self,num_step):
        #definizione vettori da usare
        y_re_1 = np.zeros(self.N, dtype = "complex_")
        y_compl = np.zeros(self.N, dtype = "complex_")
        y_re_2 = np.zeros(self.N, dtype = "complex_")
        #applicazione di G1
        y_re_1 = np.exp(-1j*self.dt/2*self.V)*self.traj[:,num_step-1]
        #applicazione di G2
        y_compl = np.fft.fft(y_re_1)
        y_compl = np.exp(-1j*self.dt/2*(self.k[:]**2))*y_compl
        #applicazione di G3
        y_re_2 = np.fft.ifft(y_compl)
        y_re_2 = np.exp(-1j*self.dt/2*self.V)*y_re_2

        self.traj[:,num_step] = y_re_2
        self.norma[num_step] = np.sum(np.abs(self.traj[:,num_step])**2)
```

Listing 2: Metodo evoluzione

* Input: **num_step**.
* Compito: Procede a calcolare il valore di **traj** al timestep **num_step** a partire da quello a **num_step -1** come spiegato nel procedimento alla fine della sezione **Teoria** (Teoria). Calcola inoltre il valore di **norma** per il timestep corrente e lo inserisce nel vettore.
* Output: Anche questo metodo non ha output, tutto ció che viene calcolato é mantenuto nelle variabili di istanza.

Classe Animazione

La classe **Animazione** é composta da 4 attributi e 2 metodi (uno dei quali é il costruttore della classe). Riportiamo di seguito una breve descrizione dei loro scopi:

- Attributi:
  - **x** : Contiene il vettore delle coordinate x dei punti da graficare, ha lunghezza **N** (numero di step spaziali).
  - **y** : Contiene la matrice **traj** con l'evoluzione dell'onda, ha dimensioni (**N**,**M**).
  - **line**: é un oggetto di tipo Line2D (di *matplotlib*) che si puó utilizzare, aggiungendogli i dati da graficare, per comporre i vari frame.
  - **ax**: é un oggetto di tipo Axes (di *matplotlib*).Viene passato da **MainOnda** ed é legato al grafico che si sta cercando di animare.

- Metodi:
  - Costruttore (__init__):

```
def __init__(self,ax,wave):
        self.x = wave.x
        self.y = wave.traj
        self.line, = ax.plot([],[])
        self.ax = ax

        self.ax.set_xlim(0, wave.L)
        self.ax.set_ylim(0, 2.5)
```

Listing 3: Costruttore della classe **Animazione**

    * Input: **ax** (oggetto di **Matplotlib**),**wave** (oggetto di **Onda**).
    * Compito: Definisce le variabili di istanza elencate sopra. Per le prime due i valori vengono presi dagli attributi dell'oggeto **wave** passato in input. Imposta inoltre i limiti di visualizzazione degli assi del grafico finale.
    * Output: Non ha output, serve solo per memorizzare i valori utili per il grafico nella classe in modo che possano essere usati dal metodo **new_frame**.
  - New_frame :

```
def new_frame(self,num_frame):
        self.line.set_data(self.x,np.abs(self.y[:,num_frame]))
        return self.line,
```

Listing 4: Metodo new_frame

    * Input: **num_frame**.
    * Compito: Imposta come valori nel grafico quelli relativi al **num_frame** passato come input.
    * Output: Ritorna un oggetto di tipo Line2D che sará usato per il grafico in **MainOnda**.

MainOnda

MainOnda contiene la parte principale del programma. Puó essere suddivisa in alcune sezioni che vedremo piú nel dettaglio qui di seguito:

- Imports:

```
%matplotlib widget
from Onda import Onda
from Animazione import Animazione
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as an
import time
```

Listing 5: Import utili per il programma

In questa parte di codice importiamo le classi **Onda** e **Animazione** esposte in precedenza, **numpy** per gestire i vettori, **pyplot** e **animation** per gestire i grafici e **time** per tenere traccia dei tempi di esecuzione.

- Parametri iniziali:

```
#parametri del run e condizioni iniziali
L = 50.
T = 4
N = 4096
M = 1000
sigma = 1.5
k0 = 5e0
x0 = L/4
```

Listing 6: Parametri iniziali di integrazione e per la definizione del pacchetto d'onda

Qui definiamo i parametri di integrazione **L,T,N,M** descritti nella sezione **Attributi** della classe **Onda** e i parametri **sigma, k0 e x0** che descrivono la gaussiana che useremo come pacchetto d'onda per la propagazione.

- Definizione del potenziale:

