Università degli Studi di Napoli “Federico II”

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

Dipartimento di Fisica “Ettore Pancini”

Laurea triennale in Fisica

Risoluzione numerica dell’equazione di Schrödinger unidimensionale e indipendente dal tempo

Relatori:
Prof. Eleonora Messina

Candidato:
Lorenzo Della Volpe
Matricola N85000811

A.A. 2017/2018

Indice

Introduzione

Lo scopo della tesi è la risoluzione numerica dell'equazione di Schrödinger unidimensionale e indipendente dal tempo. L'equazione di Schrödinger unidimensionale si presenta nella forma

$$i\hbar\frac{\partial\overline{\psi}}{\partial t} = -\frac{\hbar^2}{2m}\frac{d^2\overline{\psi}}{dx^2} + V(x,t)\overline{\psi} \tag{0.1}$$

dove $\hbar = \frac{h}{2\pi}$ è detta costante di Planck ridotta, m è la massa della particella soggetta al potenziale $V(x,t)$ e la $\overline{\psi}(x,t)$ è detta $funzione\, d'onda$ il cui modulo quadro rappresenta la densità di probabilità di trovare la particella in una regione dello spazio al tempo t. Se nell'equazione di Schrodinger il potenziale non dipende dal tempo si può fattorizzare la $\overline{\psi}(x,t)$ in parte spaziale e temporale costruendola come

$$\overline{\psi}(x,t) = e^{-\frac{i}{\hbar}Et}\overline{\psi}(x) \tag{0.2}$$

che inserita nella (0.1) restituisce l'equazione per la parte spaziale della funzione d'onda

$$-\frac{\hbar^2}{2m}\frac{d^2\overline{\psi}(x)}{dx^2} + V(x)\overline{\psi}(x) = E\overline{\psi}(x) \tag{0.3}$$

Qui la $\overline{\psi}(x)$ è la funzione il cui modulo quadro rappresenta la densità di probabilità di trovare la particella in un punto dello spazio. L'equazione (0.3) è un problema agli autovalori, cioè ammette soluzione in una determinata classe ($\overline{\psi} \in L^2(R)$ delle funzioni a quadrato sommabile), solo quando l'energia E assume specifici valori numerici. Tali valori sono detti *autovalori* e le soluzioni $autofunzioni$. Si osservi che la (0.3) è risolubile in maniera esatta solo in pochi casi molto semplici come il caso dell'oscillatore armonico o l'atomo di idrogeno e, nei casi in cui non è possibile, si utilizzano metodi di approssimazione come i metodi perturbativi o variazionali. In questa tesi si sceglie un approccio numerico e si analizzano i metodi e le stategie implementative che permettono di fornire una simulazione affidabile del problema, nei casi in cui questo non sia risolubile esattamente. I metodi analizzati si basano sulle tecniche di Shooting ed utilizzano il metodo di Numerov per la discretizzazione del problema. Nel ricavare le soluzioni è stato osservato come il codice presentava dei comportamenti patologici che risultavano in una divergenza non fisica del risultato. Questo problema è stato quindi

risolto con un nuovo codice, che elimina i problemi di instabilità. Al fine di verificare il corretto funzionamento per altri potenziali, si è risolto il problema di una particella in una buca di potenziale finita, problema per cui non è nota la soluzione analitica, e su considerazioni teoriche si è verificata la validità delle approssimazioni. Questo lavoro si divide in 4 capitoli:

1. Equazione di Schrödinger in potenziale armonico: in questa sezione si risolve l'equazione di Schrödinger unidimensionale in potenziale armonico le cui soluzioni analiticamente trovate saranno il banco di prova per il codice scritto.

2. Risoluzione numerica di equazioni differenziali del secondo ordine: qui si introducono concetti generali di risoluzione numerica delle equazioni differenziali del secondo ordine specializzandoli, infine, al metodo di Numerov.

3. Un codice per la risoluzione dell'equazione di Schrödinger: qui si presenta il codice scritto per la risoluzione dell'equazione di Schrödinger unidimensionale a cui si aggiunge un paragrafo di sperimentazione numerica con altri potenziali e un paragrafo di analisi dell'errore del codice scritto.

Capitolo 1

Equazione di Schrödinger in potenziale armonico

Le soluzioni della (0.3) che si andranno a considerare apparterranno all'insieme delle funzioni a quadrato sommabile $L^2(R)$ le cui corrispondenti energie E si dicono appartenenti allo *spettro discreto*. Dato che il modulo quadro delle autofunzioni ha significato probabilistico, ovvero essendo

$$|\overline{\psi}(x)|^2 dx \tag{1.1}$$

la probabilità di trovare la particella in un intervallo $[x, x + dx]$ e quindi essendo

$$\int_R |\overline{\psi}(x)|^2 dx \tag{1.2}$$

la probabilità di trovare la particella in tutto lo spazio, c'è bisogno che l'integrale (1.2) restituisca 1 e quindi che la $\overline{\psi}(x)$ sia *normalizzata*. Per assicurarsi che le autofunzioni dell'equazione (0.3) siano normalizzate, e che quindi siano coerenti con un interpretazione probabilistica, in luogo della soluzione $\overline{\psi}(x)$ si usa moltiplicare la soluzione per una *costante di normalizzazione* del tipo $\frac{1}{\sqrt{C}}$, dove la costante C è definita da

$$C = \int_R \overline{\psi}^* \overline{\psi} dx \tag{1.3}$$

ottenendo quindi una soluzione normalizzata

$$\psi(x) = \frac{1}{\sqrt{C}} \overline{\psi}(x) \tag{1.4}$$

1.1 L'oscillatore armonico quantistico

In questo capitolo si considera il caso di potenziale armonico,ovvero un potenziale della forma

$$V(x) = \frac{1}{2}mw^2x^2 \tag{1.5}$$

Il potenziale armonico si ritrova spesso in letteratura poichè si può mostrare come un potenziale arbitrario può essere sempre approssimato da uno di tipo armonico nelle vicinanze di un punto di equilibrio. Un esempio può essere il potenziale in cui vivono gli atomi appartenenti ad una molecola biatomica che oscillano attorno alla posizione di equilibrio. L'equazione di Schrodinger in potenziale armonico è risolubile analiticamente e quindi sarà possibile verificare la bontà dell'approssimazione numerica confrontandola con la soluzione esatta. Per trattare l'equazione di Schrödinger in potenziale armonico è conveniente considerare il caso adimensionale. Tra i motivi per una trattazione adimensionale del problema è che si vuole evitare di trattare le costanti tipiche di un problema quantistico che, essendo più piccole dell'epsilon macchina del calcolatore utilizzato per le simulazioni ($epsilon = 2.2204 * 10^{-16}$), complicherebbero la trattazione numerica. Per porre l'equazione di Schrödinger in una forma adeguata, la si riscrive utilizzando le relazioni:

$$\xi = (\frac{mw}{\hbar})^{\frac{1}{2}}x \tag{1.6}$$

e quindi definendo l'energia come:

$$\varepsilon = \frac{E}{\hbar w} \tag{1.7}$$

Inserendo queste variabile nell'equazione di Schrödinger se ne ottiene una versione adimensionale:

$$\frac{d^2\psi}{d\xi^2} = (\xi^2 - 2\varepsilon)\psi(\xi) \tag{1.8}$$

1.2 Soluzione esatta dell'equazione di Schrödinger

Avendo ora l'equazione di Schrödinger adimensionale si vuole derivarne la soluzione analitica e per farlo, si utilizzerà il metodo di

Frobenius per trovare la soluzione in termini di serie di potenze.
Si comincia vedendo come per $\xi \to \infty$ si può trascurare ε ottenendo un equazione della forma

$$\frac{d^2\psi}{d\xi^2} = \xi^2\psi(\xi) \tag{1.9}$$

da cui si comprende come la funzione soluzione della (1.9) deve comportarsi per $\xi \to \infty$ come:

$$\psi(\xi) \sim \xi^n e^{\pm\frac{\xi^2}{2}} \tag{1.10}$$

con n arbitrario. Nella (1.10) si deve scartare le soluzione con il segno +, essendo questa divergente e non normalizzabile. Da questa soluzione asintotica si estrapola il comportamento per ξ finiti considerando una soluzione del tipo

$$\psi(\xi) = H(\xi)e^{-\frac{\xi^2}{2}} \tag{1.11}$$

con la condizione sulla funzione $H(\xi)$ di non dover crescere più velocemente di e^{ξ^2}, così da non dare soluzioni divergenti e quindi non normalizzabili. Con questa soluzione non resta che trovare la funzione incognita $H(\xi)$ e per farlo si inserisce la (1.11) nella (1.8) ottenendo:

$$H''(\xi) - 2\xi H'(\xi) + (2\varepsilon - 1)H(\xi) = 0 \tag{1.12}$$

Ora si può espandere in serie la funzione $H(\xi)$, supposto che $H(\xi)$ sia una funzione analitica

$$H(\xi) = \sum_{n=0}^{\infty} A_n \xi^n \tag{1.13}$$

Una volta inserita la (1.13) nell'equazione (1.12) e raccogliendo tutti i termini con la stessa potenza di n si ottiene:

$$\sum_{n=0}^{\infty}[(n+2)(n+1)A_{n+2} + (2\varepsilon - 2n - 1)A_n]\xi^n = 0 \tag{1.14}$$

che si vuole sia soddisfatta per ogni valore di ξ e quindi si impone l'annullarsi di tutti i coefficienti

$$(n+2)(n+1)A_{n+2} + (2\varepsilon - 2n - 1)A_n = 0 \tag{1.15}$$

Ora si può vedere come la serie non può essere infinita ma deve annullarsi dopo un certo n. Infatti, analizzando il comportamento dei coefficienti per grandi n, si vede che questi si comportano come $\frac{A_{n+2}}{A_n} \to \frac{2}{n}$ e quindi $A_{n+2} \to \frac{1}{(n/2)!}$. Ricordando la forma dell'espansione in serie della funzione esponenziale si vede come i coefficienti della (1.14) danno un contibuto alla serie che va come e^{ξ^2} producendo quindi soluzioni divergenti. L'unico modo per impedire questo è dunque annullare tutti i coefficienti dopo un certo n, facendo diventare la serie un polinomio di grado n finito. Guardando all'equazione (1.14) si vede che questo è possibile solo per

$$\varepsilon = n + \frac{1}{2} \tag{1.16}$$

e quindi ricordando l'espressione (1.7) dell'energia, si ottiene che le energie ammesse per l'oscillatore armonico sono quantizzate, cioè hanno la forma (dimensionale):

$$E_n = \hbar w(n + \frac{1}{2}) \tag{1.17}$$

I polinomi $H_n(\xi)$ corrispondenti sono detti polinomi di Hermite. La forma finale delle autofunzioni dell'oscillatore armonico quantistico è

$$\psi_n(\xi) = H_n(\xi)e^{-\frac{\xi^2}{2}} \tag{1.18}$$

Il polinomio di Hermite di grado n ha n nodi ed è pari per n pari ($H_n(-\xi) = H_n(\xi)$) e dispari per n dispari ($H_n(-\xi) = -H_n(\xi)$) e dalla forma della soluzione si ha che, essendo questa data dal polinomio di Hermite $H_n(\xi)$ moltiplicato per un esponenziale $e^{-\frac{\xi^2}{2}}$ che non presenta nodi, essa acquisirà lo stesso comportamento di $H_n(\xi)$.

