Characterization of Resonances for 1-D Potentials through Uncertainty Measures

Aritra Bhattacharya

A dissertation submitted for the partial fulfillment of BS-MS dual degree in Science

Under the guidance of

Dr P Balanarayan

INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH

MOHALI

May 09 2021

Aritra Bhattacharya

Characterization of Resonances for 1-D Potentials through Uncertainty Measures

May 09 2021

Supervisor: **Dr P Balanarayan**

Quantum Mechanics (QM) Group

Department of Chemical Sciences

Indian Institute of Science Education and Research Mohali

Knowledge City, Sector 81

Manauli P.O. 140306 SAS Nagar

Certificate of Examination

This is to certify that the dissertation titled **"Characterization of Resonances for 1-D Potentials through Uncertainty Measures"** submitted by **Aritra Bhattacharya** (Reg. No. MS14145) for the partial fulfillment of BS-MS dual degree programme of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.



Dr Amit Kulshrestha Dr Varadharaj R Srinivasan Dr Chandrakant S Aribam

Dr P Balanarayan

(Supervisor)

Dated: May 09 2021

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Dr P Balanarayan at the Indian Institute of Science Education and Research Mohali.

This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgement of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the references.

Aritra Bhattacharya

(Candidate)

Dated: May 09 2021

In my capacity as the supervisor of the candidate's project work, I certify that the above statements by the candidate are true to the best of my knowledge.

Dr P Balanarayan

(Supervisor)

Acknowledgements

I am indebted to my institute Indian Institute of Science Education and Research Mohali for providing me the resources for this research. I would like to express my sincere gratitude for my supervisor Dr P Balanarayan and the members of my thesis committee Dr Amit Kulshrestha, Dr Varadharaj R Srinivasan and Dr Chandrakant S Aribam for reviewing drafts of my thesis. I thank the members of the *QM* group and in particular, Mr Naveen Kumar and Mr Alkit Gugalia for their support.

I owe my deepest gratitude to my parents and my friends who supported me throughout this duration. They are too numerous to be all named individually, but I must mention the ones whose contributions have been especially impactful. My heartfelt regards for Mr Aaditya Mishra, Ms Kuwari Mahanta, Mr Shubham A Singh, Mr Vaibhav K Singh, Mr Aman S Katariya, Mr Ankush Sharma, Mr Akhil P Singh, Mr Ashirbad Mishra, Mr Deepraj Verma, Mr Yogendra Kumar, Mr Sumit Thakur, Mr Shivam A Meena, Mr Swadheen Dubey and Mr Ramandeep Singh.

Lastly, I would like to mention Mr Aaditya Mishra again for generously allowing me use of his workstation and for the innumerable fruitful discussions. This dissertation would not exist in its present form without him.

List of Figures

List of Tables

Listings

Abstract

Resonances and bound states are quantum phenomena without classical analogues. They share a commonality in that they can both be obtained from the poles of the S-matrix. The probabilistic nature of the quantum world is best understood in terms of uncertainty relations like Heisenberg's celebrated principle. However, standard deviation fails to adequately capture localization for multi-modal distributions, while Shannon information entropy is lucrative due to its sole dependence on the distribution.

By simulating model 1-D quantum systems, an effort can be made to classify bound states and characterize resonances in terms of both these uncertainties in the position and momentum conjugate spaces. These measures are evaluated for the stationary wavefunctions of the finite square well and the finite rectangular barrier to study the bound states of the former and the resonances of the latter. They are also computed for a Gaussian wavelet propagating through the finite rectangular barrier as well as the symmetric double barrier with the goal of characterizing the resonant energies.

Contents

1

Introduction

Since the conception of quantum laws governing the microscopic world, the study of quantum scattering has revolutionized our understanding of the physics of molecules, atoms and sub-atomic particles[1], and even the study of chemical reactions[2]. Recently, there has been a resurgence of interest in the study of 1D tunneling across hetero-structures, the likes of barriers, wells and their combinations. These structures find applications in a plethora of opto-electronic devices based on resonant tunneling[3].

Analytical S matrix theory of potential scattering provides a unified framework for ascertaining bound and resonance states generated by the potential[4]. Both these types of states are represented by the poles of S matrix in complex energy or momentum planes. Bound states generated by the scattering potential are normalizable negative energy states with zero width. On the other hand, resonance states are complex eigenstates, the real part of which corresponds to the resonance energy and the imaginary part to its width. The energies of these resonances correspond to the peaks of transmission obtained by studying it as a function of energy.

Quantum confinement is a measure of particle localization[5]. Tunneling, related to the probability of finding a particle in classically forbidden regions, also indicates delocalization of a particle through a barrier[6]. Consequently, tunneling can be said to increase the uncertainty of finding a confined particle[7].

Standard deviation is a popular measure of localization and uncertainty in quantum theory. The Heisenberg uncertainty relation[8], $\sigma_x \sigma_p \geq \frac{\hbar}{2}$ has long been the topic of interpretation. Uncertainty in position space is a known good measure of spatial delocalization[9]. However, in multimodal probability distributions, the uncertainty increases even when the probability distribution is intuitively more localized[10]. The basic problem lies in the fact that standard deviation is defined as the second moment about the average and is highly sensitive to outliers from the mean[11].

An alternative measure of localization is the Shannon information entropy[12]. Unlike standard deviation, Shannon entropy is a global measure of uncertainty in the sense that it does not care about individual values that the variable may assume, only the probability distribution itself. The sum of Shannon position and momentum space entropies yields a

generalized uncertainty relation, known as the Białynicki-Birula Mycielski inequality[13]. It turns out to be a stronger bound than the Heisenberg uncertainty inequality since the latter can be derived as a special case.

1.1 Outline of the Thesis

The contents of the thesis is organized as follows:

- Chapter 1 briefly talks about the phenomenon of quantum tunneling, the relationship between bound and resonant state energies, and discusses the two most popular measures of uncertainty: standard deviation and Shannon entropy.
- Chapter 2 compiles a list of bounds and inequalities pertaining to Shannon entropy.
- Chapter 3 details the methodology of the work and the particulars of the computations required to simulate the properties of the model systems in study.
- Chapter 4 studies the results of above computations and attempts to draw inferences from the observed plots.
- An appendix is provided at the end which lists the full code in $Python\ 2.7$ used for the thesis.

1.2 Quantum Tunneling

Tunneling refers to particle transport through a classically forbidden region, i.e., a region whose potential is higher than the total energy of the particle. In classical mechanics, such a particle is completely reflected up to the turning points where the total energy equals the potential energy, and completely transmitted thereafter.

The Schrödinger equation is the governing equation of motion in quantum mechanics. The role of potential here can be compared to that of permittivity in the study of electromagnetic waves propagating across different media[14]. The solutions of Maxwell's wave equation must satisfy certain boundary conditions at the interface between two dielectrics of different permittivity, causing a certain portion of the incident wave to be transmitted and the rest reflected.

Likewise, the wavefunction and its derivative has to be continuous across the boundary of two regions of varying potential., leading to reflection and transmission at the boundary.

Owing to this, it is better to think of quantum tunneling with regards to the wave nature of the particle undergoing transmission. The modulus square of the wavefunction represents the probability density function for finding the particle, so an incident particle has a finite probability of tunneling through the barrier and being found on the other side.

1.2.1 Bound States

For a potential $U(x)$ vanishing sufficiently rapidly as $|x| \longrightarrow \infty$, we seek the solution $\phi(x)$ of the time-independent Schrödinger equation satisfying the asymptotic behaviour:

$$\phi(x) \underset{x \longrightarrow -\infty}{\Longrightarrow} Ae^{ikx} + Be^{-ikx}$$

$$\phi(x) \underset{x \longrightarrow \infty}{\Longrightarrow} Fe^{ikx}$$

The coefficients A B and F are satisfied by the relations[14]:

- $W[\phi, e^{-ikx}]_{x \longrightarrow -\infty} = -2ikA$
- $W[\phi, e^{ikx}]_{x \longrightarrow -\infty} = 2ikB$
- $W[\phi, e^{-ikx}]_{x \longrightarrow \infty} = -2ikF$

Here W denotes the Wronskian such that $W[f(x), g(x)] = fg' - gf'$. These relations show that A, B and F can be treated as functions of k and hence energy E. Now consider a possible solution $\phi(x)$ having $E < 0$ such that k is purely imaginary, i.e., $k = i\beta, \beta \in \mathbb{R}$. When $\beta > 0$, asymptotically we have:

$$\phi(x) \longrightarrow Fe^{-\beta x}, x \longrightarrow \infty$$

$$\phi(x) \longrightarrow Ae^{-\beta x} + Be^{\beta x}, x \longrightarrow -\infty$$

Now for $\beta = \beta_n, A = A(k) = A(i\beta_n) = 0$ then $\phi(x) \longrightarrow 0, as|x| \longrightarrow \infty$ and hence the corresponding $\phi_n(x)$ can be treated as bound state wavefunction, which can then be normalized such that:

$$\int_{-\infty}^{\infty} |\phi_n(x)|^2 dx = 1$$

Hence, searching for zeros of $A(k)$ along the positive imaginary axis leads to the bound state eigenvalues and the corresponding bound state wavefunctions.

1.2.2 Resonant States

One can view the problem of transmission across a finite barrier as the positive energy counterpart of the bound state problem. Asymptotically, the solutions are combinations of plane waves moving in $\pm x$ directions and are hence not normalizable in the usual sense.

In the continuum, the plane waves are specified with the 'delta function' normalization:

$$\phi_0(\pm k, x) = \frac{1}{\sqrt{2\pi}} e^{\pm ikx}$$

such that

$$\int_{-\infty}^{\infty} (\phi_0(\pm k, x))^* \phi_0(\pm k', x) dx = \frac{1}{\sqrt{2\pi}} \delta(k - k') \tag{1.1}$$

However, overall normalization is not important, just the ratios of the amplitudes. Ae^{ikx}, Be^{-ikx} and Fe^{ikx} represent the incident, reflected and transmitted waves respectively. The incident, reflected and transmitted flux can be evaluated by using the general expression for probability current density J given by:

$$J = \frac{\hbar}{2im} (\phi^* \nabla \phi - \phi \nabla \phi^*)$$

Let the wave numbers associated with the transmitted and incident waves be denoted as k_{tr} and k_{in} respectively. Thus, the incident flux is $\frac{|A|^2 \hbar k_{in}}{m}$ and the transmitted flux is $\frac{|F|^2 \hbar k_{tr}}{m}$. Thus, the ratio of the transmitted flux to the incident flux T and the ratio of the reflected flux to the incident flux R are given by:

$$T = \frac{k_{tr}}{k_{in}} \frac{|F|^2}{|A|}$$

$$R = \frac{|B|^2}{|A|}$$

The conservation of flux implies that $T + R = 1$.

Consider the case where $k \in \mathbb{C}$ such that $A(k_R) = 0$ where $k_R = k_r - ik_i$, $k_i > 0$, $|k_r| > k_i$. The corresponding complex energy is $E_R = E_r - \frac{i\tau_r}{2}$ where $E_r = \frac{\hbar^2}{2m}({k_r}^2 - {k_i}^2)$ and $\tau_r = \frac{\hbar^2}{2m} 4 k_r k_i$. As $x \longrightarrow \infty$, $|\phi(x)| \longrightarrow Fe^{ikx}$ and as $x \longrightarrow -\infty$, $|\phi(x)| \longrightarrow Be^{-ikx}$. The corresponding time dependent part of the wavefunction behaves like $e^{\frac{-\tau_r t}{2\hbar}}$. Thus, $E = E_r$ signifies a positive energy state. Thus, the resonances generated by the potential U(x) can be ascertained by seeking complex zeros of $A(k)$ in the open lower-half k-plane.

Thus, we see that both resonant and bound states can be understood in terms of the zeroes of $A(k)$. But, $\frac{F(k)}{A(k)}$ and $\frac{B(k)}{A(k)}$ describe transmission and reflection functions for a unit amplitude of incident wave. Hence, one finds that both bound and resonant state energies can be obtained from the poles of the S-matrix in the complex k-plane.

1.3 Measures of Uncertainty

Statistical complexity is a tool based on information theory. Uncertainty is a closely related notion. At a fundamental level, it represents the lack of information or negative information about the system under study.

Over the ages, different measures have been proposed to quantify uncertainty in a variety of systems. In this work, we confine our discussion to two such measures of central importance: standard deviation and Shannon information entropy.

1.3.1 Standard Deviation

An oft-sufficient but unrefined procedure is to evaluate an average value $\langle Q \rangle$ of an observable Q. The standard deviation[15] is then calculated as:

$$\sigma_Q = \sqrt{\langle Q - \langle Q \rangle^2 \rangle} \tag{1.2}$$

The undeniable distinction between the classical and the quantum worlds arises from the probabilistic nature of the latter. Thus, uncertainty in most measurements cannot be indefinitely decreased in quantum physics.

The state of a quantum particle in position space is completely described by a wavefunction $\psi(\mathbf{r})$. The square of the modulus of this wavefunction $\rho(\mathbf{r}) = |\psi(\mathbf{r})|^2$ determines the probability distribution of the position of the particle.

Since the probability to find the particle anywhere must equal unity,

$$\int_{\mathbb{R}^3} d^3r |\psi(\mathbf{r})|^2 = 1$$

The classical position $\mathbf{r}_{cl}$ is best associated with the average value:

$$\mathbf{r}_{cl} = \langle \mathbf{r} \rangle = \int_{\mathbb{R}^3} d^3r \mathbf{r} \rho(\mathbf{r})$$

Therefore,

$$\sigma_x = [\int_{\mathbb{R}^3} d^3r(x - \langle x \rangle)^2 \rho(\mathbf{r})]^{\frac{1}{2}} \tag{1.3}$$

Fourier transformation yields us the probability distribution of the same state in momentum space. We use a normalized (by Parseval-Plancherel theorem[16]) Fourier convention of the following form:

$$\psi(\mathbf{p}) = \int_{\mathbb{R}^3} \frac{d^3r}{(2\pi\hbar)^{\frac{3}{2}}} \exp\left(\frac{-i\mathbf{p}.\mathbf{r}}{\hbar}\right)\psi(\mathbf{r}) \tag{1.4}$$

Therefore,

$$\sigma_p = [\int_{\mathbb{R}^3} d^3p(p - \langle p \rangle)^2 \rho(\mathbf{p})]^{\frac{1}{2}} \tag{1.5}$$

Even though for different states both standard deviations σ_x and σ_p can individually be arbitrarily small, they become correlated when calculated for the same state. This correlation is usually expressed by the famous Heisenberg uncertainty relation[8]:

$$\sigma_x \sigma_p \geq \frac{\hbar}{2} \tag{1.6}$$

The bound in this inequality is saturated by Gaussian wavefunctions.