```
#creazione potenziale V
s_i_1 = N//3 - 5        #step di inizio barriera 1 (incluso)
s_f_1 = s_i_1 + 10        #step di fine barriera 1 (escluso)
s_i_2 = N//2 - 5        #step di inizio barriera 2 (incluso)
s_f_2 = s_i_2 + 10        #step di fine barriera 2 (escluso)
int_1 = 10              #intensit barriera 1
int_2 = 100             #intensit barriera 2

V = np.zeros(N)
V[s_i_1:s_f_1] += int_1
V[s_i_2:s_f_2] += int_2
```

Listing 7: Definizione del potenziale a cui é soggetta l'onda

In questa sezione definiamo il potenziale a cui sará soggetta l'onda durante la propagazione. Creiamo un vettore **V** che contiene il valore del potenziale per ogni punto di griglia spaziale.

- Inizializzazione dell'oggetto:

```
#creazione oggetto
wave = Onda(L,T,N,M,V)
wave.traj[:,0] = (1./((2.*np.pi*(sigma**2))**(1/4)))*\
                    np.exp(-((wave.x[:]-x0)**2)/(4.*(sigma**2)))*\
                    np.exp(1j*k0*wave.x[:])

wave.norma[0] = np.sum(np.abs(wave.traj[:,0])**2)
```

Listing 8: Creazione dell'oggetto della classe **Onda**

Ora creiamo l'oggetto della classe **Onda** passando come argomenti **L,T,N,M e V**, inizializziamo la forma della nostra onda (**traj** a tempo 0) ad una gaussiana come spiegato nella teoria e calcoliamo la norma (che servirá in seguito).

- Evoluzione del sistema:

```
#evoluzione dell'onda
tic = time.process_time()
for i in range (1,M+1):
    wave.evoluzione(num_step = i)
print("time = ",time.process_time()-tic )
```

Listing 9: Eseguiamo i calcoli sull'evoluzione del sistema

Qui sfruttiamo il metodo **evoluzione** della classe **Onda** illustrato in precedenza (Classe **Onda**) per generare **M** step di evoluzione. Usiamo anche il metodo **process_time** di **time** per monitorare il tempo di esecuzione.

- Animazione:

```
#animazione
fig , ax = plt.subplots()
plt.step(wave.x,wave.V)
obj = Animazione(ax,wave)
anim = an.FuncAnimation(fig , obj.new_frame, frames=M, interval=10, blit=True)
plt.plot()
ax.set_ylim(0,0.75)
```

Listing 10: Creiamo l'animazione dell'evoluzione dell'onda

Finalmente produciamo la simulzione grazie al metodo **FuncAnimation** di **matplotlib.animation**([3]). Questo metodo chiama iterativamente **M** volte il metodo **new_frame** da noi creato nella classe **Animazione** passando come parametro un indice che é "in range(**M**)". Il procedimento dá vita ad un grafico che cambia autonomamente tra le configurazioni spaziali (identificate dal timestep) calcolate durante l'evoluzione.

- Modulo:

```
#grafico modulo
fig2 , ax2 = plt.subplots()
ax2.plot(wave.norma*wave.dx)
plt.show()
```

Listing 11: Grafichiamo il valore del modulo quadro dell'onda durante l'evoluzione

Controlliamo anche che, nonostante il pacchetto si apra come ci aspettiamo, esso rimanga sempre normalizzato a 1. Per fare questo basta graficare il vettore **norma** di cui abbiamo parlato in Classe **Onda**. Viene in realtá graficato **norma*dx** perché, a causa della natura quantizzata del problema, la norma é stata calcolata tramite una somma e non un integrale.

- Salva figure:

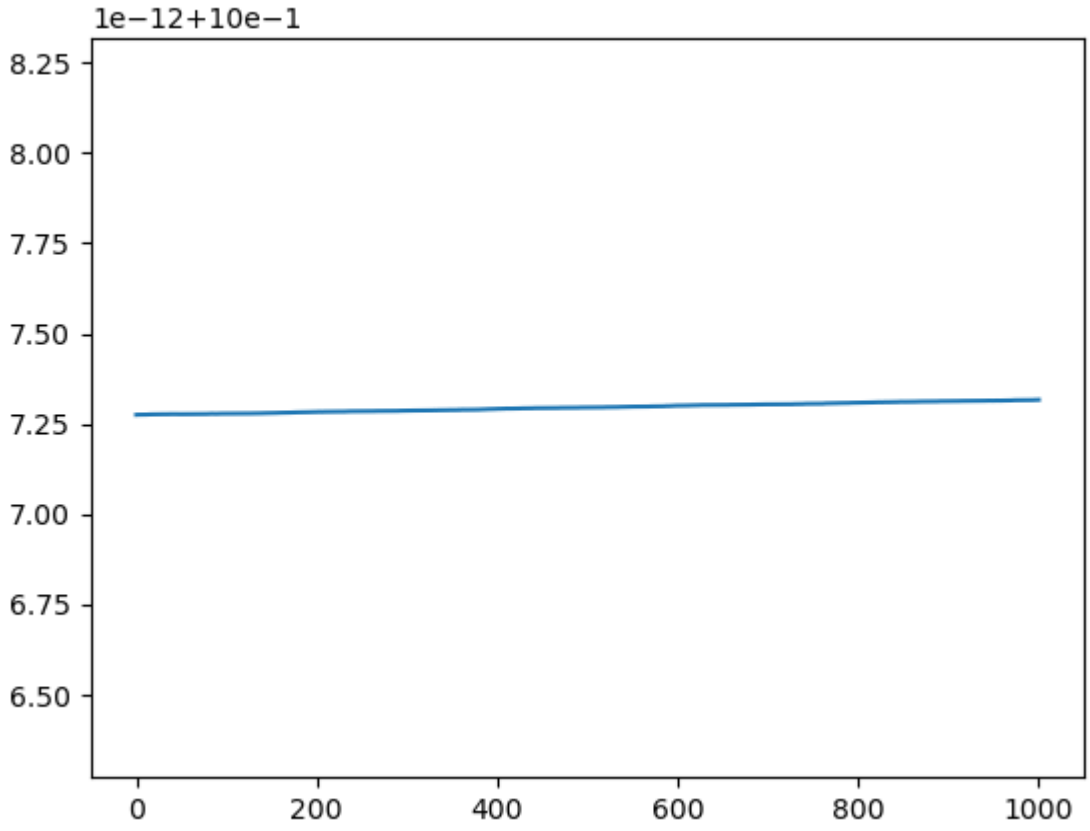
```
#save figures
save_times=[0,M//4,M//2]
for i in range(len(plot_times)):
    fig,ax = plt.subplots()
    ax.set_ylim(0,0.75)
    plt.step(wave.x,wave.V)
    plt.plot(wave.x,np.abs(wave.traj[:,plot_times[i]]))
    plt.savefig('nomegrafico_%d.png' %(i+1))
```

Listing 12: Salviamo delle immagini dell'animazione a determinati timestep

Infine salviamo i grafici dell'animazione ad alcuni timestep scelti per utilizzo esplicativo o dimostrativo in assenza del codice eseguibile.

Risultati[1]

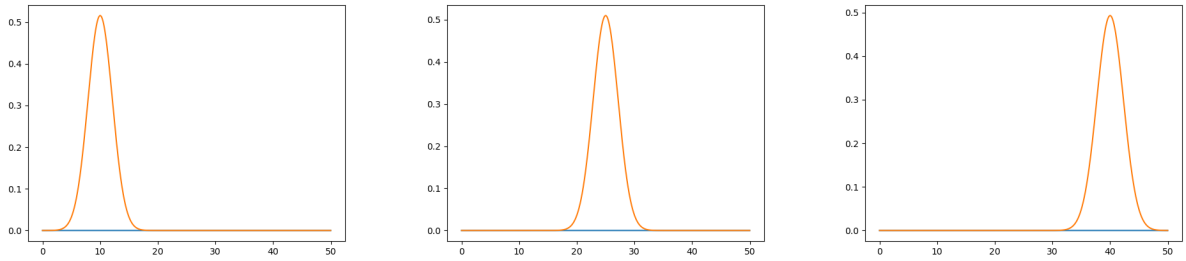
In questa sezione riporteremo qualche fermoimmagine (per impossibilitá di inserire video) delle simulazioni effettuate effettuate grazie al codice sviluppato (Il Codice[1]).
Si invita quindi il lettore interessato a provare le simulazioni proposte grazie al codice diponibile su GitHub ([1]): al fine di permettere questo utilizzo ogni immagine sará corredata di una tabella contenente i valori da dare ai parametri di simulazione. Inseriamo per primo il grafico che mostra l'andamento del modulo quadro dell'onda. Lo mettiamo all'inizio perché sembra non essere molto influenzato dalle diverse configurazioni che proveremo in seguito:



Come si puó notare il valore del modulo quadro si aggira sempre intorno all'1. Aumenta leggermente durante la simulazione, probabilmente a causa delle approssimazioni del calcolatore nei calcoli floating point, ma la variazione é insignificante (dell'ordine di 10^{-14}) rispetto al valore (≈ 1).

Passiamo ora alle simulazioni vere e proprie:

- Particella libera:
  Pariamo dal caso semplice di una particella libera per testare la qualitá della simulazione.



(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

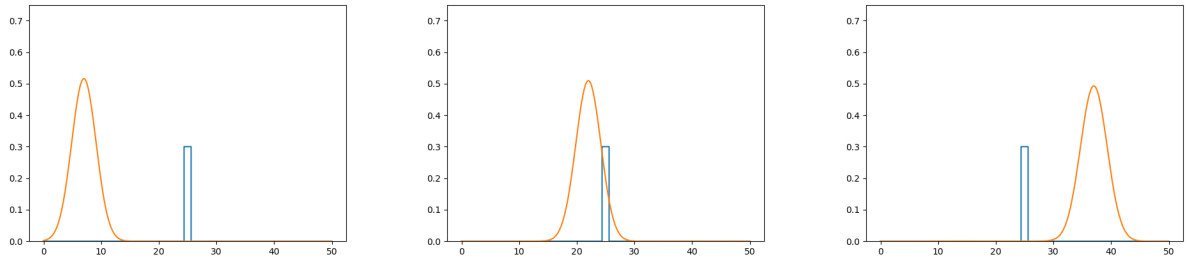
Figura 1: Posizione dell'onda ad alcuni timestep differenti, in assenza di potenziale

I parametri utilizzati per questa simulazione sono:

σ	k_0	x_0	int_1	int_2
1.5	15e0	7	0	0

Tabella 1: I primi 3 campi sono quelli visti nella sezione **Parametri iniziali** di **MainOnda**, i successivi due sono stati spiegati invece nella sezione **Definizione del potenziale**

- Singola barriera di potenziale:
  Tentiamo ora di studiare il comportamento della particella soggetta ad un potenziale del tipo barriera. Ci aspettiamo che, in base all'entitá della barriera, una porzione piú o meno grande dell'onda venga riflessa: proviamo quindi con vari valori di int_2.

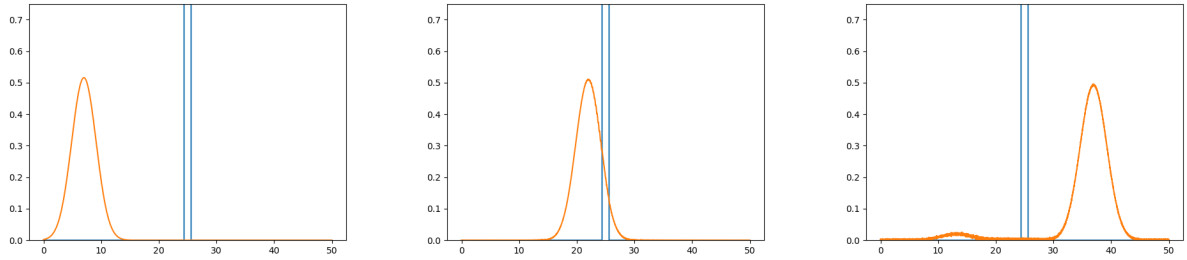


(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

Figura 2: Posizione dell'onda ad alcuni timestep differenti, con potenziale a barriera di intensitá 0.3

σ	k_0	x_0	int_1	int_2
1.5	15e0	7	0	0.3

Come si puó notare l'onda non avverte la presenza del potenziale. Questo avviene perché, nonostante le dimensioni sembrino comparabili, il grafico ci trae in inganno. Per onda e potenziale i valori sull'asse delle y non hanno lo stesso significato. Mentre il potenziale é effettivamente di 0.3 V, per l'onda i valori sulle y sono numeri puri (la normalizzazione deve dare 1). Se dovessimo confrontare in energia l'onda e il potenziale ci renderemmo conto che, siccome l'energia dell'onda é proporzionale a k^2, esse non sono comparabili.
Proviamo dunque con un'intensitá maggiore:



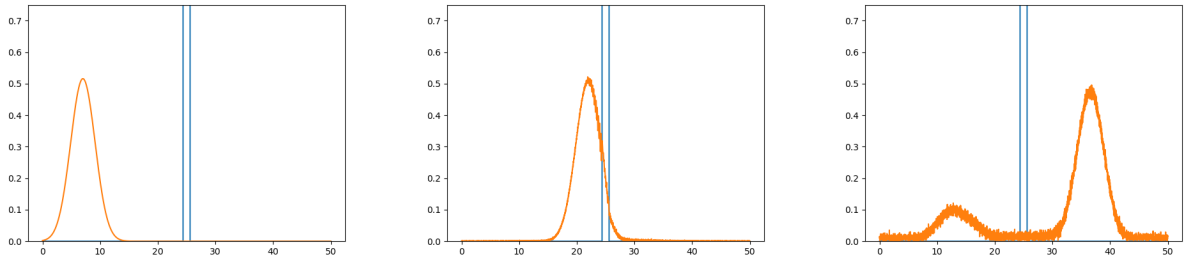
(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

Figura 3: Posizione dell'onda ad alcuni timestep differenti, con potenziale a barriera di intensitá 10

σ	k_0	x_0	int_1	int_2
1.5	15e0	7	0	10

In questo caso vediamo che l'onda interagisce con il potenziale e viene, almeno parzialmente, riflessa. Infatti, ritornando al discorso precedente, abbiamo mosso il potenziale di un intero