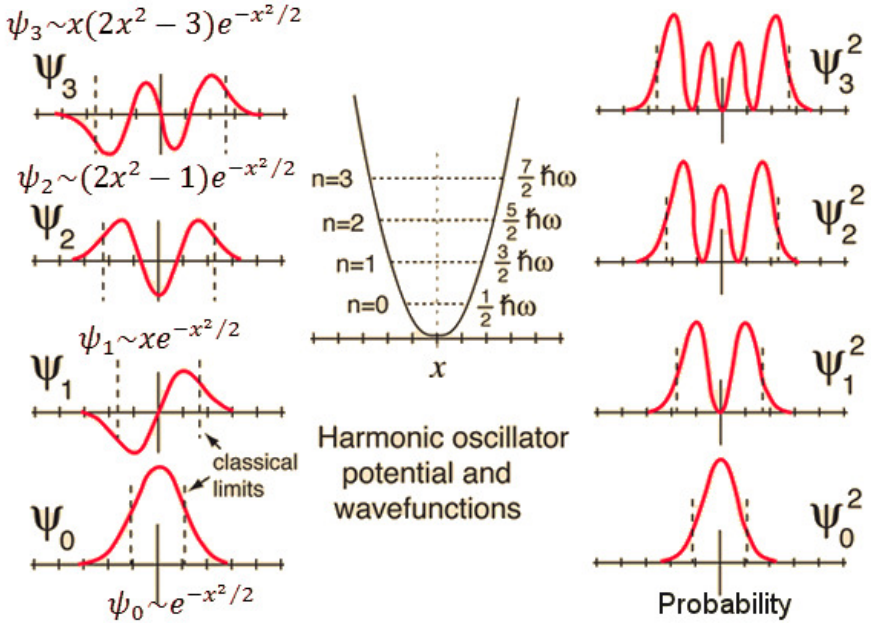

Figura 1.1: Prime 4 autofunzioni dell'oscillatore armonico con le corrispondenti densità di probabilità
[1]

Ora si dimostrano dei teoremi che saranno utili in seguito.

Theorem 1.2.1 *(Teorema di non degenerazione) Data una generica equazione di Schrodinger unidimensionale e un' energia E_n appartenente allo spettro discreto, allora non si potrà avere più di una soluzione corrispondente alla stessa energia.*

Dimostrazione: Si procede per assurdo supponendo di avere due soluzioni $\psi_1(x), \psi_2(x)$ corrispondenti ad una stessa energia E_n, e che esse siano linearmente indipendenti. Scrivendo l'equazioni di Schrodinger per queste due soluzioni si ottiene

$$\frac{\psi_1''}{\psi_1} = -\frac{2m}{\hbar^2}(V(x) - E_n) \tag{1.19}$$

$$\frac{\psi_2''}{\psi_2} = -\frac{2m}{\hbar^2}(V(x) - E_n) \tag{1.20}$$

Sottraendo la (1.19) alla (1.20) si ottiene

$$\frac{\psi_1''}{\psi_1} = \frac{\psi_2''}{\psi_2} \tag{1.21}$$

che si può riscrivere come

$$\frac{d}{dx}(\psi_2'\psi_1 - \psi_1'\psi_2) = 0 \tag{1.22}$$

Essendo la derivata nulla, la quantità tra parentesi deve essere uguale ad una costante C

$$\psi_2'\psi_1 - \psi_1'\psi_2 = C \tag{1.23}$$

Dato che si sta supponendo che le soluzioni ψ_1, ψ_2 corrispondano ad un energia dello spettro discreto, queste devono essere normalizzabili e quindi

$$\lim_{x\to\infty} \psi_1 = 0$$
$$\lim_{x\to\infty} \psi_2 = 0$$

Questo implica che la costante C, dovendo la (1.23) valere $\forall x \in [-\infty, \infty]$, non può che essere zero e quindi

$$\psi_2'\psi_1 - \psi_1'\psi_2 = 0 \tag{1.24}$$

Questa è un equazione differenziale separabile la cui soluzione è

$$\psi_2(x) = K\psi_1(x) \tag{1.25}$$

con K costante. L'ultimo risultato implica l'assurdo dell' ipotesi iniziale avendo imposto che ψ_1 e ψ_2 fossero linearmente indipendenti. □

Questo teorema è utile per dimostrare come non si avranno problemi di degenerazione quando si andrà ad applicare il metodo numerico.

Theorem 1.2.2 *[3] (Teorema di Oscillazione) Numerando gli autovalori dello spettro discreto come E_n $(n = 1, 2...)$ con $E_n < E_{n+1}$, l'autofunzione ψ_n corrispondente all'n-esimo autovalore si annulla in $n-1$ punti (ha quindi $n-1$ nodi).*[1]

[1] Qui si utilizza n=1 per lo stato fondamentale in luogo dell'usuale n=0

Dimostrazione: Dato un generico potenziale $V(x)$ possiamo costruire una nuova famiglia di potenziali $V_a(x)$, cosi che

$$\begin{cases} V_a(x) = V(x) \ per \ |x| < a \\ V(x) = \infty \ per \ |x| > a \end{cases} \tag{1.26}$$

per $a = \varepsilon$ abbastanza piccolo abbiamo una buca di potenziale infinita le cui autofunzioni sono note e sono $\psi_k^o(x;\varepsilon) \propto sin(\frac{k\pi x}{\varepsilon})$ per parità dispari con $k = 1, 2, 3, ...$ e $\psi_k^e(x;\varepsilon) \propto cos(\frac{(2k+1)\pi x}{2\varepsilon})$ per parità pari con $k = 0, 1, 2....$ Data quindi la forma delle soluzioni si ha che ψ_n ha $n-1$ nodi per $|x| < a$.
Se si tratta il caso dello stato fondamentale, che si può assumere sia positivo poichè non ha nodi,si ha $\psi_1'(-\varepsilon, \varepsilon) > 0$ e $\psi_1'(\varepsilon, \varepsilon) < 0$. Ora, se si immagina di aumentare a la funzione d'onda $\psi_1(x;a)$ diventerà sempre più simile a quella che descrive il vero stato fondamentale del potenziale $V(x)$ e quindi si può pensare che $\psi_1(x;a)$ possa sviluppare un nodo tra $\pm a$.
Se l'autofunzione dello stato fondamentale sviluppa effettivamente un nodo, questo può avvenire in solo due modi: (1) almeno una delle derivate a $\pm a$ cambia segno o (2) le derivate non cambiano segno ma l'autofunzione si annulla in due punti (sviluppa quindi 2 zeri). In entrambi casi ci sarà un valore critico di a tale che l'autofunzione e la sua derivata prima svaniscano nello stesso punto.
Nel caso (1) l'autofunzione e la sua derivata si annullano in uno dei punti al bordo poichè le condizioni al bordo di una buca di potenziale infinita, implicano l'annullarsi dell'autofunzione e il cambio di segno della derivata implica che questa deve essere zero per il valore critico di a. Nel caso (2) ci deve essere un valore di a tale che l'autofunzione tocchi l'asse subito prima di sviluppare i due nodi e quindi, in quel punto, sarà nulla con la sua derivata. Ora dato che l'equazione di Schrodinger soddisfa i criteri di esistenza e unicità,dato il valore dell'autofunzione e della derivata in un punto, la soluzione deve essere unica.
Nei due casi analizzati, come si è detto inizialmente, si ha che l'autofunzione e la sua derivata si annullano nello stesso punto il che porterebbe ad avere come soluzione per l'equazione di Schrodinger $\psi = 0$. Dato che in questa trattazione si suppone che esista sempre una soluzione non banale dell'equazione di Schrodinger, ψ_1 e ψ_1' non possono annullarsi nello stesso punto e quindi non si può avere lo sviluppo di un nodo. In generale si ha che, partendo dalle autofun-

zioni della buca di potenziale con $n-1$ nodi, queste non possono svilupparne altri. I risultati precedenti implicano che all'aumentare dell'intervallo $[-a,+a]$ l'autofunzione della buca di potenziale che inzialmente ha $n-1$ diventerà l'autofunzione del vero potenziale, senza però diminuire o aumentare i nodi.

Nella trattazione di altri potenziali è utile un teorema che qui si enuncerà senza dimostrazione:

Theorem 1.2.3 *Se il potenziale $V(x)$ è limitato inferiormente da V_0 e soddisfa*

$$\lim_{x\to\pm\infty} V(x) = V_{\pm}$$

si ha che i valori dello spettro discreto(se esistono) sono vincolati a rispettare

$$V_0 < E_n < min(V_+, V_-) \tag{1.27}$$

Da questo teorema si può vedere come nel caso del potenziale armonico, essendo questo divergente per $x \to \pm\infty$ e inferiormente limitato da $V_0 = 0$, ci saranno infinite energie nello spettro discreto.

Un ultima utile definizione è quella di punto di inversione classica, ovvero quel punto x_{icl} per cui vale la condizione

$$V(x_{icl}) = E_n \tag{1.28}$$

Si può dimostrare come oltre questo punto l'autofunzione dell'equazione di Schrodinger non potrà avere nodi, ma solo comportamento crescente o decrescente.

Capitolo 2

Risoluzione numerica di equazioni differenziali del secondo ordine

Un equazione differenziale ordinaria scalare del primo ordine è data da

$$y' = f(x, y) \tag{2.1}$$

in cui y' denota la derivata prima di $y(x)$ e la funzione $f(x, y)$, che dipende da x e $y(x)$, rappresenta il campo vettoriale. L'equazione (2.1) può avere un numero infinito di soluzioni. La soluzione può essere univocamente determinata se si fissa una condizione iniziale

$$y(a) = y_0 \tag{2.2}$$

Si dirà che l'equazione differenziale, insieme con le condizioni iniziali, costituisce un problema ai valori iniziali.

Equazioni differenziali di ordine superiore si presentano nella forma:

$$y^d = f(x, y(x), y'(x), ..., y^{d-1}(x)) \tag{2.3}$$

In questo caso occorre imporre d condizioni che possono essere date nello stesso punto a (allora si ha un problema ai valori iniziali) o in punti diversi (problemi ai valori al contorno)

In generale, le equazioni di ordine superiore posso essere ridotte ad un sistema del primo ordine tramite le sostituzioni:

$$\begin{cases} y_1(x) = y(x) \\ y_2(x) = y'(x) \\ ... \\ ... \\ ... \\ y_d = y^{d-1}(x) \end{cases} \tag{2.4}$$

ottenendo un sistema del primo ordine della forma

$$\begin{cases} y_1'(x) = y_2(x) \\ ... \\ ... \\ ... \\ y_{d-1}' = y_d \\ y_d'(x) = f(x, y_1(x), ..., y_d(x)) \end{cases} \tag{2.5}$$

Il seguente teorema, qui enunciato senza dimostrazione, fornisce condizioni sufficienti per l'esistenza di un' unica soluzione di un problema ai valori iniziali per un sistema di equazioni differenziali.