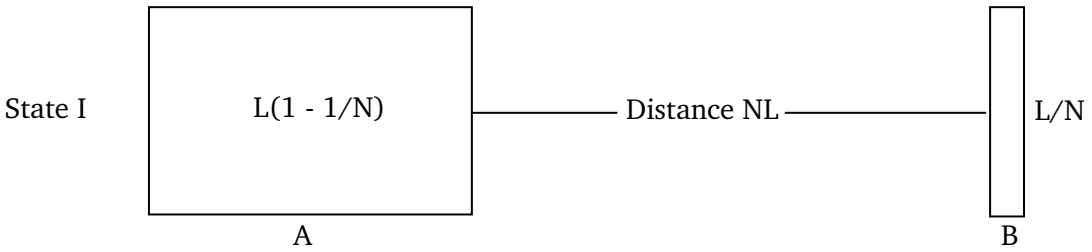
1.3.1.1 Problems with Standard Deviation

Standard deviation, despite its ease of interpretability and universal application, is not an ideal tool and fails miserably under select circumstances. Two such scenarios are furnished, exemplifying the shortcomings of standard deviation as a measure of localization[17]:

- Two regions on the real axis are considered. The first region is the line segment $A = [L(N + \frac{1}{N}), L(N + 1)]$ with its length equal to $L(1 - \frac{1}{N})$ where L is the unit length and N is a large number. The second region B is the line segment with its length equal to $\frac{L}{N}$ separated from the first region by the large distance NL.

  The probability distribution is of the form:

$$\rho(x) = \begin{cases} \frac{1}{L} & x \in A \\ \frac{1}{L} & x \in B \\ 0 & \text{elsewhere} \end{cases}$$



The standard deviation of the variable x is thus:

$$\sigma_x{}^2(\mathbf{I}) = L^2(N - \frac{1}{N} + \frac{1}{12})$$

In the case N is sufficiently large, this becomes:

$$\sigma_x(\mathbf{I}) \approx L\sqrt{N}$$

While common sense predicts that the uncertainty should remain finite as $N \longrightarrow \infty$, $\sigma_x \longrightarrow \infty$. It is simply the effect of a growing distance between the two regions. Standard deviation is based on the second moment, very sensitive to the values of an observable lying far away from the average.

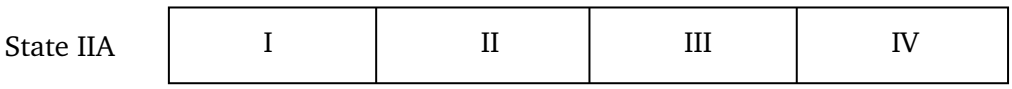
- In the previous example the size of the space where the probability distribution is constant is preserved. Now this condition is omitted.

  Two contrasting scenarios are considered. In case A, the particle is localized with a uniformly distributed probability in a box of length L, while in case B, the particle is localized with equal probabilities in two smaller boxes each of length $\frac{L}{4}$.

  The probability distributions are of the form:

$$\rho_A(x) = \begin{cases} \frac{1}{L} & x \in [0, L] \\ 0 & \text{elsewhere} \end{cases}$$

$$\rho_B(x) = \begin{cases} \frac{2}{L} & x \in [0, \frac{L}{4}] \\ \frac{2}{L} & x \in [0, L] \\ 0 & \text{elsewhere} \end{cases}$$



In the case A the size of the space where the particle can be found is L, while in the case B this size is $\frac{L}{2}$. Thus, in the case B we know more about position than in the case A. However, calculating the standard deviations, the uncertainty of the position is greater in the case B.

$$\sigma_x(\textbf{IIA}) = \frac{L}{\sqrt{12}}$$

$$\sigma_x(\textbf{IIB}) = \sqrt{\frac{7}{4}}\frac{L}{\sqrt{12}}$$

This is another manifestation of the fact that the gap between two regions (line segment $[\frac{L}{4}, 3\frac{L}{4}]$) where the particle cannot be found contributes to the standard deviation.

1.3.2 Shannon Entropy

C.E. Shannon connected the measure of the information content with a probability distribution[12]. Suppose there exists a set of possible events whose probabilities of occurrence are p_1, p_2,..., p_n. The goal is to choose a measure of how much choice is involved in the selection of the event or, in other words, how uncertain the outcome is.

If there exists such a measure, $H(p_1, p_2,..., p_n)$, it should satisfy the following properties:

1. A small perturbation in a single p_i keeping the others constant should result in a corresponding small change in H. Thus, H is continuous in each p_i.

2. When all events are equally likely, there is more choice or uncertainty when there are more possible events. Thus, if $p_i = \frac{1}{N}\ \forall i$, then H is a monotonic increasing function of n.

3. If a choice is able to be broken down into multiple successive choices, H is the weighted sum of the individual values of H.

1.3.2.1 Discrete Distributions: Shannon's Theorem

The only H satisfying the three properties above is of the form:

$$H = -K \sum_{i=1}^{n} p_i \log p_i \tag{1.7}$$

where K is a positive constant.

Let $A(n) = H(\frac{1}{N}, ..., \frac{1}{N})$. The third property allows the decomposition of a choice from s^m equally likely possibilities into a series of m choices from s equally likely possibilities. Thus, $A(s^m) = mA(s)$ and $A(t^n) = nA(t)$.

n can be chosen to be arbitrarily large and accordingly, an m is chosen to satisfy the relation:

$$s^m \leq t^n \leq s^{m+1}$$

Taking logarithm on all sides,

$$m \log s \leq n \log t \leq (m+1) \log s$$

Dividing all sides by $n \log s$,

$$\frac{m}{n} \leq \frac{\log t}{\log s} \leq \frac{m}{n} + \frac{1}{N}$$

Since n is arbitrarily large, this implies

$$|\frac{m}{n} - \frac{\log t}{\log s}| < \epsilon$$

where $\epsilon > 0$ is arbitrarily small.

By the second property, $A(n)$ is monotonically increasing. Hence

$$A(s^m) \leq A(t^n) \leq A(s^{m+1})$$

i.e.,

$$mA(s) \leq nA(t) \leq (m+1)A(s)$$

Dividing all sides by $nA(s)$,

$$\frac{m}{n} \leq \frac{A(t)}{A}(s) \leq \frac{m}{n} + \frac{1}{N}$$

i.e.,

$$|\frac{m}{n} - \frac{A(t)}{A}(s)| < \epsilon$$

This implies

$$|\frac{A(t)}{A}(s) - \frac{\log t}{\log s}| < 2\epsilon$$

i.e.,

$$A(t) = K \log t$$

where K must be positive to satisfy the monotonically increasing property of $A(n)$.

Now suppose there is a choice from n possibilities with commensurable probabilities $p_i = \frac{n_i}{\sum n_i}$, where the n_i are integers. A choice from $\sum n_i$ possibilities can be decomposed into a choice from n possibilities with probabilities p_1, p_2,..., p_n and then, if the i^{th} was chosen, a choice from n_i with equal probabilities.

Again using the third property,

$$K \log \sum n_i = H(p_1, ..., p_n) + K \sum p_i \log n_i$$

$$\Rightarrow H = K[\sum p_i \log \sum n_i - \sum p_i \log n_i]$$

$$\Rightarrow H = -K \sum p_i \log \frac{n_i}{\sum n_i} = -K \sum p_i \log p_i$$

In case the p_i are not commensurable, they are able to be approximated by rationals and by the continuity of H, the same expression holds good. The choice of the coefficient K is equivalent to the choice of a unit of measure.

1.3.2.2 Applicability in Quantum Systems

The set of probabilities obtained from quantum mechanics are inserted into the Shannon formula for information entropy:

$$H = -K \sum_{i=1}^{n} p_i \log p_i$$

The whole region of interest is divided into bins—equal segments of size δx. The bin size will be viewed as a measure of the experimental error.

For a pure state, the probability to find the particle in the i^{th} bin is:

$$q_i = \int_{i-\frac{\delta x}{2}}^{i+\frac{\delta x}{2}} dx \rho(x)$$

Therefore, the uncertainty in position is:

$$H^{(x)} = -\sum_{i=-\infty}^{\infty} q_i \log q_i \tag{1.8}$$

Similarly, the probability to find the momentum k_x in the j^{th} bin is:

$$p_j = \int_{j-\frac{\delta k}{2}}^{j+\frac{\delta k}{2}} dk \rho(k)$$

Thus, the uncertainty in momentum is:

$$H^{(k)} = -\sum_{j=-\infty}^{\infty} p_j \log p_j \tag{1.9}$$

As has been argued by Bohr and Heisenberg, it is impossible to measure the position without losing information about the momentum and vice versa. This property is embodied in the standard Heisenberg uncertainty relation and can be expressed in terms of Shannon measures of uncertainty[18]:

$$H^{(x)} + H^{(p)} > 1 - \log 2 - \log \frac{\delta x \delta p}{h} \tag{1.10}$$

1.3.2.3 Continuous Distributions

Let us choose the distribution function for position $\rho(x)$. If $\rho(x)$ does not change appreciably over the distance dx, the Shannon entropy can be approximated by:

$$H^{(x)} \approx -\sum_{k=-\infty}^{\infty} \delta x \rho(x_k) \log |\rho(x_k)\delta x|$$

where $x_k = k\delta x$

$$\Rightarrow H^{(x)} \approx -\sum_{k=-\infty}^{\infty} \delta x \rho(x_k) \log |\rho(x_k)L| - \log \frac{\delta x}{L}$$

where L is some fixed unit of length. The first term is a Riemann sum and when $\delta x \longrightarrow 0$, it tends to the following integral:

$$S^{(x)} = -\int_{\mathbb{R}} \delta x \rho(x) \log |\rho(x) L| \tag{1.11}$$

This integral can be termed as the entropy of the continuous distribution $\rho(x)$ [10]. The second term $-\log \frac{\delta x}{L}$ measures the difference between $H^{(x)}$ and $S^{(x)}$. This must tend to ∞ since the information measured by $H^{(x)}$ grows indefinitely when $\delta x \longrightarrow 0$, while $S^{(x)}$ remains finite.

1.3.2.4 The Principle of Maximum Entropy

The principle of maximum entropy is used to estimate input probabilities of a probability distribution[19]. The result is consistent with known constraints expressed in terms of expected values of one or more quantities, but as unbiased as possible.

The principle,formulated by E. T. Jaynes, states that the probability distribution having maximal entropy with respect to prior constraints best represents the current state of knowledge.

1.3.2.5 The Biased Dice Problem

Consider a n-sided biased dice, about which the only information available is the expectation value (mean) of its rolls (where the sides are numbered from 1 onwards.) The principle of maximum entropy is used to assign probabilities to the rolls of the different sides of this dice.

A numerical approach using $Python\ 2.7$ for solving the biased dice probabilities is provided in the appendix in **Listing** 4.1.

Figure 1.1 shows the estimated input probabilities for the biased dice with $N = 6$ and $\mu = 3$. The dotted line represents the uniform distribution of the unbiased dice. The strictly decreasing trend is caused by the mean being lower than the unbiased mean. Any violation of monotonicity for a biased dice would fail entropy maximization.

An analytical avenue is furnished below.

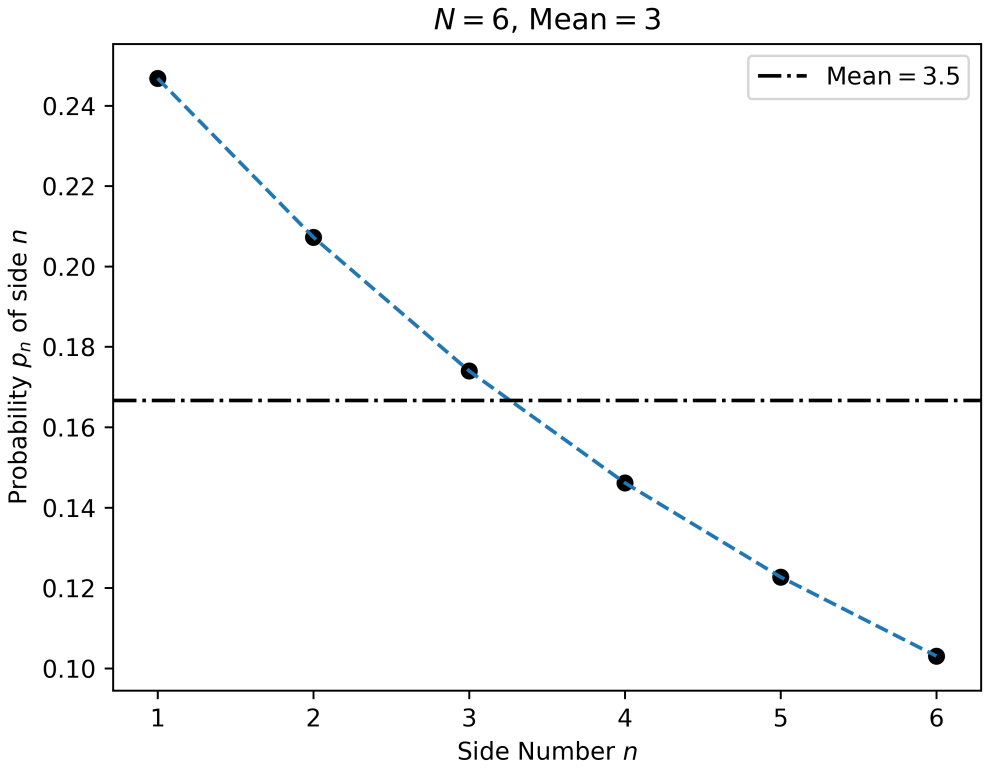


Fig. 1.1 – Input Probabilities for Biased Dice with $N = 6, \mu = 3$.

The constraints of the system are:

$$g(p_1, p_2, ..., p_n) = \sum_{i=1}^{n} p_i = 1 \Longrightarrow \sum_{i=1}^{n} dp_i = 0 \tag{1.12}$$

$$p_i \geq 0 \forall i \tag{1.13}$$

$$h(p_1, p_2, ..., p_n) \sum_{i=1}^{n} p_i i = M \Longrightarrow \sum_{i=1}^{n} dp_i i = 0 \tag{1.14}$$

We seek the distribution $(p_1, p_2, ..., p_n) = (p'_1, p'_2, ..., p'_n)$ that causes the entropy function, $S(p_1, p_2, ..., p_n) = -k \sum_i p_i \log p_i$ to be at its maximum possible value. Since k has the role of a constant, let us assume $k = 1$.