ordine di grandezza verso l'energia dell'onda stessa.
Proviamo ad aumentare ancora l'intensitá del potenziale:

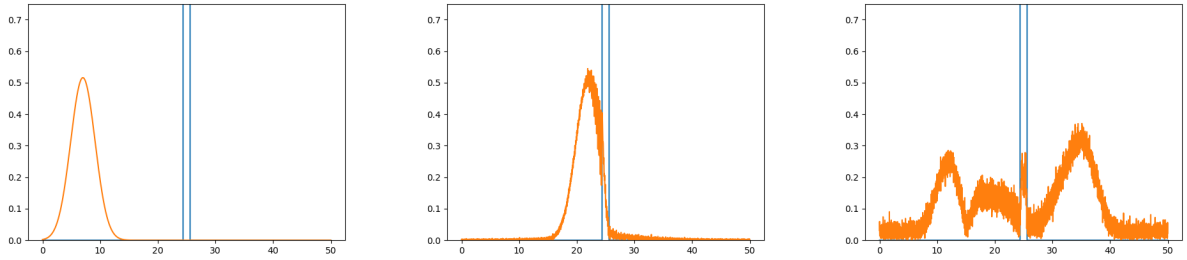


(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

Figura 4: Posizione dell'onda ad alcuni timestep differenti, con potenziale a barriera di intensitá 50

σ	k_0	x_0	int_1	int_2
1.5	15e0	7	0	50

Qui la riflessione é ancora piú visibile ma allo stesso tempo si inizia ad intuire un nuovo effetto con cui non si era ancora entrati in contatto: l'interferenza. Per vari motivi, tra cui i requisiti per poter calcolare le trasformate di Fourier, il nostro ambiente di integrazione é ciclico. Questo significa che quando l'onda si scontra con una delle due pareti esterne continua a propagarsi rientrando dall'altra parte. Purtroppo questo effetto fa sí che l'onda trasmessa interferisca con la sua parte riflessa, uscita da sinistra e rientrata da destra, rendendo "rumoroso" il segnale e peggiorando la nostra precisione di osservazione.
Aumentando ancora il valore dell'intensitá ci aspettiamo un segnale quasi incomprensibile:



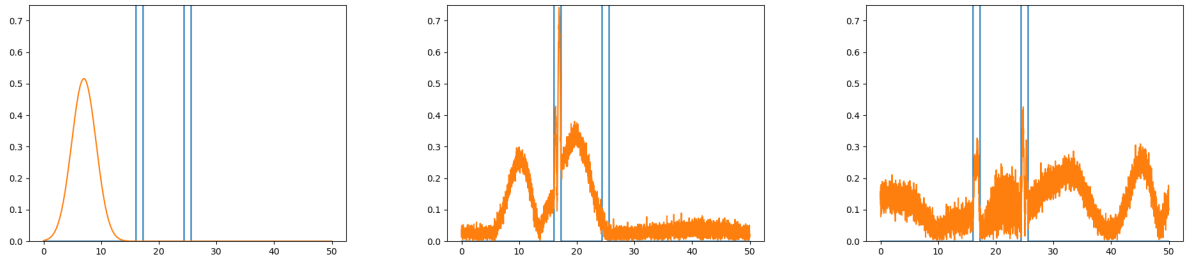
(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

Figura 5: Posizione dell'onda ad alcuni timestep differenti, con potenziale a barriera di intensitá 100

σ	k_0	x_0	int_1	int_2
1.5	15e0	7	0	100

Come ci aspettavamo la parte riflessa si avvicina sempre di piú a quella trasmessa ma purtroppo il rumore dovuto all'interferenza ci rende difficile discernere l'una dall'altra.

- Doppia barriera di potenziale:
  Proviamo ora a simulare l'evoluzione dell'onda in 2 potenziali di tipo barriera, scartiamo fin dall'inizio i primi valori di potenziale perché giá sappiamo che l'onda non interagisce particolarmente con bassi valori di potenziale.

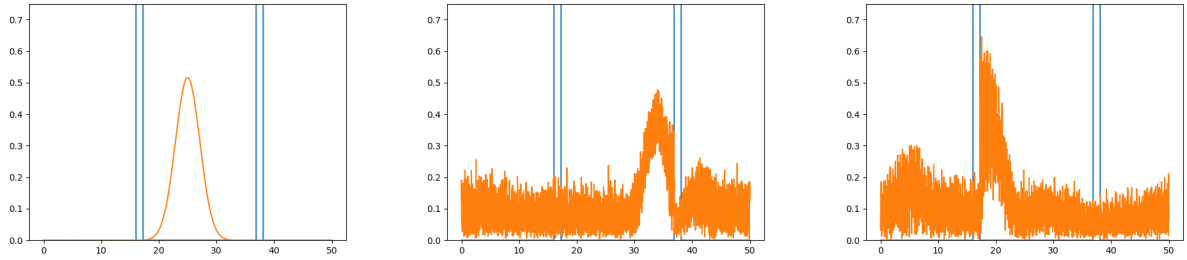


(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

Figura 6: Posizione dell'onda ad alcuni timestep differenti, con potenziale a doppia barriera entrambe di intensitá 100

σ	k_0	x_0	int_1	int_2
1.5	15e0	7	100	100