Theorem 2.0.1 *Sia f(x,y) definita e continua nella striscia infinita* $S = \{(x, y) : -\infty < a \leq x \leq b < \infty, y \in \mathbb{R}^m\}$ *Supponiamo inoltre che la funzione f(x,y) sia, in S, lipschitziana nella variabile y, uniformemente rispetto x, ovvero esista una costante* $L > 0$ *tale che* $\|f(x, y_1) - f(x, y_2)\| \leq L \|y_1 - y_2\| \, \forall x \in [a, b]$ *e* $\forall$ *coppia* $y_1, y_2 \in \mathbb{R}^m$. *Allora* $\forall x \in [a, b]$ *e* $\forall y_0 \in \mathbb{R}^m$ *esiste esattamente un unica funzione y(x) tale che:*

- $y(x) \in \mathbb{C}^1[a, b]$
- $y'(x) = f(x, y(x)) \ \forall x \in [a, b]$
- $y(a) = y_0$

Nel seguito si supporrà sempre che tali condizioni siano soddisfatte. La trattazione nei seguenti paragrafi è stata tratta dai libri [5] [4]

2.1 Trattamento numerico di problemi ai valori iniziali per equazioni differenziali

Si consideri un problema ai valori iniziali

$$\begin{cases} y' = f(x,y) \\ y(a) = y_0 \quad x \in [a,b] \end{cases} \tag{2.6}$$

Dato che nella trattazione successiva ci si troverà a risolvere solo equazioni scalari, si tratterà solo questo tipo di problemi. I metodi numerici che saranno descritti in questo capitolo si basano sull' idea di discretizzazione, ovvero si sostituisce all'intervallo continuo $[a,b]$ un insieme discreto di punti

$$x_n = a + nh \quad n = 0, 1, ..., N \tag{2.7}$$

con

$$h = \frac{b-a}{N} \tag{2.8}$$

detto passo di discretizzazione.
Per capire intuitivamente come i metodi numerici riescano a fornire una soluzione approssimata si presenta qui il metodo di Eulero, che permette di illustrare i concetti generali dei metodi numerici senza dover trattare l'algebra complessa delle tecniche più avanzate.

2.1.1 Metodo di Eulero

Il metodo di Eulero è il metodo più elementare per risolvere equazioni differenziali del primo ordine. Si comincia con il definire l'equazione differenziale da risolvere, che sarà nella forma:

$$\begin{cases} \frac{dy}{dx} = f(x,y) \\ y(a) = \alpha \end{cases} \tag{2.9}$$

con $[a,b]$ l'intervallo in cui si vuole trovare la soluzione. L'intervallo $[a,b]$ dovrà essere discretizzato, e questo è fatto scegliendo un N e selezionando i punti della griglia come

$$x_i = a + ih \quad i = 0, 1, ..., N-1 \tag{2.10}$$

e h, dato da:

$$h = \frac{b-a}{N} = x_{i+1} - x_i \tag{2.11}$$

Il metodo di Eulero si basa sull'idea di approssimare la curva soluzione di (2.9) con una curva poligonale che connette i punti $y_0, ..., y_n$. Per trovare una formula iterativa da poter utilizzare si comincia con l'approssimare la derivata della funzione $y(x)$ tramite il rapporto incrementale tra due punti consecutivi della griglia

$$y'(x_n) \approx \frac{y(x_{n+1}) - y(x_n)}{h} \tag{2.12}$$

Guardando ora alla forma del problema (2.9) si sostituisce $f(x_n, y_n)$ nella (2.12) ottenendo

$$\frac{y(x_{n+1}) - y(x_n)}{h} \approx f(x_n, y_n) \tag{2.13}$$

e quindi la forma finale del metodo di Eulero:

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{2.14}$$

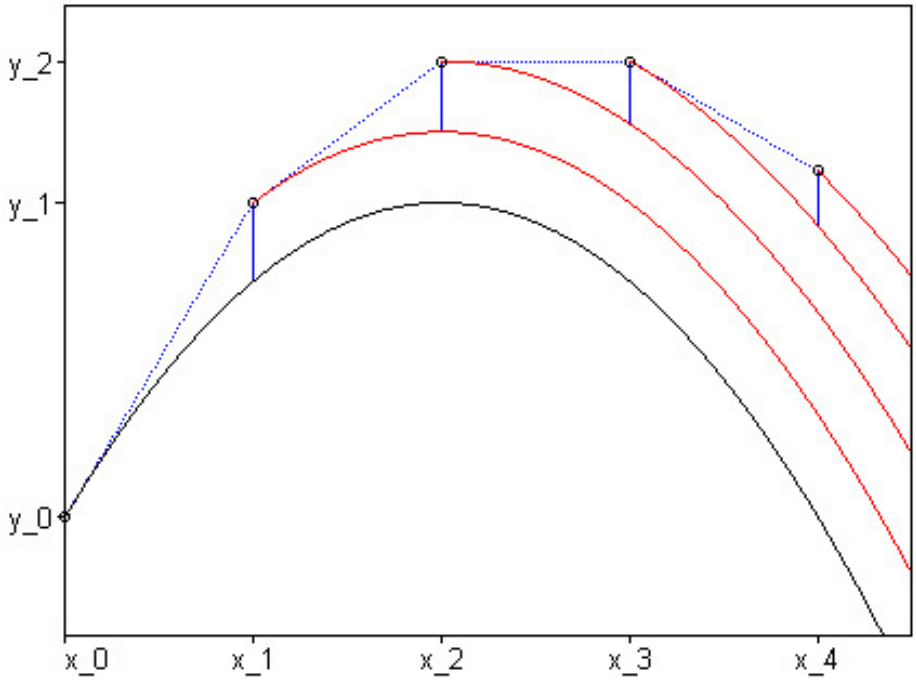

Figura 2.1: Primi tre passi del metodo di Eulero. La poligonale tratteggiata è l'approssimazione tramite il metodo di Eulero della soluzione reale (in nero). [2]

Come si può vedere dal grafico il metodo di Eulero approssima la soluzione di (2.9) con una poligonale che connette i punti $y_0, y_1, ..., y_n$. Intuitivamente si può comprendere che la sorgente principale di errore sia l'utilizzare per l'approssimazione y_{n+1} la tangente al punto y_n, che è a sua volta un approssimazione della soluzione reale generando quindi un distaccamento dalla soluzione reale per via dell'accumularsi di questo errore ad ogni step del metodo.

2.2 Metodi multistep lineari

Per ottenere una migliore approssimazione rispetto al metodo di Eulero si sono sviluppate tecniche che utilizzano più punti precedenti al punto da calcolare. Una classe di metodi che utilizzano questo approccio sono i metodi multistep lineari.
Sia y_n un approssimazione di y nel punto x_n ($y_n \simeq y(x_n)$) allora, la sequenza di valori y_n che approssima la soluzione di (2.6) nell'insieme discreto di punti x_n, costituisce la soluzione numerica del problema. La sequenza y_n può essere calcolata numericamente, supponendo noti i valori iniziali $y_0, y_1, ..., y_{k-1}$, attraverso il metodo multistep lineare definito da:

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(x_{n+j}, y_{n+j}) \tag{2.15}$$

dove α_j e β_j sono costanti. Si suppone che $\alpha_k = 1$ e α_0 e β_0 non siano entrambi nulli. Il metodo (2.15) si dice ad un passo ($one - step$) se $k = 1$ e a più passi ($multi - step$) se $k > 1$. Inoltre si dirà che il metodo è esplicito se $\beta_k = 0$ ed implicito se $\beta_k \neq 0$. Per un metodo esplicito il valore y_{n+k} si determina direttamente in termini di y_{n+j} e $f(x_{n+j}, y_{n+j})$ per $j = 0, ..., k-1$ che sono stati già calcolati nei passi precedenti. Per un metodo implicito, invece, occorre ad ogni passo risolvere un equazione che, se la f non è lineare, richiede l'uso di metodi iterativi.

Per un qualsiasi metodo numerico ci si aspetta che l'approssimazione migliori al diminuire del passo h e all'aumentare dei punti n in cui si discretizza l'intervallo. Si può quindi definire la *convergenza* come

Definition 2.2.1 *Il metodo* (2.15) *è detto convergente se, per ogni problema ai valori iniziali, abbiamo che*

$$\max_{0\leq n\leq N} |y(x_n) - y_n| \to 0 \ \ per \ \ \ h \to 0 \tag{2.16}$$

Si noti che i valori iniziali $y_0, y_1, ..., y_{k-1}$ devono convergere a y_0.
Per studiare l'accuratezza di un metodo si definisce il residuo, ovvero la quantità a meno della quale la soluzione esatta soddisfa l'operatore discreto:

$$R(y(x_{n+k}), h) \coloneqq \sum_{j=0}^{k} \alpha_j y(x_{n+j}) - h\beta_j f(x_{n+j}, y_{n+j}) \tag{2.17}$$

e si definisce la *consistenza* e l'*ordine di consistenza* del metodo come:

Definition 2.2.2 *Il metodo definito dalla* (2.15) *è consistente di ordine p, intero positivo, se risulta*

$$R(\cdot) = O(h^{p+1}) \tag{2.18}$$

con $p \geq 1$

Si dà quindi un ultima definizione del concetto di zero-stabilità

Definition 2.2.3 *Siano* $\delta_n, n = 0, 1, ..., N$ *e* $\delta_n^*, n = 0, 1, ...N$ *due perturbazioni numeriche, e siano* $z_n, n = 0, 1, ..., N$ *e* $z_n^*, n = 0, 1, ..., N$ *le corrispondenti soluzioni perturbate trovate tramite l'applicazione di* (2.15)*. Se esistono le costanti* S *e* h_0 *tali che* $\forall h \in (0, h_0]$ *si ha*

$$|z_n - z_n^*| \leq S\epsilon \ \ 0 \leq n \leq N \tag{2.19}$$

e

$$|\delta_n - \delta_n^*| \leq \epsilon \ \ 0 \leq n \leq N \tag{2.20}$$

diciamo che il metodo (2.15) *è zero − stabile*

Le perturbazioni nella definizione precedente possono essere interpretate come perturbazioni dovute a errori di round-off, sempre presenti quando si usa un calcolatore. La zero-stabilità è una proprietà del metodo numerico che assicura che questo risulti robusto rispetto alle perturbazioni.

2.3 Trattamento numerico di equazioni del secondo ordine

Equazioni differenziali del secondo ordine possono essere trasformate in un sistema del primo ordine ed integrate con i metodi classici. Un approccio più diretto consiste nel costruire metodi adattati alla forma del problema. Questo approccio risulta particolarmente vantaggioso quando si ha un equazione in cui non compare la derivata prima. Poichè questo è il caso dell'equazione di Schrödinger, si esamina qui la classe di metodi lineari multistep in analogia a quanto fatto nel paragrafo precedente per equazioni del primo ordine.