To solve it by the Lagrange multiplier method with multipliers α and β, we want:

$$\frac{\partial S}{\partial p_i} - \alpha \frac{\partial g}{\partial p_i} - \beta \frac{\partial h}{\partial p_i} = 0 \forall i$$

The partial derivatives are evaluated for each p_i:

$$\frac{\partial S}{\partial p_i} = -1 - \log p_i$$

$$\frac{\partial g}{\partial p_i} = 1$$

$$\frac{\partial h}{\partial p_i} = i$$

Substituting,

$$-1 - \log p_i' - \alpha - \beta i = 0 \Rightarrow p_i' = e^{(-1-\alpha-\beta i)}$$

To eliminate α from the equation,

$$p_i' = \frac{p_i'}{\sum_{i=1}^{n} p_i'} = \frac{e^{(-1-\alpha-\beta i)}}{\sum_{i=1}^{n} e^{(-1-\alpha-\beta i)}}$$

$$\Rightarrow p_i' = \frac{e^{-\beta i}}{\sum_{i=1}^{n} e^{-\beta i}}$$

Thus,

$$M = \sum_{i=1}^{n} p_i' i = \frac{\sum_{i=1}^{n} e^{-\beta i} i}{q}$$

where $q = \sum_{i=1}^{n} e^{-\beta i}$ is called the partition function.

In our present case,

$$n = 6 \Rightarrow q = (x + x^2 + x^3 + x^4 + x^5 + x^6)$$

where $x = e^{-\beta}$

Therefore,

$$M = 3 = \frac{x + 2x^2 + 3x^3 + 4x^4 + 5x^5 + 6x^6}{x + x^2 + x^3 + x^4 + x^5 + x^6}$$

Solving for x, we have $x = 0.839769$. Thus, $q = 3.402876$.

Then, the probabilities are

$$\frac{x^i}{q} \approx [0.246782, 0.207239, 0.174033, 0.146148, 0.122730, 0.103065]$$

References

1. Liu, S. On the Relationship between Densities of Shannon Entropy and Fisher Information for Atoms and Molecules. *J. Chem. Phys.* **2007**, *126* (19), 191107.

2. Esquivel, R. O.; Flores-Gallegos, N.; Iuga, C.; Carrera, E. M.; Angulo, J. C.; Antolín, J. Phenomenological Description of the Transition State, and the Bond Breaking and Bond Forming Processes of Selected Elementary Chemical Reactions: An Information-Theoretic Study. *Theor. Chem. Acc.* **2009**, *124* (5–6), 445–460.

3. Sollner, T. C. L. G.; Goodhue, W. D.; Tannenwald, P. E.; Parker, C. D.; Peck, D. D. Resonant Tunneling through Quantum Wells at Frequencies up to 2.5 THz. *Appl. Phys. Lett.* **1983**, *43* (6), 588–590.

4. Wheeler, J. A. On the Mathematical Description of Light Nuclei by the Method of Resonating Group Structure. *Phys. Rev.* **1937**, *52* (11), 1107–1122.

5. Razavy, M. Quantum Theory of Tunneling. *World Scientific, Singapore*. **2003**.

6. Schumm, T.; Hofferberth, S.; Andersson, L. M.; Wildermuth, S.; Groth, S.; Bar-Joseph, I.; Schmiedmayer, J.; Krüger, P. Matter-Wave Interferometry in a Double Well on an Atom Chip. *Nat. Phys.* **2005**, *1* (1), 57–62.

7. Niquet, Y. M.; Allan, G.; Delerue, C.; Lannoo, M. Quantum Confinement in Germanium Nanocrystals. *Appl. Phys. Lett.* **2000**, *77* (8), 1182–1184.

8. Heisenberg, W. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Eur. Phys. J. A* **1927**, *43* (3–4), 172–198.

9. Vidal, J.; Doucot, B.; Mosseri, R.; Butaud, P. Interaction Induced Delocalization for Two Particles in a Periodic Potential. *Phys. Rev. Lett.* **2000**, *85* (18), 3906-3909.

10. Bialynicki-Birula, I.; Rudnicki, Ł. Entropic Uncertainty Relations in Quantum Physics. In *Statistical Complexity*; Springer Netherlands: Dordrecht, **2011**; pp 1–34.

11. Kumar, N.; Raj, P.; Balanarayan, P. Atop-the-barrier Localization in Periodically Driven Double Wells: A Minimization of Information Entropic Sums in Conjugate Spaces. *Int. J. Quantum Chem.* **2020**, *120* (7).

12. Shannon, C. E. A Mathematical Theory of Communication. *Bell Syst. tech. j.* **1948**, *27* (4), 623–656.

13. Białynicki-Birula, I.; Mycielski, J. Uncertainty Relations for Information Entropy in Wave Mechanics. *Commun. Math. Phys.* **1975**, *44* (2), 129–132.

14. Maheswari, A. U. Study of Resonances and Absorption in One Dimensional Quantal Scattering. *Ph.D. Dissertation* **2011**.

15. Pearson, K. III. Contributions to the Mathematical Theory of Evolution. *Philos. Trans. R. Soc. Lond. A* **1894**, *185* (0), 71–110.

16. Plancherel, M.; Leffler, M. Contribution Á L'Étude de la reprÉsentation D'une fonction arbitraire par des intÉgrales dÉfinies. *Rend. Circ. Mat. Palermo (2)* **1910**, *30* (1), 289–335.

17. Bialynicki-Birula, I. Rényi Entropy and the Uncertainty Relations. In *AIP Conference Proceedings*; AIP, **2007**.

18. Bialynicki-Birula, I. Entropic Uncertainty Relations. *Phys. Lett. A* **1984**, *103* (5), 253–254.

19. Jaynes, E. T. Information Theory and Statistical Mechanics. *Phys. Rev.* **1957**, *106* (4), 620–630.

2

Uncertainty Relations

Pure states of an N-level system are described in quantum mechanics by normalized vectors in the N-dimensional Hilbert space $\mathscr{H}_N$. Let us choose two orthonormal bases $|a_i\rangle$ and $|b_j\rangle$ where $i = 1, 2, ..., N$.

The squares of the absolute values of the complex expansion coefficients are interpreted in quantum mechanics as the probabilities, say q_i and p_j, to find the states represented by the vectors $|a_i\rangle$ and $|b_j\rangle$:

$$q_i = |\langle a_i|\psi\rangle|^2$$

$$p_j = |\langle b_j|\psi\rangle|^2$$

The normalization of vectors and the orthonormality of the bases guarantee that the probabilities q_i and p_j sum up to 1.

2.1 Deutsch inequality

From the sets of position and momentum probabilities q_i and p_j, two Shannon entropies can be constructed:

$$H^{(a)} = -\sum_{i=1}^{N} q_i \log q_i$$

$$H^{(b)} = -\sum_{j=1}^{N} p_j \log p_j$$

Their sum is

$$H^{(a)} + H^{(b)} = \sum_{i=1}^{N}\sum_{i=1}^{N} q_i p_j Q_{ij}$$

where

$$Q_{ij} = -\log\left(\langle a_i|\psi\rangle\langle\psi|a_i\rangle\right) - \log\left(\langle b_j|\psi\rangle\langle\psi|b_j\rangle\right)$$

Deutsch found a lower bound on this sum[1]. For a normalized vector $|\psi\rangle$ each Q_{ij} is nonnegative. Therefore, by finding the minimal value of all Q_{ij}'s a lower bound on the sum can be determined. The function $|\psi_{ij}\rangle$ for which the minimal value is obtained can be found by varying Q_{ij} with respect to ψ with the normalization constraint $\langle\psi|\psi\rangle = 1$.

Thus,

$$\frac{\partial}{\partial|\psi_{ij}\rangle}[Q_{ij} + \kappa\,\langle\psi_{ij}|\psi_{ij}\rangle] = 0$$

where κ is a Lagrange multiplier.

$$\Rightarrow \psi_{ij} = \frac{1}{\kappa}(\frac{|a_i\rangle}{\langle\psi_{ij}|a_i\rangle} + \frac{|b_j\rangle}{\langle\psi_{ij}|b_j\rangle})$$

Taking scalar product with $\langle\psi_{ij}|$, $\kappa = 2$ is obtained.

Let $a = \langle\psi_{ij}|a_i\rangle$ and $b = \langle\psi_{ij}|b_j\rangle$.

$$\Rightarrow |a|^2 = \frac{1}{2} + \frac{a\,\langle a_i|b_j\rangle}{2b}$$

$$|b|^2 = \frac{1}{2} + \frac{b\,\langle b_j|a_i\rangle}{2a}$$

Therefore, $\frac{a\langle a_i|b_j\rangle}{2b}$ and $\frac{b\langle b_j|a_i\rangle}{2a}$ must be real.

Dividing,

$$\frac{|a|^2}{|b|^2} - 1 = \frac{a}{b}\,\langle a_i|b_j\rangle - \frac{a^*}{b^*}\langle a_i|b_j\rangle^*$$

$$\Rightarrow |a| = |b| \Rightarrow |a|^2 = \frac{1}{2}[1 \pm |\,\langle a_i|b_j\rangle\,|]$$

Choosing the plus sign,

$$|\psi_{ij}\rangle = \frac{\exp(i\phi)}{\sqrt{2[1 \pm |\,\langle a_i|b_j\rangle\,|]}}[|a_i\rangle + \exp(-i\arg\langle a_i|b_j\rangle)\,|b_j\rangle]$$

where $\exp(i\phi)$ is an arbitrary phase factor.

Inserting into the original sum as a minimizer:

$$H^{(a)} + H^{(b)} \geq -2\sum_{i=1}^{N}\sum_{i=1}^{N} q_i p_j \log[\frac{1}{2}(1 \pm |\,\langle a_i|b_j\rangle\,|)]$$

The inequality holds if each $|\langle a_i|b_j\rangle|$ is replaced by $C_B = \sup_{(i,j)} |\langle a_i|b_j\rangle|$.

Therefore, the Deutsch inequality is obtained:

$$H^{(a)} + H^{(b)} \geq -2\log[\frac{1}{2}(1 + C_B)] \tag{2.1}$$

2.2 Maassen-Uffink inequalities

The bound in the Deutsch uncertainty relation is saturated when the two bases have common vectors. However, there is room for improvement in the bound. An improved relation was conjectured by Kraus[2] and later proved by Maassen and Uffink[3]:

$$H^{(a)} + H^{(b)} \geq -2\log C_B \tag{2.2}$$

The Riesz theorem[4] states that for every N-dimensional complex vector X and a unitary transformation matrix $T = (t_{ji})$, the following inequality between the norms is valid:

$$c^{\frac{1}{\mu}}||X||^{\mu} \leq c^{\frac{1}{\nu}}||TX||^{\nu} \tag{2.3}$$

where

- $c = \sup_{(i,j)} |t_{ji}|$
- $\frac{1}{\mu} + \frac{1}{\nu} = 1$
- $1 \leq \nu \leq 2$
- $||X||^{\mu} := [\sum_k |x_k{}^{\mu}|]^{\frac{1}{\mu}}$

Let $x_i = \langle a_i|\psi\rangle$ and $t_{ji} = \langle b_j|a_i\rangle$.

Now,

$$\sum_i |a_i\rangle\langle a_i| = 1 \longrightarrow \sum_{i=1}^{N} t_{ji}x_i = \langle b_j|\psi\rangle$$

Using the definitions of q_i and p_j and the Riesz theorem,

$$c^{\frac{1}{\mu}}[\sum_{j=1}^{N} q_j{}^{\frac{\mu}{2}}]^{\frac{1}{\mu}} \leq c^{\frac{1}{\nu}}[\sum_{i=1}^{N} p_i{}^{\frac{\nu}{2}}]^{\frac{1}{\nu}}$$

where $c = \sup(i,j)|t_{ij}| = \sup(i,j)|\langle a_i|b_j\rangle| = C_B$

Also, $\mu = 2\alpha$, $\nu = 2\beta$ where α and β are conjugate parameters.

Finally, taking logarithm:

$$H_\alpha{}^{(a)} + H_\beta{}^{(b)} \geq -2\log C_B$$

In the limit $\alpha \longrightarrow 1$ and $\beta \longrightarrow 1$, this inequality reduces to the Maassen-Uffink result.

2.3 Logarithmic Sobolev inequality

The (Gross-Nelson) logarithmic Sobolev inequality[5] for the Shannon entropy reads:

$$S^{(x)} \geq \frac{1}{2}(1+\log 2\pi) - \frac{1}{2}\log\left[L^2 \int_{\mathbb{R}} dx \frac{1}{\rho(x)} \left|\frac{d\rho(x)}{dx}\right|^2\right] \tag{2.4}$$

The salient feature of this inequality is that there need be only one function $\rho(x)$. Therefore, we do not need conjugate variables to obtain a lower bound of the Shannon entropy.

On the other hand, the right hand side depends on $\rho(x)$, so it is a functional relation rather than an uncertainty one. In order to obtain an uncertainty relation, the inverse logarithmic Sobolev inequality[6] is exploited:

$$S^{(x)} \leq \frac{1}{2}(1+\log 2\pi) + \log L\sigma_x$$

In momentum space,

$$S^{(k)} \leq \frac{1}{2}(1+\log 2\pi) + \log\frac{\sigma_k}{L}$$

Hence, taking exponential of these two inequalities,

$$\sigma_x\sigma_k \geq \frac{1}{2}\exp\left(S^{(x)} + S^{(k)} - 1 - \log\pi\right) \geq \frac{1}{2} \tag{2.5}$$

This uncertainty relation is stronger than the standard Heisenberg uncertainty relation[7]. Whenever the sum of the Shannon entropies exceeds its lower bound $1+\log p$, this is a stronger bound.

The uncertainty relation also holds for any pair of distributions, not only ones related by the Fourier transformation of the wavefunctions.

2.4 The Białynicki-Birula Mycielski inequality

. For normalized wavefunctions, the uncertainty relation has the form[8]:

$$-\langle \log \varrho \rangle - \langle \log \varrho' \rangle \geq n(1 + \log \pi) \tag{2.6}$$

where ϱ and ϱ' are probability densities in n-dimensional position space and momentum space respectively:

$\varrho(\mathbf{r}) = |\psi(\mathbf{r})|^2$; $\varrho'(\mathbf{k}) = |\psi'(\mathbf{k})|^2$

The chosen normalization of the Fourier transform is:

$$\psi'(\mathbf{k}) = \frac{1}{(2\pi)^{\frac{n}{2}}} \int d^n r \exp(-i\mathbf{k}.\mathbf{r})\psi(\mathbf{r}) \tag{2.7}$$

This inequality does not depend on the unit of length used in measuring ϱ and ϱ'.