Si nota, nonostante il rumore d'interferenza, che una porzione di onda rimane intrappolata tra le 2 barriere di potenziale. Aumentando i valori di intensitá dovremmo intrappolare meglio l'onda all'interno dei potenziali ma cosí facendo la maggiorparte dell'onda verrá riflessa allo scontro col primo potenziale. Per ovviare a questo problema spostiamo l'origine dell'onda all'interno dei due potenziali in modo da non "perdere" nulla nel primo scontro.



(a) Timestep 0 (b) Timestep M//4 (c) Timestep M//2

Figura 7: Posizione dell'onda ad alcuni timestep differenti, con potenziale a doppia barriera di intensitá 10000

σ	k_0	x_0	int_1	int_2
1.5	15e0	25	10000	10000

Come si puó notare dal grafico é stato neccessario spostare il secondo potenziale a $\frac{3}{4}N$ (era precedentemente a $\frac{N}{2}$) per riuscire a far rientrare l'onda completamente tra i due potenziali. Il risultato prova quello che ci aspettavamo: tralasciando il rumore di interferenza, la maggior parte dell'onda é rimasta intrappolata all'interno dei due potenziali.

Conclusioni

Il programma sviluppato ha dato i risultati attesi e ci ha permesso di verificare le nostre ipotesi sul comportamento di un pacchetto d'onda in presenza di potenziali a barriera.

Bibliografia

[1] Federico Braidi. *GitHub Repository*. 2023. URL: https://github.com/LFC2022/propagazione-pacchetto-d-onda-quantistico-Kayen01.

[2] Mauro Ferrario e Guido Goldoni. *Dispense di Laboratorio di Fisica Computazionale*. 2021.

[3] MatplotlibDocumentation. *DocumentazioneFuncAnimation*. https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.FuncAnimation.html. [Online; accessed 10-May-2023]. 2023.

[4] NumpyDocumentation. *DocumentazioneFFTFREQ*. https://numpy.org/doc/stable/reference/generated/numpy.fft.fftfreq.html. [Online; accessed 10-May-2023]. 2023.

[5] Qiskit. *Trotter-Suzuki approximation*. https://qiskit.org/documentation/stubs/qiskit.synthesis.SuzukiTrotter.html. [Online; accessed 10-May-2023]. 2023.

[6] Wikipedia. *Unità naturali — Wikipedia, The Free Encyclopedia*. http://it.wikipedia.org/w/index.php?title=Unit%C3%A0%20naturali&oldid=126032393. [Online; accessed 10-May-2023]. 2023.