2.3.1 Metodi multistep per equazioni del secondo ordine

Si consideri il problema del secondo ordine di forma:

$$\begin{cases} y'' = f(x, y) \\ y(a) = y_0 \\ y'(a) = y'_0 \end{cases} \tag{2.21}$$

un metodo numerico lineare a k passi si potrà scrivere, in analogia alla (2.15), come

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h^2 \sum_{j=0}^{k} \beta_j f(x_{n+j}, y_{n+j}) \tag{2.22}$$

con α, β costanti e in particolare $\alpha_k = 1$ e α_0 e β_0 che non svaniscono entrambi. Intuitivamente si capisce come k dovrà essere almeno $k = 2$ dovendo almeno avere tre valori di y per poter approssimare y''. Per questa classe di metodi si può definire la consistenza specializzando la definizione (2.17), come:

$$R(y(x_{n+k}), h) = \sum_{j=0}^{k} \alpha_j y_{n+j} - h^2 \beta_j f(x_{n+j}, y_{n+j}) \tag{2.23}$$

e quindi la definizione (2.2.2) diventa:

Definition 2.3.1 *Il metodo definito dalla* (2.22) *è consistente di ordine p, intero positivo, se risulta*

$$R(\cdot) = O(h^{p+2}) \tag{2.24}$$

con $p \geq 1$

Si danno ora delle condizioni che devono valere per far si che un metodo sia consistente. Associando al metodo (2.22) i due polinomi caratteristici definiti da

$$\begin{cases} \varrho(\zeta) = \sum_{j=0}^{k} \alpha_j \zeta^j \\ \sigma(\zeta) = \sum_{j=0}^{k} \beta_j \zeta^j \end{cases} \tag{2.25}$$

si può dimostrare come (2.22) è consistente se e solo se vale:

$$\begin{cases} \varrho(1) = \varrho'(1) = 0 \\ \varrho''(1) = 2\sigma(1) \end{cases} \tag{2.26}$$

La definizione (2.2.1) di convergenza non viene modifica se non con l'aggiunta della condizione che i k punti iniziali $y_k = y_k(h)$ $k = 0, 1, ...$ soddisfino

$$\lim_{h \to 0} \frac{[y_k(h) - y_0(h)]}{h} = y_0'$$

La definizione (2.2.3) di zero stabilità si specializza nel seguente modo:

Definition 2.3.2 *Il metodo* (2.22) *è detto zero-stabile se nessuna radice del polinomio caratteristico* $\varrho(\zeta)$ *ha modulo maggiore di uno, e se ogni radice con modulo uno non ha molteplicità maggiore di due.*

Date queste definizioni si enuncerà un importante teorema di cui non sarà data la dimostrazione:

Theorem 2.3.1 *La condizione necessaria e sufficiente affinchè un metodo multistep lineare sia convergente è che esso sia consistente e zero-stabile*

2.3.2 Metodi di Stormer

Una sottoclasse di metodi lineari multistep è quella dei metodi di Stormer, per cui

$$\varrho(\zeta) = \varrho^k - 2\varrho^{k-1} + \varrho^{k-2} \tag{2.27}$$

Il più noto è il metodo di Numerov che si ottiene per k = 2, è implicito ed ha ordine di constistenza $p = 4$. Poichè il metodo di

Numerov sarà utilizzato nel codice descritto nel prossimo capitolo, descriviamo nel paragrafo seguente come si ricava e come se ne ottiene un' espressione esplicita quando usato per integrare l'equazione di Schrödinger.
Per derivare la forma generale dei metodi di Stormer è necessario definire la $differenza\,centrale$.

Definition 2.3.3 *Data una funzione $f(x)$ definita su $\forall x \in [a,b]$ e tre punti $x_0, x_0+h, x_0-h \in [a,b]$ la differenza centrale di $f(x)$ attorno al punto x_0 è*

$$\triangle f(x_0) = f(x_0+h) - f(x_0-h) \tag{2.28}$$

La differenza centrale può essere usata per dare un approssimazione numerica delle derivate. [6]
A questo punto si possono derivare i metodi di Stormer di tipo esplicito e, in particolare, si darà una derivazione di questi al quinto ordine da cui si può estrapolare il metodo di Numerov. Anche se la trattazione è qui specifica si può arrivare ad una forma generica per un qualsiasi ordine. Sia x_n un punto della griglia e $h > 0$ il passo di discretizzazione e si sviluppi in serie di Taylor la funzione $y(x)$ in $x_n + h$ e $x_n - h$

- $y(x_n+h) = y(x_n)+hy'(x_n)+\frac{h^2}{2}y'(x_n)+\frac{h^3}{6}y''(x_n)+\frac{h^4}{24}y^{(3)}(x_n)+\frac{h^5}{120}y^{(5)}+\frac{h^6}{720}y^{(6)}+...$
- $y(x_n-h) = y(x_n)-hy'(x_n)+\frac{h^2}{2}y'(x_n)-\frac{h^3}{6}y''(x_n)+\frac{h^4}{24}y^{(3)}(x_n)-\frac{h^5}{120}y^{(5)}+\frac{h^6}{720}y^{(6)}+...$

a questo punto si sommano le due approssimazioni ottenendo

$$y(x_n+h)+y(x_n-h) = 2y(x_n)+h^2y''(x_n)+\frac{h^4}{12}y^{(3)}(x_n)+\frac{h^6}{360}y^{(6)}+... \tag{2.29}$$

Ora, ricorrendo all'espressione della derivata quarta e sesta tramite l'uso delle differenze centrali, ovvero:

$$y^{(4)}(x_n) = \frac{1}{h^2}(\triangle^2 f_{n-1} - \frac{1}{12}\triangle^4 f_{n-2}+...) \tag{2.30}$$

$$y^{(6)}(x_n) = \frac{1}{h^4}(\triangle^4 f_{n-2}+...) \tag{2.31}$$

e inserendole nella (2.29) si ottiene:

$$y_{n+1} - 2y_n + y_{n-1} = h^2(f_n + \frac{1}{12}\triangle^2 f_{n-1} - \frac{1}{240}\triangle^4 f_{n-2} + ...) \quad (2.32)$$

questa espressione però non è ancora utile poichè, una volta sviluppate le differenze centrali, contiene i termini f_{n+1}, f_{n+2}. Per avere quindi un espressione esplicita della formula, trascurando le differenze finite del quinto ordine, si ottengono le approssimazioni

$$\triangle^2 f_{n-2} \approx \triangle^2 f_{n-4} \quad (2.33)$$

e

$$\triangle^2 f_{n-1} = \triangle^2 f_{n-2} + \triangle^3 f_{n-3} + \triangle^4 f_{n-3} \approx \triangle^2 f_{n-2} + \triangle^3 f_{n-3} + \triangle^4 f_{n-4} \quad (2.34)$$

che inserite nella (2.32) danno

$$y_{n+1} - 2y_n + y_{n-1} = h^2 f_n + \frac{h^2}{12}(\triangle^2 f_{n-2} + \triangle^3 f_{n-3} + \frac{19}{20}\triangle^4 f_{n-4}) \quad (2.35)$$

che è la forma finale del metodo di Stormer esplicito al 5 ordine.

2.3.3 Metodo di Numerov

Il metodo di Numerov è un caso speciale dei metodi di Stormer. Infatti, se si considera la (2.35) e si trascura la differenza centrale di ordine 4 si ottiene

$$y_{n+1} - 2y_n + y_{n+1} = h^2(f_n + \frac{1}{12}\triangle^2 f_{n-1}) = \frac{h^2}{12}(f_{n+1} + 10f_n + f_{n-1}) \quad (2.36)$$

Questa espressione presenta al membro di destra il termine f_{n+1}, che renderebbe la formula implicita. Tuttavia, dalla forma dell'equazione di Schrödinger, si riconosce che quest'ultima è un equazione differenziale lineare. Si può dimostrare come per equazioni lineari dalla (2.36) si può estrarre una formula iterativa esplicita. Infatti, considerando una generica equazione lineare del tipo

$$\frac{d^2y}{dx^2} = f(x, y) = g(x)y(x) \quad (2.37)$$

considerando che $f(x, y) \coloneqq g(x)y(x)$ e inserendola nella (2.36) si ottiene

$$y_{n+1} - 2y_n + y_{n-1} = \frac{h^2}{12}(g_{n+1}y_{n+1} + 10g_n y_n + g_{n-1}y_{n-1}) \quad (2.38)$$

raccogliendo i termini in y_{n+1}, y_n, y_{n-1}

$$y_{n+1}(1 - \frac{h^2}{12}g_{n+1}) = 2(1 + \frac{5}{12}h^2 g_n)y_n + (\frac{h^2}{12}g_{n-1} - 1)y_{n-1} \quad (2.39)$$

Ora si definisce una variabile ausiliaria

$$z_n = (1 - \frac{h^2}{12}g_n) \quad (2.40)$$

che, una volta inserita nella (2.39), dopo manipolazioni algebriche, produce la forma finale del metodo di Numerov:

$$y_{n+1} = \frac{(12 - 10z_n)y_n - z_{n-1}y_{n-1}}{z_{n+1}} \quad (2.41)$$

Si può vedere ora che il metodo di Numerov definito dalla (2.36) è un metodo a due passi lineare. Ricorrendo alla definizione (2.22) i coefficienti che compaiono nel metodo hanno valori:

$$\begin{cases} \alpha_0 = 1 \\ \alpha_1 = -2 \\ \alpha_2 = 1 \\ \beta_0 = \frac{1}{12} \\ \beta_1 = \frac{5}{6} \\ \beta_2 = \frac{1}{12} \end{cases} \quad (2.42)$$

Si vede che il metodo di Numerov rispetta anche le condizioni necessarie e sufficienti alla consistenza. Infatti, inserendo il valore dei coefficienti nei polinomi caratteristici $\varrho(\zeta)$ $\sigma(\zeta)$ si ottiene

$$\begin{cases} \varrho(\zeta) = 1 - 2\zeta + \zeta^2 \\ \sigma(\zeta) = \frac{1}{12} + \frac{5}{6}\zeta + \frac{1}{12}\zeta^2 \end{cases} \quad (2.43)$$

Calcolati $\varrho(1), \varrho(1)', \varrho(1)'', \sigma(1)$ si verifica che valgono le due condizioni (2.26) e quindi il metodo è consistente. Si può quindi vedere come il metodo di Numerov rispetta anche le condizioni di zero-stabilità date nella definizione (2.3.2). Infatti, si ha:

$$\varrho(\zeta) = 1 - 2\zeta + \zeta^2 = (\zeta - 1)^2 \quad (2.44)$$

e quindi il polinomio caratteristico ha una radice di modulo uno con molteplicità non maggiore di due, in accordo con la definizione.
Dato che il metodo di Numerov è zero-stabile e consistente, utilizzando il teorema (2.3.1), si può affermare che esso è anche convergente e che è un metodo, come si è detto, di ordine 4.