2.4.1 Proof of the inequality

The (p, q)-norm of the Fourier transformation is defined to be the smallest number $k(p, q)$ for which the following holds good $\forall\ \psi \in L^p$:

$$||\psi'||_q \leq k(p, q)||\psi||_p$$

where:

- $||\psi||_p = (\int d^n r|\psi|^p)^{\frac{1}{p}}$
- $\frac{1}{p} + \frac{1}{q} = 1$
- $q \geq 2$

The value of $k(p, q)$ has been calculated [9] and extended to non-integral values of q [10]:

$$k(p, q) = (\frac{2\pi}{q})^{\frac{n}{2q}}.(\frac{2\pi}{p})^{-\frac{n}{2p}} \tag{2.8}$$

Therefore,

$$W(q) \equiv k(p, q)||\psi||_p - ||\psi'||_q \geq 0$$

where p is a function of q.

Since by the Parseval-Plancherel theorem[11], $W(2) = 0$, the right-derivative of $W(q)$ at $q = 2$ must be non-negative, i.e.,

$$-\frac{n}{4}N(1+\log\pi)-\frac{1}{2}N^{-1}\int d^n r|\psi(\mathbf{r})|^2\log|\psi(\mathbf{r})|-\frac{1}{2}N^{-1}\int d^n r|\psi'(\mathbf{k})|^2\log|\psi'(\mathbf{k})|+N\log N\geq 0$$

where $N = ||\psi||_2 = ||\psi'||_2$.

Choosing $N = 1$ yields the desired inequality.

2.4.2 Relations to other inequalities

Let us first calculate the maximum value of the position-space entropy subject to:

- $\langle 1\rangle = 1$
- $\langle(\mathbf{r}-\langle r\rangle)^2\rangle = {r_0}^2$

This is equivalent to finding the extremum of the following functional of ϱ:

$$-\langle\log\varrho\rangle-\lambda(\langle 1\rangle-1)-\mu(\langle(\mathbf{r}-\langle r\rangle)^2\rangle-{r_0}^2)$$

where λ and μ are Lagrange multipliers.

Equating the variation of this functional with respect to ϱ to zero:

$$\log\varrho(\mathbf{r})+1+\lambda+\mu r^2-2\mu\langle r\rangle.\mathbf{r}=0$$

This extremum is a true maximum as entropy is a strictly concave functional of ϱ. The Lagrange multipliers λ and μ can be determined using the above constraints:

$$\varrho_{max}(\mathbf{r}) = (r_0(\frac{2\pi}{n})^{\frac{1}{2}})^{-n}\exp\frac{-n\langle(\mathbf{r}-\langle r\rangle)^2\rangle}{2{r_0}^2}$$

and

$$-\langle\log\varrho\rangle_{max} = \frac{n}{2}\log(2\pi e{r_0}^{\frac{2}{n}})$$

The following inequality has thus been proved:

$$-\langle \log \varrho \rangle \leq \frac{n}{2} \log \left(2\pi e r_0{}^{\frac{2}{n}}\right)$$

which can also be written as:

$$e\pi \exp(\frac{2}{n} \langle \log \varrho \rangle) \geq \frac{n}{2} (\langle (\mathbf{r} - \langle r \rangle)^2 \rangle)^{-1}$$

Owing to symmetry between the r and k spaces, the same inequality can be written for momentum-space entropy and dispersion in the k variable. In the inverted form:

$$\frac{2}{n} (\langle (\mathbf{k} - \langle k \rangle)^2 \rangle) \geq (e\pi)^{-1} \exp(-\frac{2}{n} \langle \log \varrho' \rangle)$$

To summarize, the following chain of inequalities hold:

$$\frac{2}{n} (\langle (\mathbf{k} - \langle k \rangle)^2 \rangle) \geq (e\pi)^{-1} \exp(-\frac{2}{n} \langle \log \varrho' \rangle) \geq e\pi \exp(\frac{2}{n} \langle \log \varrho \rangle) \geq \frac{n}{2} (\langle (\mathbf{r} - \langle r \rangle)^2 \rangle)^{-1} \tag{2.9}$$

After the identification $p = \hbar k$, the first and the last part of this chain give the Heisenberg uncertainty in n dimensions. Taking only the first and the third term, the logarithmic Sobolev inequality[5] is obtained.

All three inequalities become exact equalities for every Gaussian wavefunction. Moreover, it follows from variational calculation that the first and the third inequalities become equalities only for Gaussian functions. This is also the case for the second inequality[8].

2.5 Gaussian Wavefunctions

Gaussian wavefunctions play a unique role in uncertainty relations[12]. They saturate the standard uncertainty relations expressed in terms of standard deviations as well as the Shannon entropies for continuous variables. Thus, the continuous uncertainty relations are sharp unlike their discrete counterparts.

Additionally, Gaussians provide a direct link between the Shannon entropies for continuous variables and the uncertainties measured by the standard deviation. The Gaussian

distribution function gives the maximal value of the Shannon entropy subject to normalization and a given σ_x [12]. To prove this, the maximum of the following functional is required:

$$\int_{\mathbb{R}} dx[-\rho(x)\log\rho(x) + \lambda x^2\rho(x) + \mu\rho(x)]$$

where λ and μ are Lagrange multipliers satisfying the constraints:

- $\int_{\mathbb{R}} dx\rho(x) = 1$
- $\int_{\mathbb{R}} dx x^2\rho(x) = \sigma_x^2$

$$\Rightarrow -1 - \log\rho(x) + \lambda x^2 + \mu = 0$$

$$\Rightarrow \rho(x) = \frac{1}{\sqrt{2\pi}\sigma_x}\exp -\frac{x^2}{2\sigma_x^2}$$

This is in the form of a Gaussian distribution. This extremum is a true maximum since the entropy is a strictly concave functional of $\rho(x)$ [8].

References

1. Deutsch, D. Uncertainty in Quantum Measurements. *Phys. Rev. Lett.* **1983**, *50* (9), 631–633.
2. Kraus, K. Complementary Observables and Uncertainty Relations. *Phys. Rev. D Part. Fields* **1987**, *35* (10), 3070–3075.
3. Maassen, H.; Uffink, J. B. Generalized Entropic Uncertainty Relations. *Phys. Rev. Lett.* **1988**, *60* (12), 1103–1106.
4. Riesz, M. Sur Les Maxima Des Formes Bilinéaires et Sur Les Fonctionnelles Linéaires. *Acta Math.* **1926**, *49* (3–4), 465–497.
5. Gross, L. Logarithmic Sobolev Inequalities. *Amer. J. Math.* **1975**, *97* (4), 1061.
6. Chafaï, D. Gaussian Maximum of Entropy and Reversed Log-Sobolev Inequality. *Séminaire de probabilités de Strasbourg* **2002**, *36*, 194–200.
7. Dodonov, V. V.; Man'ko, O. V. Universal Invariants in Quantum Mechanics and Physics of Optical and Particle Beams. *J. Russ. Laser Res.* **2000**, *21* (5), 438–464.
8. Białynicki-Birula, I.; Mycielski, J. Uncertainty Relations for Information Entropy in Wave Mechanics. *Commun. Math. Phys.* **1975**, *44* (2), 129–132.

9. K. I. Babenko, An inequality in the theory of Fourier integrals, *Izv. Akad. Nauk SSSR Ser. Mat.*, **1961**, *25* (4) 531–542.

10. Beckner, W. Inequalities in Fourier Analysis on R. *Proc. Natl. Acad. Sci. U. S. A.* **1975**, *72* (2), 638–641.

11. Plancherel, M.; Leffler, M. Contribution Á L'Étude de la reprÉsentation D'une fonction arbitraire par des intÉgrales dÉfinies. *Rend. Circ. Mat. Palermo (2)* **1910**, *30* (1), 289–335.

12. Bialynicki-Birula, I.; Rudnicki, Ł. Entropic Uncertainty Relations in Quantum Physics. In *Statistical Complexity*; Springer Netherlands: Dordrecht, **2011**; pp 1–34.

3 Methodology

3.1 Notations and Conventions

System of Units: The atomic natural unit system uses the following constants to have numeric value 1 in terms of the resulting units:

$$c, m_e, \hbar, \epsilon_0$$

where c is the speed of light, m_e is the electron mass, $\hbar$ is the reduced Planck's constant, and ϵ_0 is the vacuum permittivity.

Variables and Units in terms of SI:

- $\psi(x)$: Position wavefunction of the particle
- $U(x)$: Height of the potential ($m_e c^2 \approx 8.187 \times 10^{-13} J$)[1]
- E: Energy of the particle (same unit as $U(x)$)
- M: Mass of the particle ($m_e \approx 9.109 \times 10^{-31} Kg$)[2]
- L: Length of the well or the barrier ($\frac{\hbar}{m_e c} \approx 3.862 \times 10^{-13} m$)[3]
- t: Time ($\frac{\hbar}{m_e c^2} \approx 1.288 \times 10^{-21} s$)[4]

3.2 Stationary Wavefunctions

The one-dimensional time-independent Schrödinger equation reads:

$$E\psi(x) = U(x)\psi(x) - \frac{\hbar^2}{2M}\frac{d^2\psi(x)}{dx^2} \tag{3.1}$$

Since $U(x)$ is piece-wise constant, $\psi(x)$ is defined piece-wise on these regions with the appropriate index which are then determined by solving the above differential equation and matching appropriate boundary conditions related to the continuity of the wavefunction and its derivative. Then, the composite function is numerically normalized.

To obtain the momentum space wavefunctions, a Fourier transformation of $\psi(x)$ needs to be conducted into momentum space to take it to $\psi'(p)$.

By our normalized (by Parseval-Plancherel theorem[5]) Fourier convention:

$$\psi'(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \psi(x) e^{-ikx} dx \tag{3.2}$$

A numerical implementation of a Fast Fourier Transform algorithm is adopted for efficiency wherever possible.

3.3 Propagation of Gaussian Wavepacket

The time-dependent Schrödinger equation that is responsible for governing the dynamics of any one-dimensional quantum system reads:

$$i\hbar \frac{\delta \psi(x,t)}{\delta t} = U(x)\psi(x,t) - \frac{\hbar^2}{2M} \frac{\delta^2 \psi(x,t)}{\delta x^2} \tag{3.3}$$

The wavefunction ψ is a function of both position x and time t, and contains the complete description of the particle in position space.

Extending our Fourier convention to the time-dependent case:

$$\psi'(k,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \psi(x,t) e^{-ikx} dx \tag{3.4}$$

The associated inverse Fourier Transform by the convention adopted is:

$$\psi(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \psi'(k,t) e^{ikx} dk \tag{3.5}$$

Substituting this into the Schrödinger equation and simplifying gives the momentum space form of the Schrödinger equation:

$$i\hbar \frac{\delta \psi}{\delta t} = \frac{\hbar^2 k^2}{2M} \psi + V(i\frac{\delta}{\delta k})\psi \tag{3.6}$$

The two versions of the Schrödinger equation are similar in the sense that time evolution involves the wavefunction ψ, as well as its derivative with respect to x or k. It is arduous to evaluate the equation fully within any one representation; however, each basis offers a simple calculation of one of the contributions. This leads to the following efficient strategy to numerically solve the Schrödinger Equation.

First we focus on the straightforward part of the x-space Schrödinger equation:

$$i\hbar\frac{\delta\psi}{\delta t} = V(x)\psi$$

For time step Δt, the solution is of the form

$$\psi(x, t + \Delta t) = \psi(x, t)e^{-iV(x)\frac{\Delta t}{\hbar}}$$

Next, we solve the straightforward part of the k-space Schrödinger equation:

$$i\hbar\frac{\delta\psi'}{\delta t} = \frac{\hbar^2k^2}{2M}\prime\psi$$

For time step Δt, the solution is of the form

$$\psi'(k, t + \Delta t) = \psi(k, t)e^{-ihk^2\frac{\Delta t}{2M}}$$

3.3.1 Numerical Fourier Transform

A numerical solution will necessitate repeated computations of the Fourier transform of $\psi(x, t)$ and the inverse Fourier transform of $\psi'(k, t)$. The Fast Fourier Transform (FFT) is a class of optimal algorithms for the computation of numerical Fourier transform as it reduces the complexity of computing the Discrete Fourier Transform (DFT) from $O(N^2)$ to $O(N \log N)$, where N is the data size[6].

$$F'_m = \sum_{n=0}^{N-1} F_n \exp[-\frac{2\pi inm}{N}]$$

and its inverse

$$F_n = \frac{1}{N}\sum_{n=0}^{N-1} F'_m \exp[\frac{2\pi inm}{N}]$$

To see the association with the continuous Fourier transforms defined above, assume that the infinite forward transform integral is well-approximated by the finite integral from a to b:

$$\psi'(k,t) = \frac{1}{\sqrt{2\pi}} \int_a^b \psi(x,t) e^{-ikx} dx$$

This is equivalent to the constraint that the potential $V(x) \longrightarrow \infty$ at $x \leq a$ and $x \geq b$. Let us approximate this integral as a Riemann sum of N terms, and define $\Delta x = \frac{(b-a)}{N}$, and $x_n = a + n\Delta x$:

$$\psi'(k,t) \simeq \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{N-1} \psi(x_n,t) e^{-ikx_n} \Delta x$$

Let us now define $k_m = k_0 + m\Delta k$, with $\Delta k = \frac{2\pi}{N\Delta x}$. Our approximation thus becomes

$$\psi'(k_m,t) \simeq \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{N-1} \psi(x_n,t) e^{-ik_m x_n} \Delta x$$

By limiting the range of k through limiting the range of x above, the high-frequency components of the signal get lost in our approximation. The Nyquist sampling theorem[7] says this is an inevitable consequence of choosing discrete steps in space, and the spacing chosen above exactly satisfies the Nyquist limit[8] if $k_0 = -\frac{\pi}{\Delta x}$

Plugging our expressions for x_n and k_m into the Fourier approximation:

$$\psi'(k_m,t) e^{imx_0\Delta k} \simeq \sum_{n=0}^{N-1} [\frac{\Delta x}{\sqrt{2\pi}} \psi(x_n,t) e^{-ik_0 x_n}] \exp(-\frac{2\pi imn}{N})$$

Similar arguments for the inverse Fourier transform yield:

$$\frac{\Delta x}{\sqrt{2\pi}} \psi(k_m,t) e^{-ik_0 x_n} \simeq \frac{1}{N} \sum_{m=0}^{N-1} [\psi(k_m,t) e^{-imx_0\Delta k}] \exp(\frac{2\pi imn}{N})$$

Comparing to discrete Fourier transforms, we find that the continuous Fourier pair

$$\psi(x,t) \iff \psi'(k,t)$$

corresponds to the discrete Fourier pair

$$\frac{\Delta x}{\sqrt{2\pi}} \psi(x_n,t) e^{-ik_0 x_n} \iff \psi'(k_m,t) e^{-imx_0\Delta k}$$

subject to the approximations mentioned above.

3.3.2 Algorithm for Time Propagation

With the above considerations, the following algorithm is implemented:

1. Choose a, b, N and k_0 to adequately represent the initial wavefunction $\psi(x)$. Once chosen, then $\Delta x = \frac{(b-a)}{N}$ and $\Delta k = \frac{2\pi}{(b-a)}$. Define $x_n = a + n\Delta x$ and $k_m = k_0 + m\Delta k$.

2. Discretize the wavefunctions. Let $\psi_n(t) = \psi(x_n, t)$, $V_n = V(x_n)$, and $\psi'_m = \psi'(k_m, t)$.

3. To time evolve a single step Δt:

   - Compute a half-step in x: $\psi_n \longrightarrow \psi_n \exp[-i\frac{V_n \Delta t}{2\hbar}]$
   - Calculate ψ'_m from ψ_n using FFT.
   - Compute a full-step in k: $\psi'_m \longrightarrow \psi'_m \exp[-i\hbar(k \cdot k)\frac{\Delta t}{2M}]$
   - Calculate ψ_n from ψ'_m using inverse FFT.
   - Compute a second half-step in x: $\psi_n \longrightarrow \psi_n \exp[-i\frac{V_n \Delta t}{2\hbar}]$
   - Repeat until the desired time is reached.

The x-space time-step has been split into two half-steps since this leads to a comparatively stabler numerical solution. This is known as the leap-frog integration technique[9].

The implementation detailed above is known as the Strang[10] splitting method $(\frac{1}{2}, 1, \frac{1}{2})$. However, the implementation in the code differs slightly despite being theoretically equivalent. We start with the first third of a Strang step, then switch to the Lie[11]-Trotter[12] method $(1, 1)$ and finish with the last third of a Strang step.

3.4 Uncertainty Measures

In the one-dimensional case, the position-space (S_x) and momentum-space (S_p) Shannon information entropies[13] are defined respectively as:

$$S_x = -\int_{-\infty}^{\infty} |\psi(x)|^2 \log |\psi(x)|^2 dx \tag{3.7}$$

$$S_k = -\int_{-\infty}^{\infty} |\psi'(k)|^2 \log |\psi'(k)|^2 dk \tag{3.8}$$

Similarly, the position-space (S_x) and momentum-space (S_p) standard deviations[14] are defined respectively as:

$$\sigma_x = \int_{-\infty}^{\infty} x^2 |\psi(x)|^2 - (\int_{-\infty}^{\infty} x |\psi(x)|^2)^2 dx \tag{3.9}$$

$$\sigma_p = \int_{-\infty}^{\infty} x^2 |\psi(p)|^2 - (\int_{-\infty}^{\infty} x |\psi(p)|^2)^2 dp \tag{3.10}$$

The position integrals need to be conducted piece-wise as the analytical form of $\psi(x)$ varies widely from one region to another.

Both the above measures are computed using numerical quadratures.

References

1. CODATA Value: natural unit of energy (accessed Apr 17, 2021).
2. CODATA Value: natural unit of mass (accessed Apr 17, 2021).
3. CODATA Value: natural unit of length (accessed Apr 17, 2021).
4. CODATA Value: natural unit of time (accessed Apr 17, 2021).
5. Plancherel, M.; Leffler, M. Contribution Á L'Étude de la reprÉsentation D'une fonction arbitraire par des intÉgrales dÉfinies. *Rend. Circ. Mat. Palermo (2)* **1910**, *30* (1), 289–335.
6. Cooley, J. W.; Tukey, J. W. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comput.* **1965**, *19* (90), 297.

7. Whittaker, E. T. XVIII.—On the Functions Which Are Represented by the Expansions of the Interpolation-Theory. *Proc. R. Soc. Edinb.* **1915**, *35*, 181–194.

8. Brice, R. Digital Signal Processing. *In Newnes Guide to Digital TV*; Elsevier, **2003**; pp 85–111.

9. Birdsall, C. K.; Langdon, A. B. *Plasma Physics via Computer Simulation*; CRC Press, **2018**; pp 56.

10. Strang, G. On the Construction and Comparison of Difference Schemes. *SIAM J. Numer. Anal.* **1968**, *5* (3), 506–517.

11. Lie, S. Theorie der Transformationsgruppen I. *Math. Ann.* **1880**, *16* (4), 441–528.

12. Trotter, H. F. On the Product of Semi-Groups of Operators. *Proc. Am. Math. Soc.* **1959**, *10* (4), 545–545.

13. Cover, T. M.; Thomas, J. A. *Elements of Information Theory*; Wiley, **2005**.

14. Pearson, K. III. Contributions to the Mathematical Theory of Evolution. *Philos. Trans. R. Soc. Lond. A* **1894**, *185* (0), 71–110.

4

Results and Discussion

The full code in *Python* 2.7 used for these results is provided in the appendix from **Listing** 4.2 onwards.

4.1 Finite Square Well

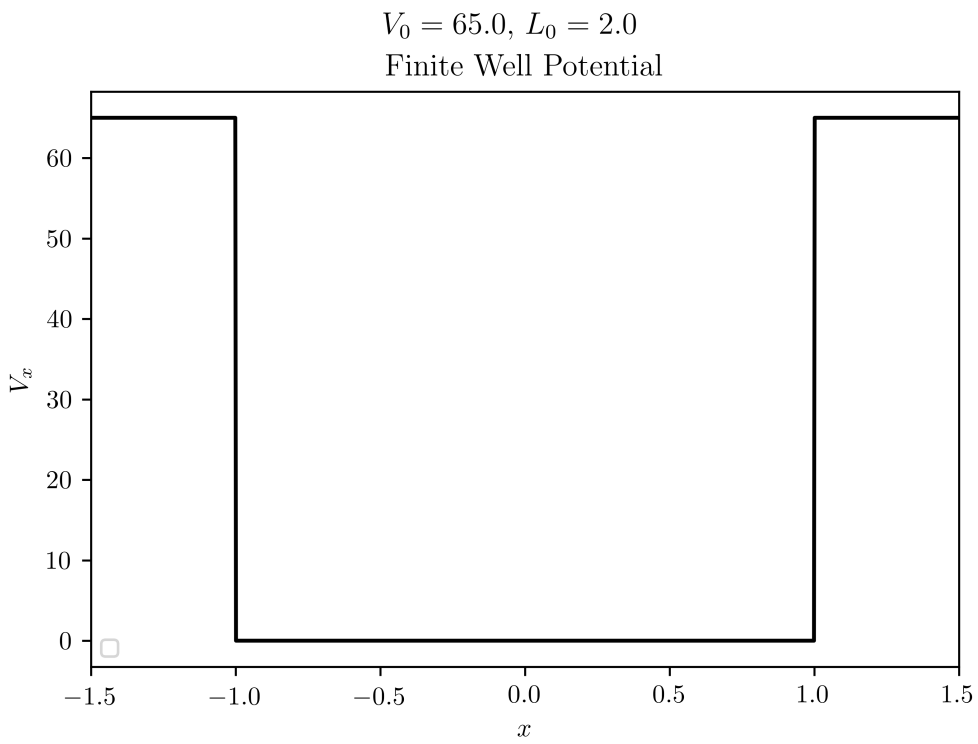


Fig. 4.1 – Potential for a Finite Well for $L = 2.0, U = 65.0$.

$$U(x) = \begin{cases} 0 & x \in (-\frac{L}{2}, \frac{L}{2}) \\ U & \text{elsewhere} \end{cases}$$

Figure 4.1 shows the potential for a finite well with $L = 2.0$ and $U = 65.0$.

Let us divide the system into three regions:

1. $x \leq -\frac{L}{2}$: Left of the well
2. $|x| < \frac{L}{2}$: Inside the well
3. $\frac{L}{2} \leq x$: Right of the well

4.1.1 Stationary Wavefunctions

Inside the box, i.e., in Region 2, the potential is zero, hence the particle is free.

Therefore,

$$E\psi_2(x) = -\frac{\hbar^2}{2M}\frac{d^2\psi(x)}{dx^2}$$

$$\Rightarrow \psi_2(x) = A\sin kx + B\cos kx \tag{4.1}$$

where

$$k = \frac{\sqrt{2ME}}{\hbar}$$

Regions 1 and 3 have similar formulations.

Let

$$\alpha = \frac{\sqrt{2M(U-E)}}{\hbar}$$

Then,

$$\frac{d^2\psi_{1,3}}{dx^2} = \alpha^2\psi_{1,3}$$

$$\Rightarrow \psi_1(x) = C_1e^{\alpha x} + D_1e^{-\alpha x} \tag{4.2}$$

and

$$\psi_3(x) = C_2e^{\alpha x} + D_2e^{-\alpha x} \tag{4.3}$$

The boundary conditions are:

- ψ is normalized and cannot diverge.

  But,

  $$D_1 \neq 0 \Rightarrow D_1e^{-\alpha x} \longrightarrow \infty$$

as $x \longrightarrow -\infty$. Therefore,

$$D_1 = 0 \Rightarrow \psi_1(x) = C_1 e^{\alpha x} \longrightarrow 0$$

as $x \longrightarrow -\infty$.

Similarly,

$$C_2 \neq 0 \Rightarrow C_2 e^{\alpha x} \longrightarrow \infty$$

as $x \longrightarrow \infty$. Therefore,

$$C_2 = 0 \Rightarrow \psi_3(x) = D_2 e^{-\alpha x} \longrightarrow 0$$

as $x \longrightarrow \infty$.

- ψ is continuous and differentiable everywhere, including at the overlap $\{-\frac{L}{2}, \frac{L}{2}\}$.

  Therefore,

$$\psi_1(-\frac{L}{2}) = C_1 e^{-\frac{\alpha L}{2}} = \psi_2(-\frac{L}{2}) = -A \sin \frac{kL}{2} + B \cos \frac{kL}{2}$$

  and

$$\psi_3(\frac{L}{2}) = D_2 e^{-\frac{\alpha L}{2}} = \psi_2(\frac{L}{2}) = A \sin \frac{kL}{2}) + B \cos \frac{kL}{2}$$

  Also,

$$\psi_1'(-\frac{L}{2}) = \alpha C_1 e^{-\frac{\alpha L}{2}} = \psi_2'(-\frac{L}{2}) = kA \cos \frac{kL}{2} + kB \sin \frac{kL}{2}$$

  and

$$\psi_3'(\frac{L}{2}) = -\alpha D_2 e^{-\frac{\alpha L}{2}} = \psi_2'(\frac{L}{2}) = kA \cos \frac{kL}{2} - kB \sin \frac{kL}{2}$$

Since the potential is symmetric i.e., $U(x) = U(-x)$, these equations have two sorts of solutions:

Symmetric:

$$\psi(x) = \psi(-x)$$

This is satisfied by: $A = 0$ and $C_1 = D_2$

$$\Rightarrow C_1 e^{-\frac{\alpha L}{2}} = B \cos \frac{kL}{2}$$

and

$$-\alpha C_1 e^{-\frac{\alpha L}{2}} = -kB \sin \frac{kL}{2}$$

$$\Rightarrow \alpha = k \tan \frac{kL}{2} \tag{4.4}$$

Since $\psi(x)$ is normalized,

$$\int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1$$

Since

$$|\psi(x)|^2 = |\psi(-x)|^2$$

$$\Rightarrow 2\int_{0}^{\infty} |\psi(x)|^2 dx = 1$$

$$\Rightarrow 2[|B|^2 \int_{0}^{\frac{L}{2}} \cos^2 kx dx + |C_1|^2 \int_{\frac{L}{2}}^{\infty} e^{-2\alpha x} dx] = 1$$

$$\Rightarrow 2[|B|^2 (\frac{L}{4} + \frac{\sin kL}{4k}) + |C_1|^2 \frac{e^{-\alpha L}}{2\alpha}] = 1$$

But,

$$C_1 = Be^{\frac{\alpha L}{2}} \cos \frac{kL}{2}$$

$$\Rightarrow |B|^2 [\frac{L}{2} + \frac{\sin kL}{2k} + \frac{\cos^2 \frac{kL}{2}}{\alpha}] = 1$$

Now,

$$\alpha = k \tan \frac{kL}{2}$$

$$\Rightarrow |B|^2 [\frac{L}{2} + \frac{\sin \frac{kL}{2} \cos \frac{kL}{2}}{k} + \frac{\cos^3 \frac{kL}{2}}{k \sin \frac{kL}{2}}] = 1$$

$$\Rightarrow |B|^2 [\frac{L}{2} + \frac{\cos \frac{kL}{2}}{k \sin \frac{kL}{2}}] = 1$$

$$\Rightarrow |B|^2 [\frac{L}{2} + \frac{1}{k \tan \frac{kL}{2}}] = 1$$

$$\Rightarrow |B|^2 [\frac{L}{2} + \frac{1}{\alpha}] = 1$$

$V_0 = 65.0,\ L_0 = 2.0,\ m = 0.5,\ n = [1, 3, 5]$

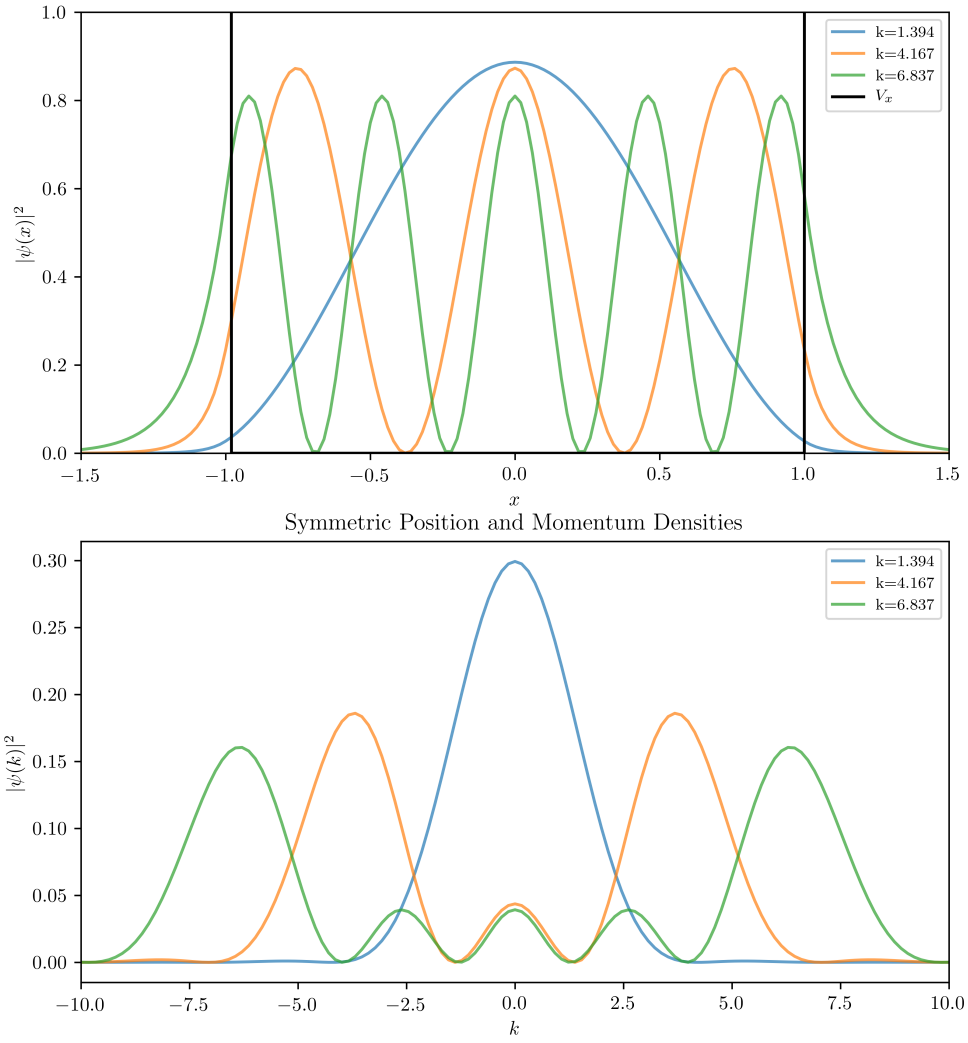


Fig. 4.2 – Symmetric Position and Momentum Densities of the Finite Well. $M = 0.5, L = 2, U = 65$ are chosen.