2.4 Metodo di Shooting

Quando si vuole risolvere l'equazione di Schrödinger ci si trova a dover affrontare problemi a valori al bordo in cui vengono dati due punti e i valori che la soluzione deve avere in questi due. Per esempio, nel caso che si affronterà nel seguito, ci si troverà a dover affrontare un problema definito dall'equazione (1.8) e dalle condizioni al bordo

$$\begin{cases} \psi(-\infty) = 0 \\ \psi(+\infty) = 0 \end{cases} \tag{2.45}$$

Le condizioni al bordo definite dalla (2.45) sono necessarie poichè il modulo quadro delle soluzioni dell'equazione di Schrödinger ha significato probabilistico e quindi le soluzioni devono essere normalizzabili. In questa tesi si illustra il metodo di Shooting che verrà implementato nel codice. Partendo da un generico problema ai valori al bordo

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = A \\ y(b) = B \end{cases} \tag{2.46}$$

con $a < b$ e assumendo che questo abbia un unica soluzione, si da un valore iniziale s per $y'(a)$ e si utilizza questo valore per definire il problema ai valori iniziali:

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = A \\ y'(a) = s \end{cases} \tag{2.47}$$

Chiamata $y(x; s)$ la soluzione del problema (2.47), si utilizza la notazione $u(x; s) = y(x; s)$ e $\frac{\partial y(x;s)}{\partial x} = v(x; s)$ così da trasformare il sistema (2.47) nel sistema del primo ordine:

$$\begin{cases} \frac{\partial u(x;s)}{\partial x} = v(x; s) \\ \frac{\partial v(x;s)}{\partial x} = f(x, u(x; s), v(x; s)) \\ u(a; s) = A \\ v(a; s) = s \end{cases} \tag{2.48}$$

La soluzione $u(x; s)$ di (2.48) sarà una soluzione anche di (2.46) se si riesce a trovare una valore per s tale che

$$\phi(s) \coloneqq u(b; s) - B = 0 \tag{2.49}$$

Questa equazione si può risolvere tramite il metodo di bisezione (come si farà nel codice) per trovare un valore di s che restituisca, entro la tolleranza richiesta, la soluzione di (2.46). Come si può vedere la (2.49) richiede la risoluzione di un equazione differenziale ad ogni passo della bisezione e questa, nel caso considerato in questa tesi, è effettuata tramite il metodo di Numerov.

Capitolo 3

Un codice per la risoluzione dell'Equazione di Schrödinger

Il codice che si presenta in questo capitolo risolverà il problema dell'equazione di Schrödinger in potenziale armonico. Il problema da risolvere avrà la forma dell'equazione (1.8) a cui si aggiungono le condizioni al bordo (2.45).
Il codice sarà presentato in due versioni: la prima risolverà il problema cercando soluzioni che sono in accordo con le condizioni (2.45) e, una volta mostrati i problemi relativi a questo primo codice, si vedrà come superarli nella seconda versione.
I codici scritti in linguaggio C forniranno in output la stima del valore dell'energia corrispondente la soluzione, e scriveranno in un file i punti che descrivono il profilo della funzione soluzione $\psi(x)$. I dati nel file saranno poi interpretati dal programma Gnuplot [7] con cui si visualizzerà la soluzione.

3.1 Dettagli implementativi

La prima parte del codice è costituita dalle dichiarazioni delle funzioni utilizzate e dalla richiesta di alcuni parametri in input:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
```

```
6  void integrate(double xmax,double psi[],double E,double mesh,double
       x[],double vpot[],double h,double f[]);
7  void normalize(double psi[],int mesh,double h);
8  int main()
9   {
10   double e1,e2,*vpot,*x,h,e0,*psi,*f,de,b0,b1,b2,xmax;
11   int i,mesh;
12
13     FILE *out;
14     out = fopen("result","w");
15
16
17      printf("Inserire il numero di punti per la griglia: ");
18      scanf("%d",&mesh);
19      printf("Inserire l'estremo destro (positivo) dell'intervallo:"
       );
20      scanf("%lf",&xmax);
21      printf("Inserire il limite inferiore dell'energia: ");
22      scanf("%lf",&e1);
23      printf("Inserire l'intervallo de per la ricerca dell'energia:
       ");
24    scanf("%lf",&de);
25
26
```

La funzione *integrate* implementa il metodo di Numerov per l'integrazione dell'equazione e la funzione *normalize* opera la normalizzazione della funzione, così come ci si aspetta si presenti la soluzione dell'equazione di Schrödinger. Si vede quindi che sono presenti le dichiarazioni di alcune variabili il cui scopo sarà chiaro nel seguito. Nelle righe 13-14 si apre il file che conterrà i valori in output del codice e che verranno usati successivamente da Gnuplot per disegnare i grafici.
Dalla riga 17 alla 24 sono presenti le richieste all'utente dei parametri necessari a risolvere l'equazione e in particolare:

- Il primo dato in input richiesto è il numero di punti della griglia (che tipicamente è di qualche centinaio).

- Il secondo dato in input è l'estremo destro dell'intervallo in cui cercare la soluzione. Nel codice scritto si considera sempre un intervallo di tipo simmetrico rispetto l'origine e quindi fornito l'estremo destro si conosce l'intero intervallo.

- Il terzo dato in input è il limite inferiore dell'energia che servirà come valore iniziale per il metodo di Shooting.

- Il quarto dato in input è l'incremento $\triangle e$ utilizzato nella ricerca di un intervallo in cui è presente l'energia ricercata.

Successivamente si ha una parte di codice che si occupa dell'inizializzazione degli array che conterranno le variabili discretizzate, del passo di integrazione h e del potenziale.

```
1  vpot = (double *)malloc((mesh+1)*sizeof(double));
2  psi  = (double *)malloc((mesh+1)*sizeof(double));
3  x    = (double *)malloc((mesh+1)*sizeof(double));
4  f    = (double *)malloc((mesh+1)*sizeof(double));
5
6
7  h    = 2.*xmax/mesh;
8
9  for(i=0;i <= mesh;i++)
10 {
11      x[i]    = (double) (i-mesh/2)*h;
12      vpot[i] = 0.5*x[i]*x[i];
13 }
```

Come si vede dal codice è possibile cambiare il potenziale presente nell'equazione di Schrödinger semplicemente sostituendo nella riga 12 la forma del potenziale che si vuole utilizzare. Ora si ha la parte di codice dedicata alla ricerca di un intervallo che comprende l'energia corrispondente alla soluzione.

```
1  integrate(xmax,psi,e1,mesh,x,vpot,h,f);
2  b1 = psi[mesh];
3
4  while(1)
5    {
6          e2 = e1 + de;
7          integrate(xmax,psi,e2,mesh,x,vpot,h,f);
8          b2 = psi[mesh];
9          if(b1*b2 < 0)
10             break;
11         e1 = e2;
12         b1 = b2;
13
14     }
```

L'intervallo è trovato partendo dall'energia fornita dall'utente e si cercano due valori $e1$ e $e2$ per cui l'energia $e0$, a cui corrisponde la soluzione cercata, sia tale che $e0 \in [e1, e2]$. Inizialmente si effettua una prima integrazione con il metodo di Numerov partendo dall'energia $e1$ fornita dall'utente e si conserva il valore dell'ultimo punto della soluzione nella variabile $b1$.

Dalla riga 4 alla 14 è presente un ciclo in cui si risolve l'equazione di Schrödinger per valori $e2$ dell'energia incrementati di de ad ogni iterazione rispetto l'energia $e1$. Una volta ottenuta la soluzione corrispondente a $e2$ si conserva l'ultimo punto di questa nella variabile $b2$ e si verifica se tra $b1$ e $b2$ c'è un cambio di segno e se è cosi, il ciclo si ferma; se il cambio di segno non è verificato si continua aggiornando $e1$ al valore corrente dell'energia e si ricomincia il ciclo. La ragione per la ricerca di due valori dell'energia corrispondenti a due soluzioni di segno opposto si può capire quando si consideri le condizioni al bordo (2.45). Quest'ultime impongono l'annullarsi della soluzione all'infinito, e quindi, l'energia corrispondente alla soluzione dell'equazione di Schrödinger è necessariamente tra due valori dell'energia che restituiscono soluzioni aventi segno opposto all'estremità (qui considerata come il valore della funzione nell'ultimo punto della griglia).
Successivamente si ha una parte di codice relativa al metodo di Shooting.

```
1  while(1)
2      {
3          e0 = (double)(e1 + e2)/2;
4          integrate(xmax,psi,e0,mesh,x,vpot,h,f);
5          b0 = psi[mesh];
6
7          if(fabs(e1 - e2) < 1e-09)
8              break;
9          if(b0*b1 < 0)
10         {
11             e2=e0;
12             b2=b0;
13         }else
14         {
15             e1=e0;
16             b1=b0;
17         }
18
19
20     }
```

Qui si ha la ricerca della soluzione. Inizialmente si entra nel ciclo con i due valori dell'energia $e1, e2$ che delimitano l'intervallo in cui è presente l'energia voluta. Si prende il valore di mezzo tra i due estremi dell'intervallo e lo si assegna a $e0$. Dopodichè si effettua una integrazione trovando la soluzione corrispondente ad $e0$ e si conserva nella variabile $b0$ il valore di questa nell'ultimo punto.
Alla riga 7 si ha la verifica della tolleranza che farà uscire dal ciclo quando è rispettata ed è qui imposta a 10^{-9}. Se la tolleranza non è

rispettata si arriva alla riga 9 dove se si ha un cambio di segno tra la soluzione corrispondente a $e0$ e alla soluzione corrispondente $e1$ l'intervallo viene ristretto assegnando ad $e2$ il valore di $e0$ e quindi restringendo l'intervallo da destra. Se il cambio di segno si ha tra $b0$ e $b2$ l'intervallo si restringe da sinistra assegnando a $e1$ il valore di $e0$. La ragione per il controllo sul cambio di segno è la stessa spiegata nella sezione precedente.
Ora si guardano in dettaglio le due funzioni utilizzate nel codice

```
1  void integrate(double xmax,double psi[],double E,double mesh,double
       x[],double vpot[],double h,double f[])
2  {
3
4      psi[0] = 0;
5      psi[1] = 0.0001;
6      int i;
7
8      for(i = 0;i <= mesh;i++)
9      {
10        f[i] = 1 + 2.*(E - vpot[i])*h*h/12;
11     }
12
13
14
15
16     for(i=1;i < mesh;i++)
17     {
18         psi[i+1] = ((12-10*f[i])*psi[i] - f[i-1]*psi[i-1])/f[i+1];
19     }
20
21     normalize(psi,mesh,h);
22
23 }
```

Alla funzione *integrate* è affidata l'esecuzione del metodo di Numerov. Si fissano i valori nei primi due punti a sinistra dell'intervallo per permettere l'innesco del metodo. I due punti sono scelti in accordo con le condizioni al bordo considerando che nel caso ideale $psi[0] = \psi(-\infty)$ e per $psi[1]$ è scelto un valore abbastanza piccolo da essere ragionevolmente il valore della soluzione nel punto "successivo" a $-\infty$.
Dalla riga 8 a 11 si inizializza l'array $f[i]$ che è il corrispettivo della variabile ausiliaria definita nella (2.40).
Dalla riga 16 a 19 si esegue il metodo di Numerov così come definito dalla (2.41). Nella riga 21 c'è una chiamata alla funzione *normalize* che ha il compito di eseguire la normalizzazione della funzione, e che si analizza nel dettaglio:

```
void normalize(double psi[],int mesh,double h)
{
    int i;
    double norm = 0;
    for(i = 1;i <= mesh;i++)
    {
        norm += (psi[i]*psi[i] - psi[i-1]*psi[i-1])*h;
    }
    norm = sqrt(norm);

    for(i=0;i <= mesh;i++)
    {
        psi[i] /= norm;
    }


}
```

Come si può vedere dal codice la normalizzazione è fatta seguendo la definizione (1.4), ovvero dividendo la funzione per la radice quadrata dell' integrale del modulo quadro della funzione d'onda. L'integrale è calcolato tramite la formula dei trapezi. Il risultato dell'esecuzione del codice è:

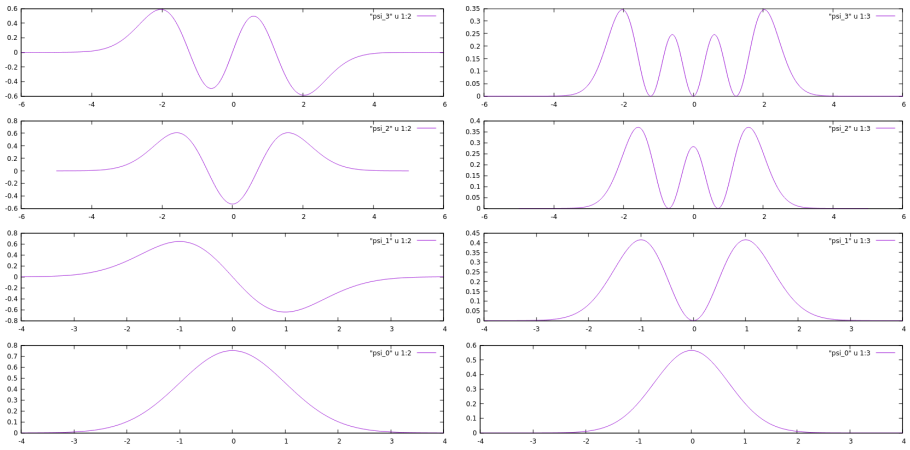

Figura 3.1: Prime 4 autofunzioni dell'oscillatore armonico(a sinistra) con i corrispondenti moduli quadro(a destra) ricavate dall'esecuzione del codice

I valori dell'energia corrispondenti alle quattro soluzioni sono:

$$\begin{cases} \epsilon_0 = 0.5000004907 \\ \epsilon_1 = 1.5000000033 \\ \epsilon_2 = 2.5000000834 \\ \epsilon_3 = 3.5000012193 \end{cases} \tag{3.1}$$

e come si può vedere sono valori precisi fino almeno alla quinta cifra decimale rispetto i valori teorici definiti dalla (1.16). Il codice descritto sembra funzionare finquando si sceglie un intervallo non troppo grande rispetto all'intervallo in cui la soluzione è apprezzabilmente diversa da zero. Infatti, se si scegliesse un intervallo troppo grande, si vedrebbe una divergenza esponenziale della funzione che la renderebbe non normalizzabile. Graficamente questo si può verificare eseguendo il codice per trovare la soluzione corrispondente ad un energia di 3.5 (quindi con 3 nodi) in un intervallo di grandezza 20. L'esecuzione risulta in:

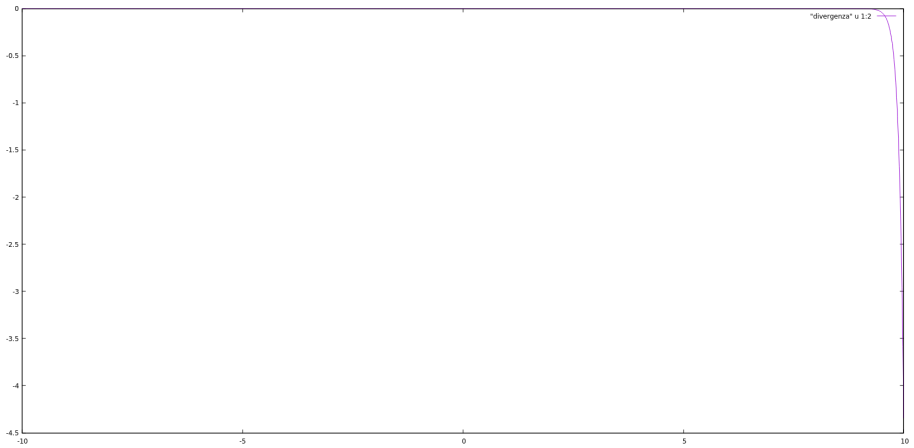

Figura 3.2: Esecuzione del codice per una soluzione corrispondente al valore di energia 3.5 in un intervallo di 20

La causa del problema è da ricercarsi nella doppia natura delle soluzioni dell'equazione di Schrödinger. Nel capitolo 1 si è mostrato che le soluzioni possibili per l'equazioni di Schrödinger erano in realtà 2, definite dalla (1.10) ma la soluzione con l'esponenziale divergente si era scartata poichè si sapeva a priori che sarebbe stata non normalizzabile. Dal punto di vista numerico, però, per grandi $|x|$, gli errori numerici presenti nelle operazioni a precisione finita producono uno spostamento della soluzione numerica su quella esponenziale positiva e quindi una soluzione divergente.
Per prevenire questo comportamento serve trovare un algoritmo che sia intrinsecamente stabile.

3.2 Codice risolvente i problemi di instabilità

Il codice che rende stabile la soluzione dell'equazione di Schrödinger è sostanzialmente diverso da quello non stabile. La soluzione, a differenza del codice precedente, viene trovata controllando il numero di nodi che questa ha in corrispondenza di una data energia. Infatti, nel teorema (1.2.2), si è visto che per un valore fissato dell'energia E_n appartenente allo spettro discreto, si ha una soluzione oscillante con n nodi[1]. Per trovare la soluzione si utilizzano due integrazioni: una in avanti che parte dall'estremo sinistro fino al punto di inversione classica e una all'indietro che parte dall'estermo destro e arriva al punto di inversione classica. Il punto di inversione classica x_{icl} è scelto poichè si può dimostrare che oltre x_{icl} definito dalla (1.28) non ci sono più nodi ma solo comportamenti crescenti o decrescenti. Una volta trovata la soluzione con il giusto numero di nodi tramite l'integrazione in avanti, si trova la soluzione a destra attraverso un procedimento all'indietro, che combaci con continuità con la soluzione oscillante. L'incontro tra i due pezzi della soluzione deve avvenire con continuità anche della derivata prima. Per avere una formula numerica per stimare la differenza delle derivate delle due parti di soluzioni a sinistra e a destra, si procede nel seguente modo: Indicando con ψ'^L la derivata della soluzione a sinistra e ψ'^R la derivata della soluzione a destra e ponendo $x_i = x_{icl}$, si espande in serie di Taylor intorno x_{i-1} per ψ'^L e intorno x_{i+1} per ψ'^R:

$$\psi^L_{i-1} = \psi^L_i - h\psi'^L_i + \frac{h^2}{2}\psi''^L_i + O(h^3) \tag{3.2}$$

$$\psi^R_{i+1} = \psi^R_i + h\psi'^R_i + \frac{h^2}{2}\psi''^R_i + O(h^3) \tag{3.3}$$

ora si somma la (3.2) alla (3.3) e considerando che in $x_i = x_{icl}$ si ha $\psi'^L_i = \psi'^R_i$ e che $\psi''^L_i = \psi''^R_i = \psi_i g_i$ si ottiene:

$$\psi^L_{i-1} + \psi^R_{i+1} = 2\psi_i + (\psi'^R_i - \psi'^L_i)h + g_i\psi_i h^2 \tag{3.4}$$

risolvendo ora per $\psi'^R_i - \psi'^L_i$ si ottiene

$$\psi'^R_i - \psi'^L_i = \frac{\psi^L_{i-1} + \psi^R_{i+1} - [2 + g_i h^2]\psi_i}{h} \tag{3.5}$$

[1]Nel teorema (1.2.2) si è utilizzato n=1 come stato fondamentale e quindi per l'n-esima autofunzione si avevano n-1 nodi. Qui lo stato fondamentale è considerato n=0 e quindi si hanno, per l'n-esima autofunzione, n nodi

Dopo manipolazione algebriche, si riesce a inserire la z_n di Numerov definita da (2.40) ottenendo:

$$\psi_i'^R - \psi_i'^L = \frac{\psi_{i-1}^L + \psi_{i+1}^R - [14 - 12z_n]\psi_i}{h} \tag{3.6}$$

Tramite l'equazione (3.6) posso quindi stimare il salto tra le derivate della soluzione a sinistra e destra che, per un soluzione ideale, deve essere zero. Ora si mostrerà in dettaglio il codice scritto.

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


void forward_integrate(double xmax,double psi[],double E,int mesh,
    double x[],double vpot[],double h,double f[],int icl);
void inward_integrate (double xmax,double psi[],double E,int mesh,
    double x[],double vpot[],double h,double f[],int icl);
int calculate_icl(double xmax,double E,int mesh,double vpot[]);

void set_init_condition(double x[],int mesh,double vpot[],double h)
    ;

void normalize(double psi[],int mesh,double h);


int main()
{
    double e1,e2,*vpot,*x,h,e0,*psi,*f,xmax,djump,psiicl;
    int i,mesh,icl,nodes,crossing;

    FILE *out;
    out = fopen("result","w");


    printf("Inserire il numero di punti per la griglia: ");
    scanf("%d",&mesh);
    printf("Inserire l'estremo destro (positivo) dell'intervallo:")
    ;
    scanf("%lf",&xmax);
    printf("Inserire il numero di nodi: ");
    scanf("%d",&nodes);



    vpot        = (double *)malloc((mesh+1)*sizeof(double));
    psi         = (double *)malloc((mesh+1)*sizeof(double));
    x           = (double *)malloc((mesh+1)*sizeof(double));
    f           = (double *)malloc((mesh+1)*sizeof(double));

    h           = 2.*xmax/mesh;
    set_init_condition(x,mesh,vpot,h);


    e1 = vpot[mesh];
```

```
43      e2 = vpot[mesh];
44
45      for (i = 0; i <= mesh; ++i)
46      {
47          if (vpot[i] < e1)
48              e1 = vpot[i];
49          if (vpot[i] > e2)
50              e2 = vpot[i];
51      }
52
53      e0 = 0.5*(e1 + e2);
```

In questa prima parte di codice sono presenti le dichiarazioni delle funzioni e in particolare *forward_integrate* e *inward_integrate* sono responsabili delle integrazioni in avanti e indietro.
La funzione *calculate_icl* restituisce l'indice all'interno dell'array corrispondente al punto di inversione classica.
La funzione *set_init_condition* si occupa di inzializzare il potenziale e l'array della posizione.
La funzione *normalize* si occupa di normalizzare la soluzione.
Alle righe 17-18 corrispondono dichiarazioni di variabili il cui significato sarà chiaro in seguito.
Alle righe 20-21 si apre il file in cui verrà scritto il risultato dell'output del codice.
Dalla riga 24 a 29 sono presenti le richieste da parte del programma di parametri in input. In particolare viene richiesto il numero di punti in cui dividere la griglia, l'estremo destro dell'intervallo che servirà a caratterizzarlo completamente, poichè anche qui si utilizza un intervallo simmetrico rispetto l'origine e infine viene richiesto il numero di nodi che servirà per identificare la soluzione, cosi come garantito dal teorema (1.2.2).
Dalla riga 33 alla 38 sono presenti le inizializzazioni degli array e del passo di integrazione h.
Alla riga 39 c'è la chiamata alla funzione *set_init_condition*() per inzializzare la posizione e il potenziale.
Dalla riga 42 a 51 si inizializzano due valori di partenza dell'energia $e1$ e $e2$ come il minimo e il massimo del potenziale, in accordo con quanto affermato nel teorema (1.2.3), dopodichè alla riga 53 si assegna il valore del punto di mezzo a $e0$.