$$\Rightarrow B = \frac{1}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}}$$

and

$$C_1 = \frac{e^{\frac{\alpha L}{2}} \cos \frac{kL}{2}}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}}$$

Therefore,

$$\psi_1(x) = \frac{e^{\frac{\alpha L}{2}} \cos \frac{kL}{2} e^{\alpha x}}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}} \tag{4.5}$$

$$\psi_2(x) = \frac{\cos kx}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}} = \psi_2(-x) \tag{4.6}$$

$$\psi_3(x) = \frac{e^{\frac{\alpha L}{2}} \cos \frac{kL}{2} e^{-\alpha x}}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}} = \psi_1(-x) \tag{4.7}$$

Despite the analytical normalization here, a numerical method is implemented instead owing to better precision.

Figure 4.2 shows the symmetric position (a) and momentum (b) densities of the finite well. $M = 0.5, L = 2, U = 65$ are chosen. All the even parity states are plotted. For both position and momentum, the n^{th} state has n peaks and $(n + 1)$ nodes. Higher n values have higher spread towards the extrema of the box. 0 is a common peak.
(a) The peaks are all equal.
(b) The peaks are symmetric about 0 but of different heights.

Anti-symmetric:

$$\psi(x) = -\psi(-x)$$

This is satisfied by: $B = 0$ and $C_1 = -D_2$.

$$\Rightarrow C_1 e^{-\frac{\alpha L}{2}} = -A \sin \frac{kL}{2}$$

and

$$\alpha C_1 e^{-\frac{\alpha L}{2}} = kA \cos \frac{kL}{2}$$

$$\Rightarrow \alpha = -k \cot \frac{kL}{2} \tag{4.8}$$

Again, since $\psi(x)$ is normalized,

$$\int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1$$

Since

$$|\psi(x)|^2 = |\psi(-x)|^2$$

$$\Rightarrow 2\int_{0}^{\infty} |\psi(x)|^2 dx = 1$$

$$\Rightarrow 2[|A|^2 \int_{0}^{\frac{L}{2}} \sin^2 kx dx + |C_1|^2 \int_{\frac{L}{2}}^{\infty} e^{-2\alpha x} dx] = 1$$

$$\Rightarrow 2[|A|^2 (\frac{L}{4} - \frac{\sin kL}{4k}) + |C_1|^2 \frac{e^{-\alpha L}}{2\alpha}] = 1$$

But,

$$C_1 = -Ae^{\frac{\alpha L}{2}} \sin \frac{kL}{2}$$

$$\Rightarrow |A|^2 [\frac{L}{2} - \frac{\sin kL}{2k} + \frac{\sin^2 \frac{kL}{2}}{\alpha}] = 1$$

Now,

$$\alpha = -k \cot \frac{kL}{2}$$

$$\Rightarrow |A|^2 [\frac{L}{2} - \frac{\sin \frac{kL}{2} \cos \frac{kL}{2}}{k} - \frac{\sin^3 \frac{kL}{2}}{k \cos \frac{kL}{2}}] = 1$$

$$\Rightarrow |A|^2 [\frac{L}{2} - \frac{\sin \frac{kL}{2}}{k \cos \frac{kL}{2}}] = 1$$

$$\Rightarrow |A|^2 [\frac{L}{2} - \frac{1}{k \cot \frac{kL}{2}}] = 1$$

$$\Rightarrow |A|^2 [\frac{L}{2} + \frac{1}{\alpha}] = 1$$

$$\Rightarrow A = \frac{1}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}}$$

$V_0 = 65.0,\ L_0 = 2.0,\ m = 0.5,\ n = [2, 4, 6]$

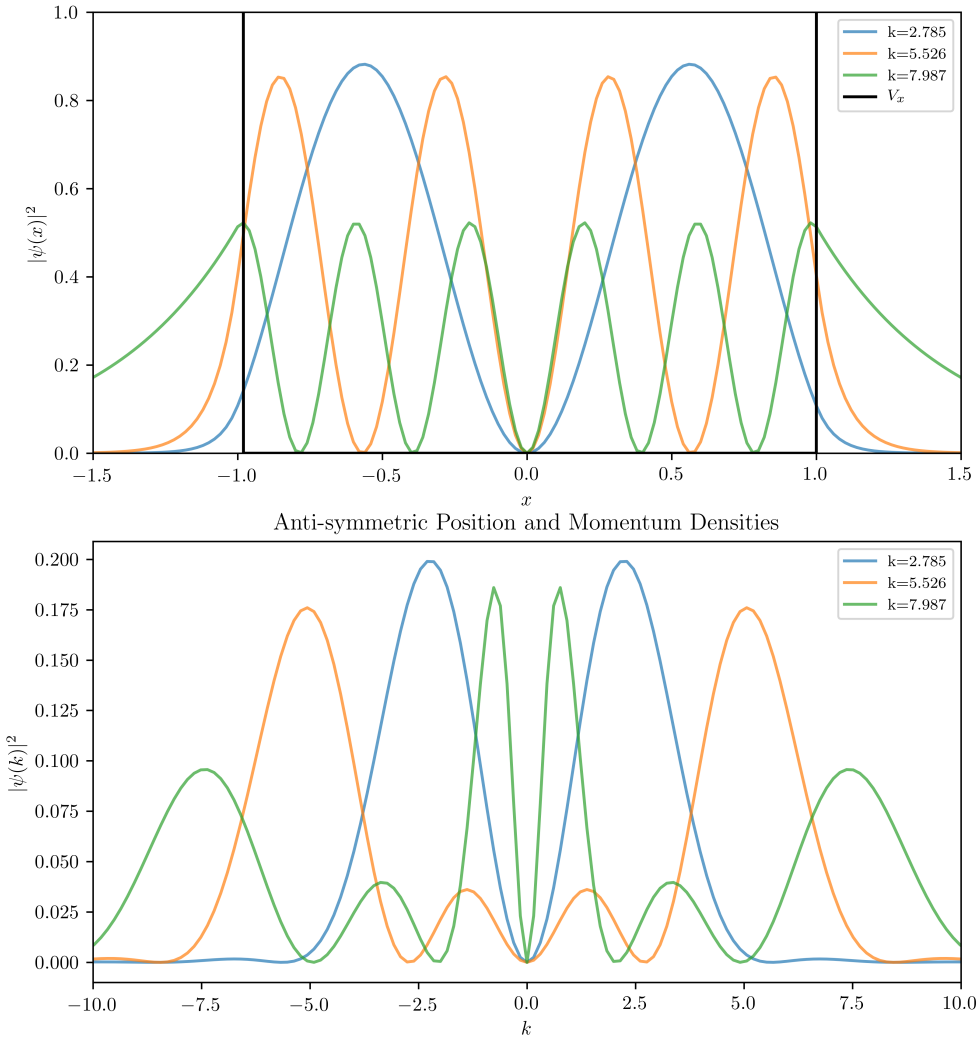


Fig. 4.3 – Anti-symmetric Position and Momentum Densities of the Finite Well. $M = 0.5, L = 2, U = 75$ are chosen.

and

$$C_1 = \frac{-e^{\frac{\alpha L}{2}} \sin \frac{kL}{2}}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}}$$

Therefore,

$$\psi_1(x) = -\frac{e^{\frac{\alpha L}{2}} \sin \frac{kL}{2} e^{\alpha x}}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}} \tag{4.9}$$

$$\psi_2(x) = \frac{\sin kx}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}} = -\psi_2(-x) \tag{4.10}$$

$$\psi_3(x) = \frac{e^{\frac{\alpha L}{2}} \sin \frac{kL}{2} e^{-\alpha x}}{\sqrt{\frac{L}{2} + \frac{1}{\alpha}}} = -\psi_1(-x) \tag{4.11}$$

Despite the analytical normalization here, a numerical method is implemented instead for better precision.

Figure 4.3 shows the anti-symmetric position (a) and momentum (b) densities of the finite well. $M = 0.5, L = 2, U = 75$ are chosen. All the odd parity states are plotted. For both position and momentum, the n^{th} state has n peaks and $(n + 1)$ nodes. Higher n values have higher spread towards the extrema of the box. 0 is a common node.
(a) The peaks are all equal.
(b) The peaks are symmetric about 0 but of different heights.

4.1.2 Bound States

Both α and k depend on E. Thus, only certain energy values, solutions to one or both of these two equations, are allowed.

We have

$$\alpha^2 + k^2 = \frac{2MU}{\hbar^2} \tag{4.12}$$

. Also, the symmetry and anti-symmetry equations read

$$\alpha = k \tan \frac{kL}{2}$$

and

$$\alpha = -k \cot \frac{kL}{2}$$

respectively.

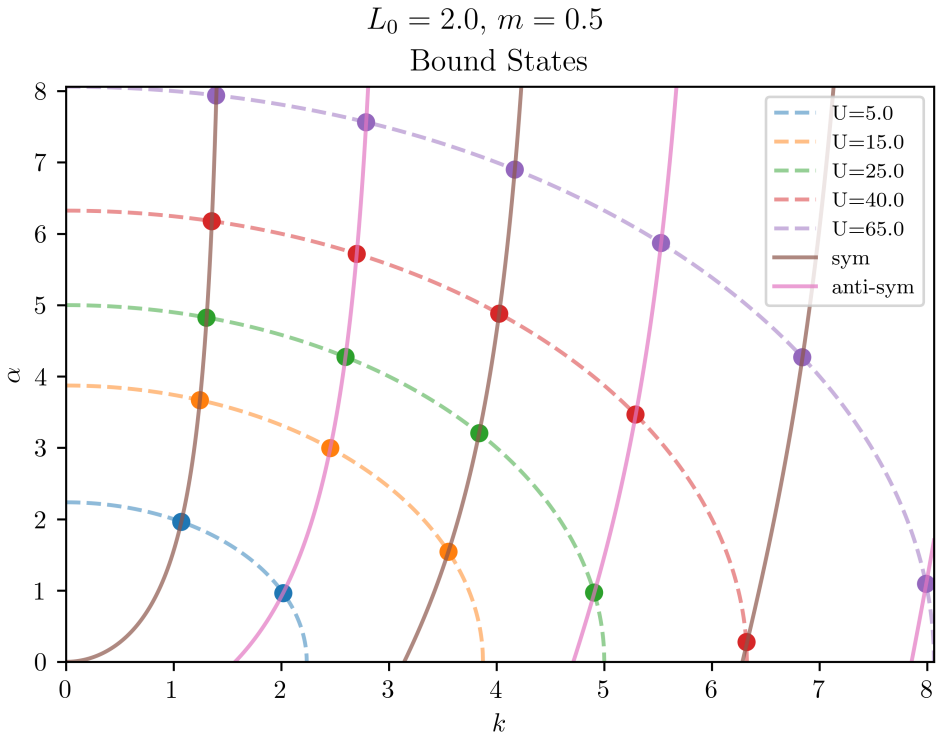


Fig. 4.4 – Bound States of the Finite Well. $M = 0.5, L = 2, U = [5, 15, 25, 40, 65]$ are chosen.

Solutions are represented by the intersection points of the first quadrant quarter-circle of radius $\sqrt{2MU}$ with the curves $k \tan \frac{kL}{2}$ and $-k \cot \frac{kL}{2}$. Each such curve represents a possible solution, k_i within the range $\frac{\pi(i-1)}{L} \leq v_i \leq \frac{\pi i}{L}$.

Thus, the total number of solutions N is determined by dividing the radius of the quarter-circle $\sqrt{2MU}$ by the range of each solution $\frac{\pi}{L}$ and using the ceiling function, i.e., $N = \left\lceil \frac{L\sqrt{2MU}}{\pi} \right\rceil$.

However small U is, there is always at least one intersection i.e., bound state. As $U \longrightarrow \infty$, the radius of the quarter-circle gets larger and the roots get closer to the roots of the infinite square well.

Figure 4.4 shows the bound states of the finite well. $M = 0.5, L = 2, U = [5, 15, 25, 40, 65]$ are chosen. The solution tuples (k, α) are formed by the descent of the curves $k \tan \frac{kL}{2}$ and $-k \cot \frac{kL}{2}$ on the first quadrant quarter-circle of radius $\sqrt{2MU}$, respectively giving the even parity and odd parity roots.

4.1.3 Uncertainty Measures

Figure 4.5 shows a standard deviation barchart of bound states of the finite well. $M = 0.5, L = 2, U = 65$ are chosen. σ_x shows a minor increasing trend while σ_p agrees with the above trend of higher symmetric states. The product is again dominated by σ_p. $n = 2$ is very close to the Heisenberg product lower bound.

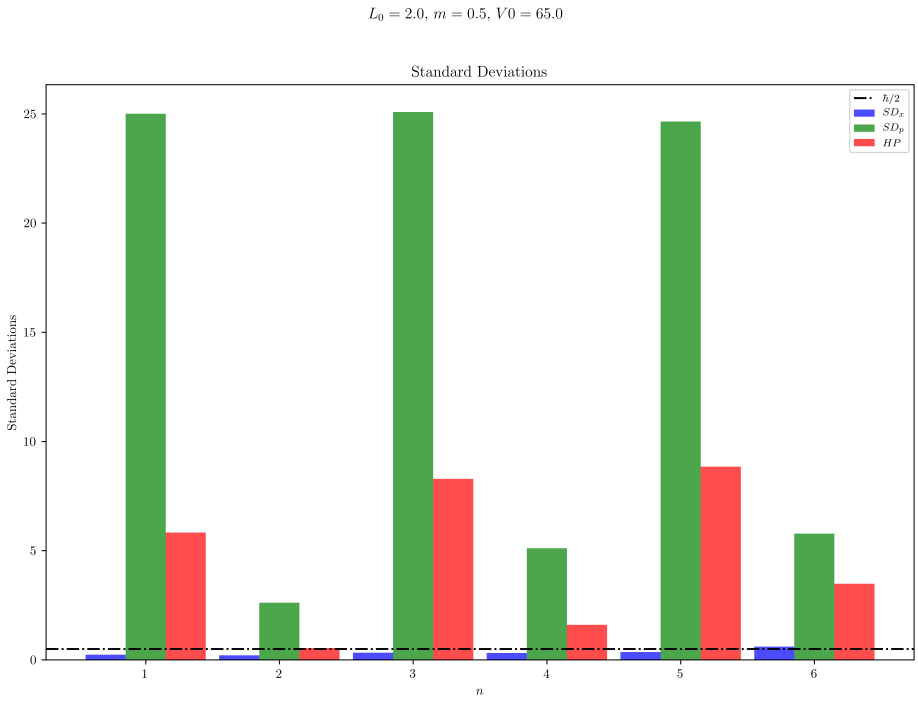


Fig. 4.5 – Standard Deviation Barchart of Bound States of the Finite Well. $M = 0.5, L = 2, U = 65$ are chosen.

Figure 4.6 shows a Shannon entropy barchart of bound states of the finite well. $M = 0.5, L = 2, U = 65$ are chosen. This is again similar to the standard deviation barchart. S_x shows an increasing trend while S_k shows higher symmetric states. The sum is thus dominated by S_k. $n = 2$ is very close to the entropy sum lower bound.

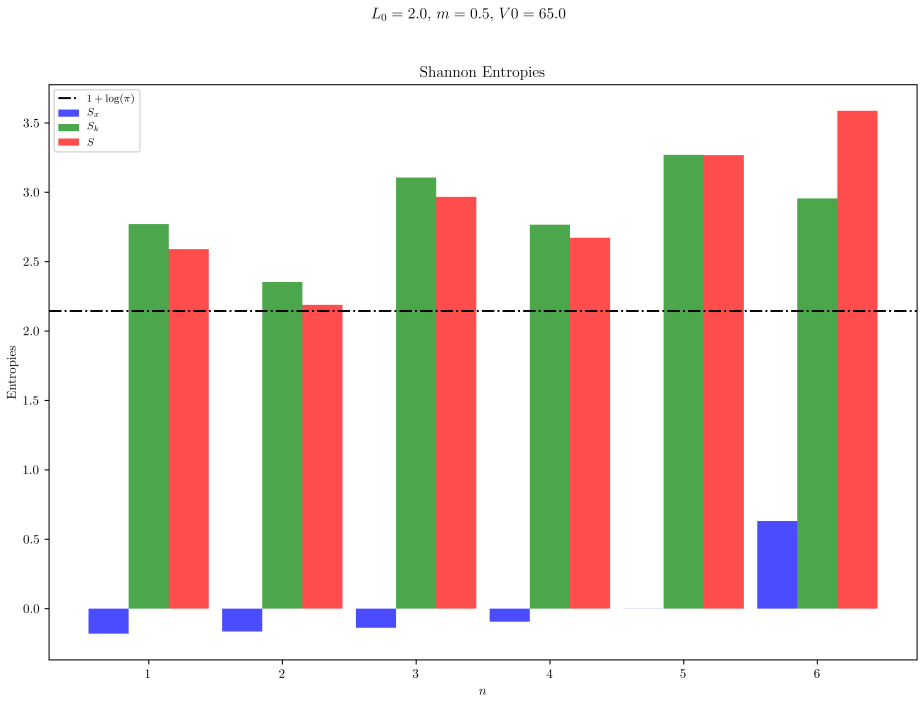


Fig. 4.6 – Shannon Entropy Barchart of Bound States of the Finite Well. $M = 0.5, L = 2, U = 65$ are chosen.

Figure 4.7 shows the position (a) and momentum (b) standard deviations and the Heisenberg product (c) of bound states of the finite well. $M = 0.5, L = 2, U = [5, 15, 25, 40, 65]$ are chosen.
(a) The states show little variation for different U but show a slightly increasing trend in n.
(b) The symmetric states are consistently higher than their anti-symmetric counterparts. Within themselves, the former show little variation while the latter show a slightly increasing trend in n.
(c) Dominated by the σ_p trend, symmetric states are consistently higher than anti-symmetric. Both groups slightly increase in n. $n = 2$ for all U are very close to the Heisenberg product lower bound.

Figure 4.8 show the position (a) and momentum (b) Shannon entropies and the entropy sum (c) of bound states of the finite well. $M = 0.5, L = 2, U = [5, 15, 25, 40, 65]$ are chosen. These plots are very similar to their standard deviation counterparts.
(a) The states show little variation for different U but show a slightly increasing trend in n.
(b) The symmetric states are consistently higher than their anti-symmetric counterparts. Within themselves, both groups show a slightly increasing trend in n.
(c) Dominated by the S_k trend, symmetric states are consistently higher than anti-symmetric. Both groups slightly increase in n. $n = 2$ for all U are very close to the entropy sum lower bound.

$L_0 = 2.0,\ m = 0.5$

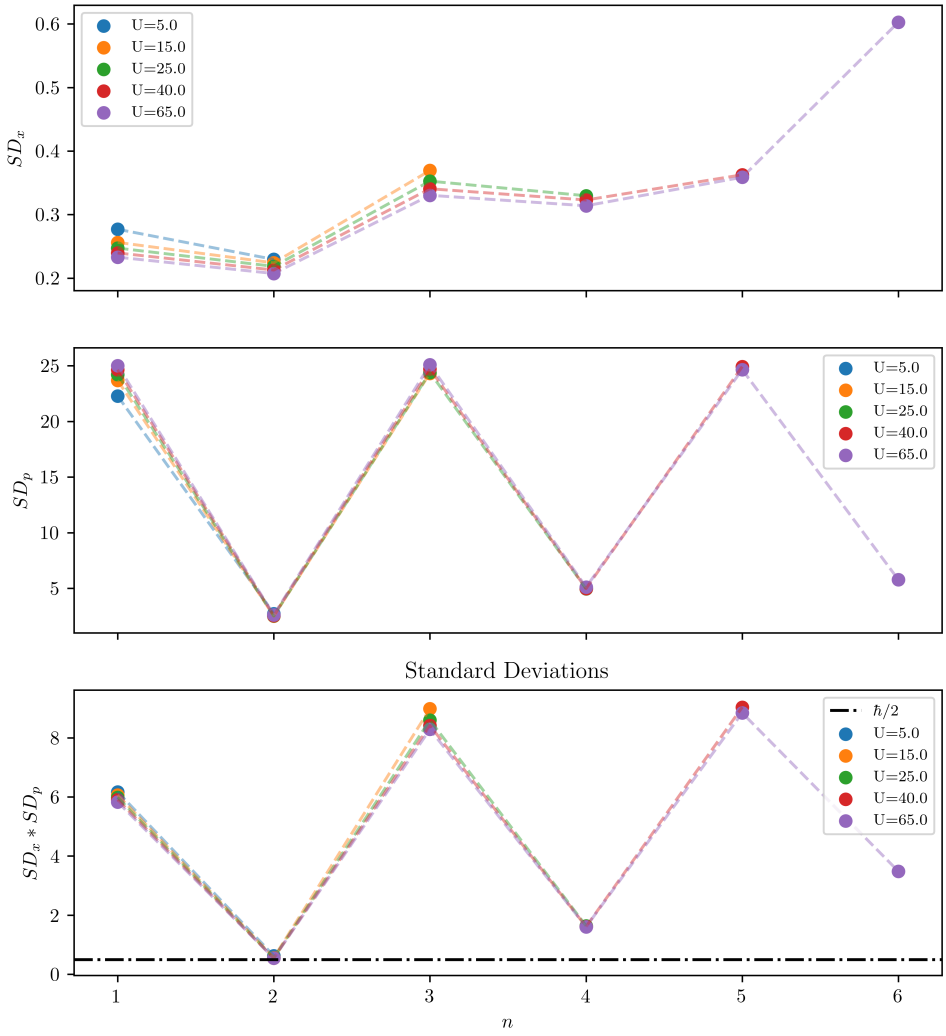


Fig. 4.7 – Standard Deviations of Bound States of the Finite Well. $M = 0.5, L = 2, U = [5, 15, 25, 40, 65]$ are chosen.

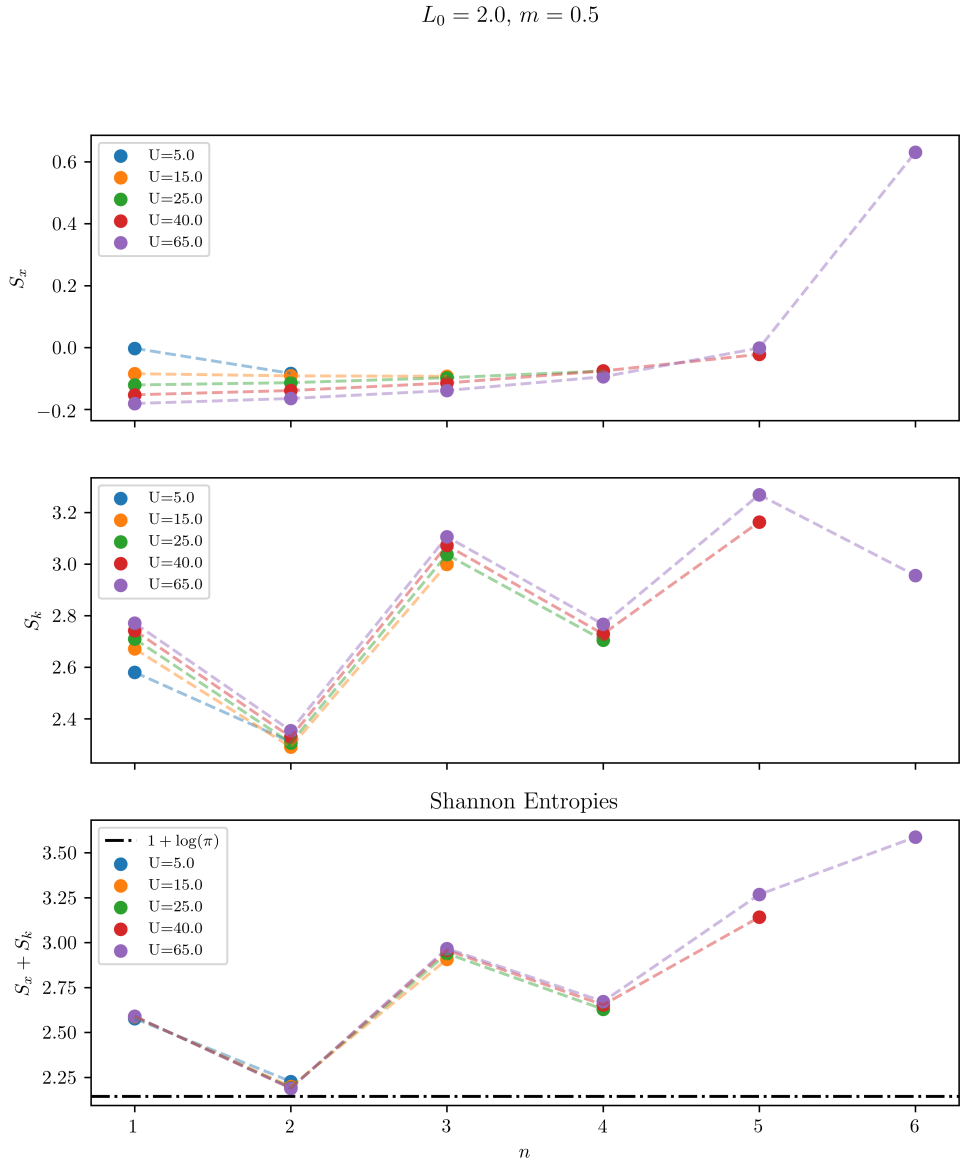


Fig. 4.8 – Shannon Entropies of Bound States of the Finite Well. $M = 0.5, L = 2, U = [5, 15, 25, 40, 65]$ are chosen.

4.1.4 Error Analysis

Mass of the particle $M = 0.5$
Length of the well $L = 2.0$
Height of the walls of the well $U = 65.0$

A total of N samples is taken with spacing Δx for the range of integration in x which is symmetric about 0. For k, again N samples are taken symmetrically about 0 but with spacing $\Delta p = \frac{2\pi}{N\Delta x}$

Table 4.1 shows the error analysis of stationary wavefunctions for the finite well. $N = 2^{11}, \Delta x = 0.02$ are chosen.

$\psi(x)$ is numerically normalized, despite the analytical normalization and the Parseval-Plancherel theorem ensures $\psi'(k)$ is consequently normalized.