A questo punto è presente la sezione di codice che servirà effettivamente a trovare la soluzione.

```
1       while(fabs(e1 - e2) > 1.e-10)
```

```
    {
       for(i = 0;i <= mesh;i++)
       {
           f[i] = 1 + 2.*(e0 - vpot[i])*h*h/12;
        }

        crossing = 0;

        icl = calculate_icl(xmax,e0,mesh,vpot);



        forward_integrate(xmax,psi,e0,mesh,x,vpot,h,f,icl);

        for(i=1;i <= icl-1;i++)
        {
            if(psi[i] != copysign(psi[i],psi[i-1]) )
            {
                crossing +=1;
            }
        }

        psiicl = psi[icl];

        if(crossing != nodes)
        {

                if(crossing > nodes)
                    e2 = e0;
                else
                    e1 = e0;

                e0 = 0.5*(e1 + e2);

        }
        else if(crossing == nodes)
        {
            inward_integrate(xmax,psi,e0,mesh,x,vpot,h,f,icl);

            psiicl /=psi[icl];

            for(i=icl;i <= mesh;i++)
            {
                psi[i] *= psiicl;
            }
            normalize(psi,mesh,h);
            djump = (psi[icl+1] + psi[icl-1] - (14. - 12.*f[icl])*
psi[icl])/h;
            if(djump*psi[icl] > 0.)
                e2 = e0;
            else
                e1 = e0;

            e0 = 0.5*(e1 + e2);


        }
```

```
58      }
```

Si vede che tutte le istruzioni sono all'interno di un ciclo che si estende dalla riga 1 alla riga 58. Questo sarà il ciclo principale che si fermerà quando sarà raggiunta la tolleranza richiesta per la soluzione (in questo caso imposta a 10^{-10}). Dalla riga 3 alla 6 si ha l'inizializzazione della $f[]$, ovvero della variabile ausiliaria (2.40).
Alla riga 8 si ha l'inzializzazione a zero della variabile *crossing* la quale contiene il numero di nodi della funzione che si troverà successivamente. Alla riga 10 è presente una chiamata a *calculate_icl* che restituisce l'indice *icl* della posizione del punto di inversione classica all'interno dell'array contenente i punti della griglia. Alla riga 14 c'è una chiamata a $forward\_integrate$ che restituisce la soluzione corrispondente al valore dell'energia $e0$ e sarà la parte di soluzione oscillante, e quindi la parte in cui sono presenti i nodi.
Dalla riga 16 alla 22 si conta il numero di nodi, che qui vengono identificati come il numero di cambiamenti di segno della funzione. Alla riga 24 si salva il valore della soluzione nel punto di inversione classica, che sarà utile nel seguito.
Tutte le istruzioni dalla riga 26 a 57 sono all'interno di un istruzione condizionale che modificherà il flusso del codice a seconda se si è trovata la soluzione sinistra con il giusto numero di nodi o no. Nel caso in cui la soluzione non ha il numero di nodi richiesto viene eseguito il codice dalla riga 29 alla riga 34. Il codice in queste righe esegue una bisezione nell'intervallo definito da $[e1, e2]$. Il numero di nodi della soluzione corrente viene confrontato con quello richiesto e, dato che il numero di nodi è crescente con l'energia, se questi sono troppi l'energia viene abbassata assegnando il valore $e0$, del punto di mezzo dell'intervallo, a $e2$. Se il numero di nodi della soluzione corrente è più basso di quello della soluzione cercata l'energia viene alzata assegnando ad $e1$ il valore di $e0$.
Una volta che si è trovata la soluzione con il giusto numero di nodi verrà eseguito il codice dalle righe 39 a 54. Questo pezzo di codice agisce facendo dei piccoli aggiustamenti dell'energia $e0$ in modo da trovare una soluzione a destra che sia decrescente e compatibile con la soluzione oscillante di sinistra, la quale non sarà modificata dai cambiamenti di $e0$ essendo questi abbastanza piccoli da non influenzarla.
Alla riga 39 si ha l'integrazione all'indietro che restituisce una soluzione corrispondente ad $e0$ sui punti della griglia dal punto di

inversione classica a $xmax$. Dalla riga 41 a 46 si scala il valore della soluzione di destra in modo che questa si incontri con la soluzione di sinistra. Il riscalamento è permesso poichè la moltiplicazione per una costante non modifica il fatto che sia la soluzione dell'equazione. Alla riga 47 si normalizza la funzione ψ che è l'unione delle due parti. A questo punto bisogna verificare la grandezza del salto che compie la derivata di ψ nel punto di incontro dei due pezzi. La verifica si fa calcolando il valore del salto tramite la (3.6) e assegnandolo alla variabile *djump*. Dopodichè è presente un controllo sul segno del salto moltiplicato per il valore della soluzione in x_{icl} cosi da tenere in conto anche il segno che questa assume. Se il salto è positivo vuol dire che $\psi^{R}_{icl}{}' > \psi^{L}_{icl}{}'$ e quindi bisogna abbassare l'estremo destro dell'intervallo assegnando il valore di $e0$ a $e2$, viceversa nel caso in cui il salto è negativo. Il ciclo continua finchè l'intervallo non rispetterà le condizioni di tolleranza nella riga 1.
Per l'implementazione in dettaglio delle funzioni usate si descrivono solo *calculate_icl* e *inward_integrate* dato che $forward\_integrate$ è ugule alla funzione *integrate* descritta a pagina 29, la funzione *normalize* è uguale a quella descritta a pagina 30, *set_init_condition* raggruppa solo le istruzioni di inizializzazione nelle righe 9-13 del codice a pagina 27.
Si vede ora in dettaglio il codice di *calculate_icl*

```
1  int calculate_icl(double xmax,double E,int mesh,double vpot[])
2  {
3      int i,icl;
4      double *t = (double *)malloc((mesh + 1)*sizeof(double));
5
6      for(i=0;i <= mesh;i++)
7      {
8          t[i] = E - vpot[i];
9          if(t[i] != copysign(t[i],t[i-1]))
10         {
11             icl = i;
12
13         }
14     }
15     return icl;
16 }
```

Questa funzione è composta da un ciclo su tutti i punti della griglia e *icl* sarà l'indice corrispondente l'ultimo cambiamento di segno tra l'energia e il potenziale, in accordo con la definizione di punto di inversione classica data in (1.28). Si noti che anche se questa funzione restituisce un valore preciso di *icl*, comunque, dato che si sta trattando un intervallo discreto, il vero punto di inversione classica

si troverà tra $x[icl]$ e $x[icl-1]$.
Il codice di *inward_integrate* è

```
1  void inward_integrate(double xmax,double psi[],double E,int mesh,
       double x[],double vpot[],double h,double f[],int icl)
2  {
3
4      psi[mesh] = 0.001;
5      psi[mesh-1] = (12.-10.*f[mesh])*psi[mesh]/f[mesh-1];
6      int i;
7
8      for(i=mesh-1;i >= icl+1;--i)
9      {
10         psi[i-1] = ((12.-10.*f[i])*psi[i] - f[i+1]*psi[i+1])/f[i
       -1];
11     }
12
13
14 }
```

L'unica modifica rispetto a *integrate* di pagina 29, è il valore che si da al punto precedente l'estremità destra della funzione, che in questo caso è ricavato tramite il primo passo del metodo di Numerov. Il ciclo alla riga 8 parte dal penultimo punto della griglia e itera all'indietro fino ad arrivare a *icl*, come ci si aspetta per un integrazione da destra verso sinistra dell'intervallo.

Si possono trovare a questo punto le soluzioni corrispondenti a $0, 1, 2, 3$ nodi, in modo analogo a quanto fatto a pagina 30, ottenendo:

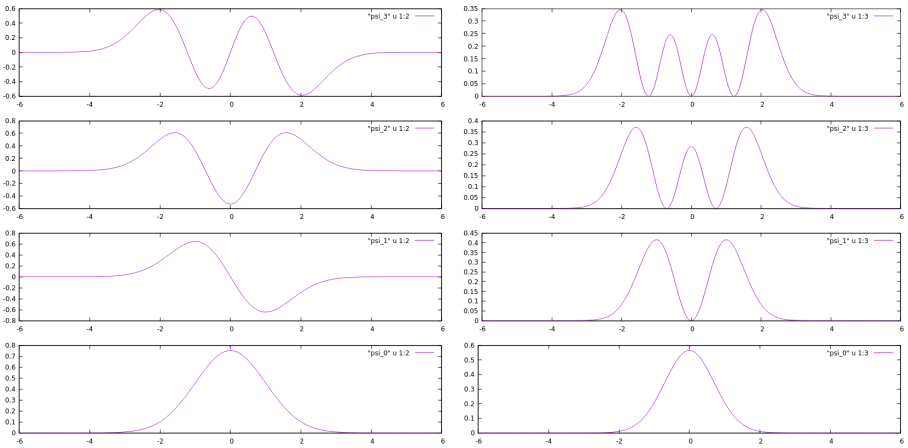

Figura 3.3: Le prime quattro autofunzioni dell'oscillatore armonico (a sinistra) con i corrispondenti moduli quadro(a destra) trovate tramite l'esecuzione del codice stabile

con valori dell'energia corrispondenti:

$$\begin{cases} \epsilon_0 = 0.50000003027890 \\ \epsilon_1 = 1.50000007223935 \\ \epsilon_2 = 2.50000009861469 \\ \epsilon_3 = 3.50000011804877 \end{cases} \tag{3.7}$$

Come si vede l'output è simile a quello del codice non stabile. Per mostrare il vantaggio nell' utilizzare questo codice in luogo del precedente si deve confrontare l' output per l'autofunzione corrispondente all'energia $\epsilon_3 = 3.5$ in un intervallo di lunghezza 20 con quello di pagina 31. L'output è:

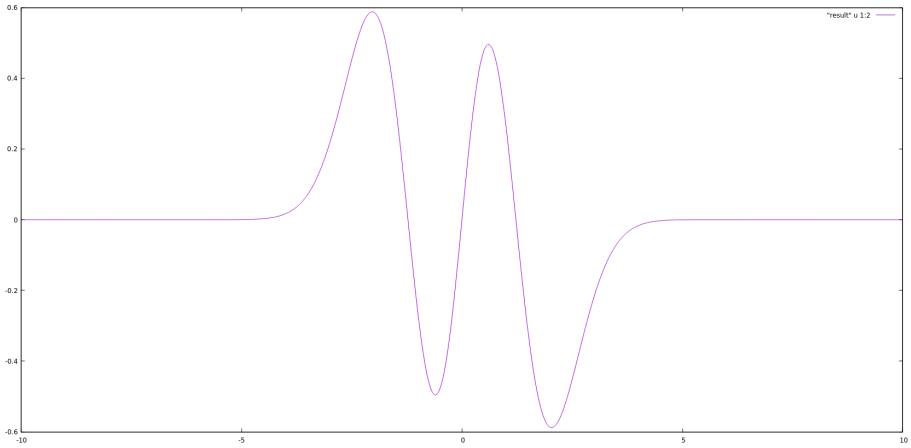

Figura 3.4: Soluzione corrispondente al valore di energia 3.5 in un intervallo di 20

che non presenta il comportamento divergente visto nel grafico in figura 3.2. Da quest'ultimo grafico si comprende il vantaggio nell'uso del codice più stabile, seppur di più difficile implementazione.

3.3 Sperimentazione numerica

Il codice scritto può essere utilizzato anche con altri potenziali diversi da quello armonico. Vediamo come esempio di potenziale quello trovato più comunemente in letteratura, ovvero il potenziale a buca

definito da:

$$\begin{cases} 0 \ \ per \ \ |x| \leq a \\ V_0 \ \ per \ \ |x| \geq a \end{cases} \tag{3.8}$$

Ripetendo l'analisi dimensionale fatta nel capitolo 1, si può pensare di porre la lunghezza della buca $a = 1$ cosicchè tutte le lunghezze siano misurate rispetto ad a. Si può anche porre $\frac{m}{h^2} = 1$ che ha le dimensioni di $(energia)^{-1}(distanza)^{-2}$ così che le energie siano misurate in unità di $\frac{h^2}{ma^2}$. Per una buca finita di potenziale non si hanno delle espressioni analitiche per i valori dell'energia, ma ci aspettiamo che per un V_0 abbastanza alto, questi siano simili ai valori delle energie in un buca di potenziale infinita. Dalla teoria si ha che le energie per una buca di potenziale infinita sono

$$E_n = \frac{h^2 n^2}{8ma^2} \tag{3.9}$$

che nel sistema di unità di misura imposto diventano

$$E_n = \frac{n^2 \pi^2}{2} \tag{3.10}$$

Un buon valore del potenziale V_0 si può scegliere considerando che valore troppo basso tenderebbe a non intrappolare la particella e valori troppo alti sarebbero troppo simili ad una buca di potenziale infinita. Dall'analisi dell' energie definite dalla (3.10) si vede che un giusto valore è $V_0 = 50$. Quello che mi aspetto dall'output del codice è che, in accordo con quando detto nel teorema (1.2.3), esistano solo energie che rispettino $0 < E_n < 50$. Eseguendo il codice si ottiene:

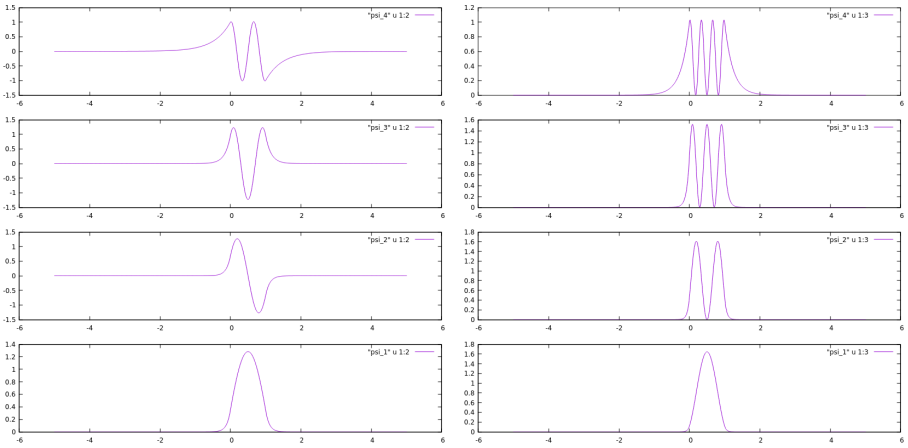

Figura 3.5: Soluzioni corrispondenti alle 4 autofunzioni della buca di potenziale finita

Le energia corrispondenti sono:

$$\begin{cases} E_1 = 3.3613 \\ E_2 = 13.2743 \\ E_3 = 29.0395 \\ E_4 = 47.6994 \end{cases} \tag{3.11}$$

Se si prova a risolvere il problema per un numero maggiore di nodi, il codice restituisce autofunzioni con evidenti discontinuità, segno che non ci sono valori altri valori delle energie nello spettro discreto e che la particella è uscita fuori dalla buca. Si vede quindi che l'output è coerente con quanto affermato nel teorema (1.2.3), valendo la condizione $E_4 < 50$. Con questi risultati si può considerare, a meno di stime più precise, il problema della buca di potenziale risolto correttamente dal codice.

3.4 Comportamento dell'errore

A pagina 39 si è visto come i valori trovati dell'energie differiscono rispetto ad i valori teorici per un errore che è maggiore della tolleranza imposta di 10^{-10}. Il problema è di come si comporta l'errore del metodo del metodo di Numerov quando esso è inserito nel contesto del metodo di Shooting. Infatti, si vede che oltre l'errore proprio del metodo di risoluzione dell'equazione differenziale, c'è anche una

dipendenza dell'errore da quello della bisezione utilizzata nel processo di "trial and error". Possiamo vedere come l'interferenza di questi due porta ad una dipendenza dell'errore finale dal numero di punti della griglia e dall'ampiezza dell'intervallo.
Sebbene in questa tesi non si fa una trattazione analitica dell'errore, si vuole comunque mostrare come varia la stima dell'energia al variare dei parametri imposti dall'utente.
Per studiare l'errore si può intutivamente pensare come questo dipenda dalla lunghezza del passo h (quindi dal numero di punti) e dalla larghezza dell'intervallo che, se più piccolo dell'intervallo in cui la funzione è apprezzabilmente diversa da zero, potrebbe causare perdita di informazione. Ci si mette nel caso della risoluzione del problema dell'equazione di Schrödinger in potenziale armonico, così da avere per le energie un valore teorico con cui confrontare la bontà dell'approssimazione numerica. Qui si utilizza la soluzione con 3 nodi corrispondente a un energia $E_{3teorica} = 3.5$.
Come prima analisi si vede come varia la precisione dell'approssimazione nel caso di intervallo fissato e numero di punti della griglia variabile.
Si scelgono come parametri fissi $xmax = 6$,$nodes = 3$ e si disegna un grafico $E_{3teorica} - E_{3numerica}$ vs $mesh$. Essendosi presi valori per $mesh$ che abbracciano vari ordini di grandezza si utilizza un grafico in scala logaritmica. Il grafico dell'errore è:

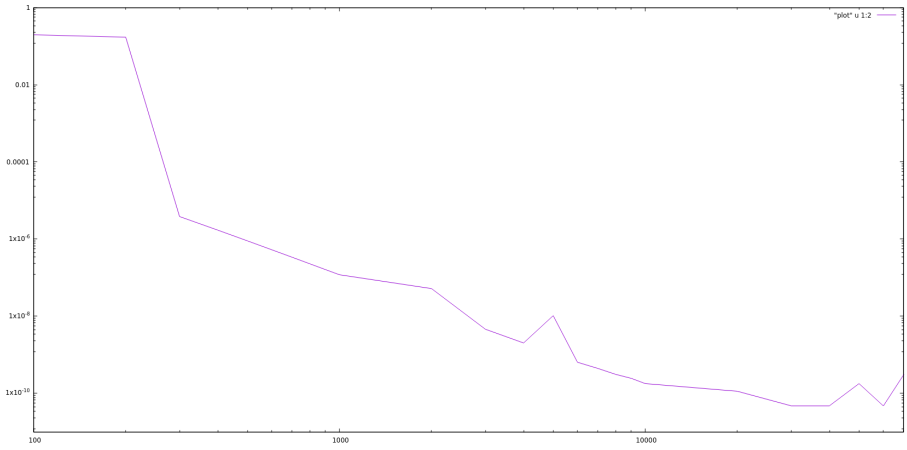

Figura 3.6: Grafico dell'errore a intervallo fisso numero di punti variabile

Come si vede dal grafico l'errore raggiunge un minimo verso i

40 mila punti della griglia, prima di cominciare a risalire per via di errori numerici via via predominanti per alti numeri di punti della griglia. Se adesso si fa un grafico dell'errore in funzione della lunghezza dell'intervallo a numero di punti fissato si ha:

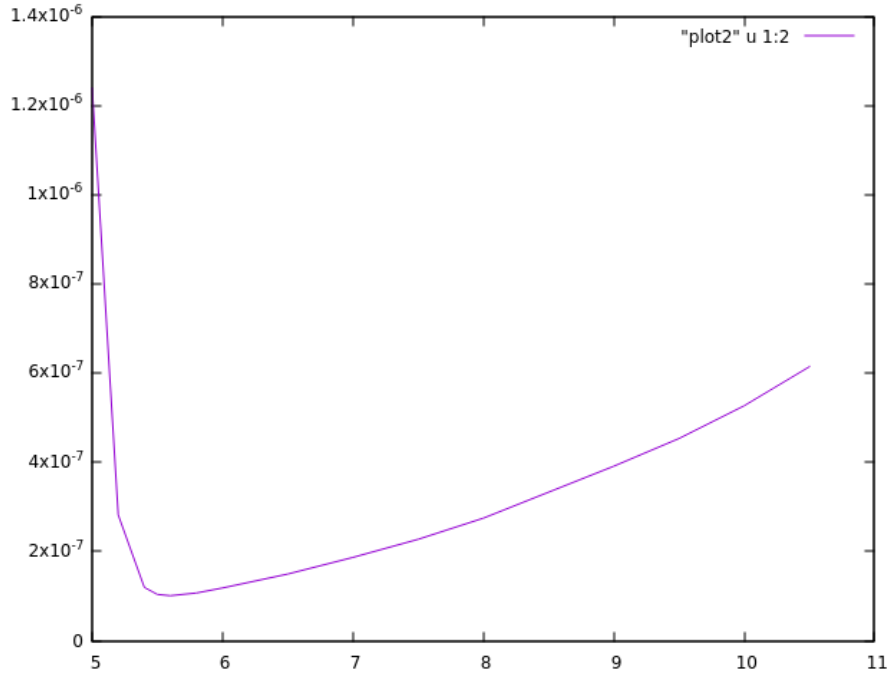

Figura 3.7: Grafico dell'errore a numero di punti fissi e intervallo variabile

Per piccoli intervalli si può vedere come il grafico presenti un picco nell'errore proprio per la perdita di informazioni sulla funzione risolvente. Dopo il minimo a $xmax = 5.5$ c'è una risalita dell'errore poichè per grandi intervalli la larghezza del passo h, che avendo fissato il numero di punti aumenta all'aumentare del intervallo, comincia ad influenzare l'andamento della soluzione.
Da questa analisi si può capire come all'aumentare del numero di nodi richiesto, per non perdere informazione, è necessario aumentare l'intervallo e aumentando quest'ultimo serve aumentare il numero di punti per mantenere l'accuratezza dalla soluzione. Queste considerazioni fanno capire come, per avere una buona approssimazioni dei valori teorici, è necessario mantenere h piccolo e quindi aumentare il numero di punti in cui si suddivide l'intervallo con conseguente

aumento del costo computazionale.

Bibliografia

[1] https://universe-review.ca/I15-71-quantum.gif

[2] http://www.math.ubc.ca/ israel/m215/euler2/euler4.gif

[3] M. Moriconi, Nodes of wavefunctions, American Journal of Physics 75, 284 (2007); https://doi.org/10.1119/1.2404960

[4] Lambert, J. D. Computational methods in ordinary differential equations. Introductory Mathematics for Scientists and Engineers. John Wiley & Sons, London-New York-Sydney, 1973.

[5] E.Hairer, S.P. Norset, G. Wanner, Solving Ordinary Differential Equations I. Springer, 2008.

[6] F. B. Hildebrand. Finite-Difference Equations and Simulations, Prentice Hall, 1968.

[7] http://www.gnuplot.info/