Tab. 4.1 – Error Analysis of Stationary Wavefunctions for the Finite Well

n	$\int \lvert\psi(x)\rvert^2 dx$	$\int \lvert\psi'(k)\rvert^2 dk$
1	0.9999999999999999	0.9999999999998336
2	1.0	0.9999999999999878
3	1.0	0.9999999999999466
4	1.0	0.9999999999999892
5	1.0	0.9999999999997083
6	0.9999999999999999	0.9999999999999887

4.2 Finite Rectangular Barrier

$$U(x) = \begin{cases} U & x \in [0, L] \\ 0 & \text{elsewhere} \end{cases}$$

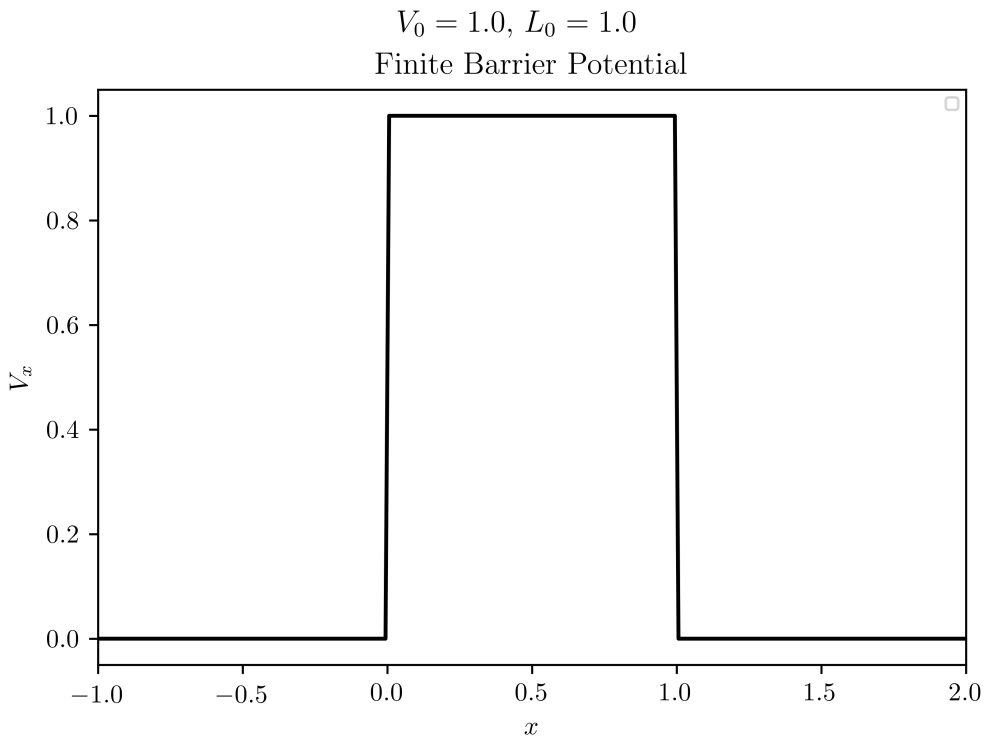


Fig. 4.9 – Potential for a Finite Rectangular Barrier for $L = 4.0, U = 1.0$.

Figure 4.9 shows the potential for a finite rectangular barrier with $L = 4.0, U = 1.0$.

Let us divide the system into three regions:

1. $x < 0$: Left of the barrier
2. $0 \leq x \leq L$: Inside the barrier
3. $L < x$: Right of the barrier

4.2.1 Stationary Wavefunctions

Outside the box, i.e., in Regions 1 and 3, the potential is zero, hence the particle is free.

Therefore,

$$E\psi_1(x) = E\psi_3(x) = -\frac{\hbar^2}{2M}\frac{d^2\psi_{1,3}(x)}{dx^2}$$

$$\Rightarrow \frac{d^2\psi_{1,3}(x)}{dx^2} + k^2\psi_{1,3} = 0$$

where

$$k = \frac{\sqrt{2ME}}{\hbar}$$

$$\Rightarrow \psi_1(x) = Ae^{ikx} + Be^{-ikx} \tag{4.13}$$

$$\psi_3(x) = Fe^{ikx} + Ge^{-ikx} \tag{4.14}$$

It is worth noting that Ae^{ikx}, Fe^{ikx} represent waves travelling in the positive direction while Be^{-ikx}, Ge^{-ikx} represent waves travelling in the negative direction. Due to reflection by the barrier, it is possible to have wave components travelling in the negative direction for $x < L$, but no such reason for $x > L$. Thus, $G = 0$.

There are three cases to be considered for Region 2.

- **Case I:** $E > U$

  Let

  $$\alpha = \frac{\sqrt{2M(E-U)}}{\hbar}$$

  Then,

  $$\frac{d^2\psi_2}{dx^2} + \alpha^2\psi_2 = 0$$

  $$\Rightarrow \psi_2(x) = Ce^{i\alpha x} + De^{-i\alpha x} \tag{4.15}$$

- **Case II:** $E = U$

  $$E = U \Rightarrow \psi_2''(x) = 0$$

Hence the solutions of the Schrödinger equation are not exponential but linear functions.

$$\Rightarrow \psi_2(x) = C + Dx \tag{4.16}$$

- **Case III:** $E < U$

  $\frac{\sqrt{2M(E-U)}}{\hbar}$ is imaginary.

  Let

  $$\alpha = \frac{\sqrt{2M(U-E)}}{\hbar}$$

  Then,

  $$\frac{d^2\psi_2}{dx^2} - \alpha^2\psi_2 = 0$$

$$\Rightarrow \psi_2(x) = Ce^{\alpha x} + De^{-\alpha x} \tag{4.17}$$

Now, ψ is continuous and differentiable everywhere, including at the overlap $\{0, L\}$.

Therefore,

$$\psi_1(0) = \psi_2(0)$$

and

$$\psi_3(L) = \psi_2(L)$$

Also,

$$\psi_1'(0) = \psi_2'(0)$$

and

$$\psi_3'(L) = \psi_2'(L)$$

Case I: $E > U$

$$A + B = C + D$$

$$ik(A - B) = i\alpha(C - D)$$

$$Ce^{i\alpha L} + De^{-i\alpha L} = Fe^{ikL}$$

$$i\alpha(Ce^{i\alpha L} - De^{-i\alpha L}) = ikFe^{ikL}$$

$$\Rightarrow k(A+B) = k(C+D)$$

$$k(A-B) = \alpha(C-D)$$

$$\alpha(Ce^{i\alpha L} + De^{-i\alpha L}) = \alpha Fe^{ikL}$$

$$\alpha(Ce^{i\alpha L} - De^{-i\alpha L}) = kFe^{ikL}$$

$$\Rightarrow 2kA = (k+\alpha)C + (k-\alpha)D$$

$$2kB = (k-\alpha)C + (k+\alpha)D$$

$$2\alpha Ce^{i\alpha L} = (k+\alpha)Fe^{ikL}$$

$$2\alpha De^{-i\alpha L} = (\alpha-k)Fe^{ikL}$$

$$\Rightarrow 2kA = \frac{(k+\alpha)^2}{2\alpha})Fe^{i(k-\alpha)L} - \frac{(k-\alpha)^2}{2\alpha})Fe^{i(k+\alpha)L}$$

$$\Rightarrow 4k\alpha e^{-ikL}A = (k+\alpha)^2Fe^{-i\alpha L} - (k-\alpha)^2Fe^{i\alpha L}$$

$$\Rightarrow 4k\alpha e^{-ikL}A = (-2ik^2\sin\alpha L + 4k\alpha\cos\alpha L - 2i\alpha^2\sin\alpha L)F$$

$$\Rightarrow A = \frac{F}{2}\frac{e^{ikL}(-i(k^2+\alpha^2)\sin\alpha L + 2k\alpha\cos\alpha L)}{k\alpha}$$

Also,

$$2kB = \frac{(k-\alpha)^2}{2\alpha}Fe^{i(k-\alpha)L} - \frac{(k-\alpha)^2}{2\alpha}Fe^{i(k+\alpha)L}$$

$$\Rightarrow 4k\alpha e^{-ikL}B = (k-\alpha)^2F(e^{-i\alpha L} - e^{i\alpha L})$$

$$\Rightarrow 2k\alpha e^{-ikL}B = -i(k^2-\alpha^2)\sin\alpha L)F$$

$$\Rightarrow B = -i\frac{F}{2}\frac{e^{ikL}(k^2-\alpha^2)\sin\alpha L}{k\alpha}$$

Now,

$$C + D = A + B$$

$$C - D = \frac{k}{\alpha}(A - B)$$

$$\Rightarrow C = \frac{1}{2}(A + B) + \frac{k}{\alpha}(A - B))$$

$$= \frac{F}{2}\frac{e^{ikL}(k\alpha + k^2)\cos\alpha L - i(k\alpha + k^2)\sin\alpha L)}{k\alpha}$$

$$\frac{F}{2}e^{ikL}(\cos\alpha L(1 + \frac{k}{\alpha}) - i\sin\alpha L(1 + \frac{k}{\alpha}))$$

$$\Rightarrow D = \frac{1}{2}(A + B) - \frac{k}{\alpha}(A - B))$$

$$= \frac{F}{2}\frac{e^{ikL}(k\alpha - k^2)\cos\alpha L + i(k\alpha - k^2)\sin\alpha L)}{k\alpha}$$

$$= \frac{F}{2}e^{ikL}(\cos\alpha L(1 - \frac{k}{\alpha}) + i\sin\alpha L(1 - \frac{k}{\alpha}))$$

Normalization will be carried out through numerical means. For now, $F = 2$ is chosen.

Thus,

$$\psi_1(x) = \frac{e^{ikL}(-i(k^2+\alpha^2)\sin\alpha L + 2k\alpha\cos\alpha L)e^{ikx} - ie^{ikL}(k^2-\alpha^2)\sin\alpha L e^{-ikx}}{k\alpha} \quad (4.18)$$

$$\psi_2(x) = e^{ikL}(\cos\alpha L(1+\frac{k}{\alpha}) - i\sin\alpha L(1+\frac{k}{\alpha})e^{i\alpha x} + e^{ikL}(\cos\alpha L(1-\frac{k}{\alpha}) + i\sin\alpha L(1-\frac{k}{\alpha}))e^{-i\alpha x} \quad (4.19)$$

$$\psi_3(x) = 2e^{ikx} \tag{4.20}$$

Case II: $E = U$

$$A + B = C$$

$$ik(A - B) = D$$

$$C + DL = Fe^{ikL}$$

$$D = ikFe^{ikL}$$

$$\Rightarrow (A - B) = \frac{D}{ik} = Fe^{ikL}$$

$$A + B = C = Fe^{ikL} - DL = Fe^{ikL}(1 - ikL)$$

$$\Rightarrow A = \frac{F}{2}e^{ikL}(2 - ikL), B = -\frac{F}{2}ikLe^{ikL}$$

Normalization will be carried out through numerical means. For now, $F = 2$ is chosen.

Thus,

$$\psi_1(x) = e^{ikL}(2 - ikL)e^{ikx} - ikLe^{ikL}e^{-ikx} \tag{4.21}$$

$$\psi_2(x) = 2e^{ikL}(1 - ikL) + 2ike^{ikL}x \tag{4.22}$$

$$\psi_3(x) = 2e^{ikx} \tag{4.23}$$

Case III: $E < U$

$$A + B = C + D$$

$$ik(A - B) = \alpha(C - D)$$

$$Ce^{\alpha L} + De^{-\alpha L} = Fe^{ikL}$$

$$\alpha(Ce^{\alpha L} - De^{-\alpha L}) = ikFe^{ikL}$$

$$\Rightarrow ik(A + B) = ik(C - D)$$

$$ik(A - B) = \alpha(C - D)$$

$$Ce^{\alpha L} + De^{-\alpha L} = Fe^{ikL}$$

$$Ce^{\alpha L} - De^{-\alpha L}) = \frac{ik}{\alpha}Fe^{ikL}$$

$$\Rightarrow 2ikA = (ik + \alpha)C + (ik - \alpha)D$$

$$2ikB = (ik - \alpha)C + (ik + \alpha)D$$

$$2Ce^{\alpha L} = (1 + \frac{ik}{\alpha})Fe^{ikL}$$

$$2De^{-\alpha L} = (1 - \frac{ik}{\alpha})Fe^{ikL}$$

$$\Rightarrow 2ikA = \frac{(ik + \alpha)^2}{2\alpha}Fe^{(ik-\alpha)L} + \frac{(ik - \alpha)^2}{2\alpha}Fe^{(ik+\alpha)L}$$

$$2ikB = -\frac{k^2 + \alpha^2}{2\alpha}Fe^{(ik-\alpha)L} + \frac{k^2 + \alpha^2}{2\alpha}Fe^{(ik+\alpha)L}$$

$$C = \frac{F}{2}e^{ikL}e^{-\alpha L}(1 + \frac{ik}{\alpha})$$

$$D = \frac{F}{2}e^{ikL}e^{\alpha L}(1 - \frac{ik}{\alpha})$$

$$\Rightarrow 4ik\alpha e^{-ikL}A = (ik+\alpha)^2)Fe^{-\alpha L} - (ik-\alpha)^2)Fe^{\alpha L}$$

$$\Rightarrow 4ik\alpha e^{-ikL}A = [(k^2-\alpha^2)(e^{\alpha L} - e^{-\alpha L}) + 2ik\alpha(e^{\alpha L} + e^{-\alpha L})]F$$

$$\Rightarrow A = \frac{F}{2}\frac{e^{ikL}(k^2-\alpha^2)\sinh\alpha L + 2ik\alpha\cosh\alpha L)}{ik\alpha}$$

Also,

$$2ikB = -\frac{k^2+\alpha^2}{2\alpha}Fe^{(ik-\alpha)L} + \frac{k^2+\alpha^2}{2\alpha}Fe^{(ik+\alpha)L}$$

$$\Rightarrow 4ik\alpha e^{-ikL}B = (k^2+\alpha^2)F(e^{\alpha L} - e^{-\alpha L})$$

$$\Rightarrow 2ik\alpha e^{-ikL}B = (k^2+\alpha^2)\sinh\alpha L)F$$

$$\Rightarrow B = \frac{F}{2}\frac{e^{ikL}(k^2+\alpha^2)\sinh\alpha L}{ik\alpha}$$

Normalization will be carried out through numerical means. For now, $F = 2$ is chosen.

Thus,

$$\psi_1(x) = \frac{e^{ikL}(k^2-\alpha^2)\sinh\alpha L + 2ik\alpha\cosh\alpha L)e^{ikx} + e^{ikL}(k^2+\alpha^2)\sinh\alpha L e^{-ikx}}{ik\alpha} \tag{4.24}$$

$$\psi_2(x) = e^{ikL}e^{-\alpha L}(1+\frac{ik}{\alpha})e^{\alpha x} + e^{ikL}e^{\alpha L}(1-\frac{ik}{\alpha})e^{-\alpha x} \tag{4.25}$$

$$\psi_3(x) = 2e^{ikx} \tag{4.26}$$

Figure 4.10 shows the position (a) and momentum (b) densities for the finite barrier. $M = 1, L = 4, U = 1, E = [0.5, 1.0, \frac{(\frac{\pi}{4})^2}{2} + 1, 1.8, \frac{(\frac{\pi}{2})^2}{2} + 1, 3.0, \frac{(\frac{3\pi}{4})^2}{2} + 1]$ are chosen.
(a) While $E = 0.5$ loses almost all its amplitude after decaying in Region 2. $E = \frac{(\frac{\pi}{4})^2}{2} + 1, \frac{(\frac{\pi}{2})^2}{2} + 1, \frac{(\frac{3\pi}{4})^2}{2} + 1$ remain constant and equal in Regions $1, 3$, indicating resonant

$V_0 = 1.0,\ L_0 = 4.0,\ m = 1.0$

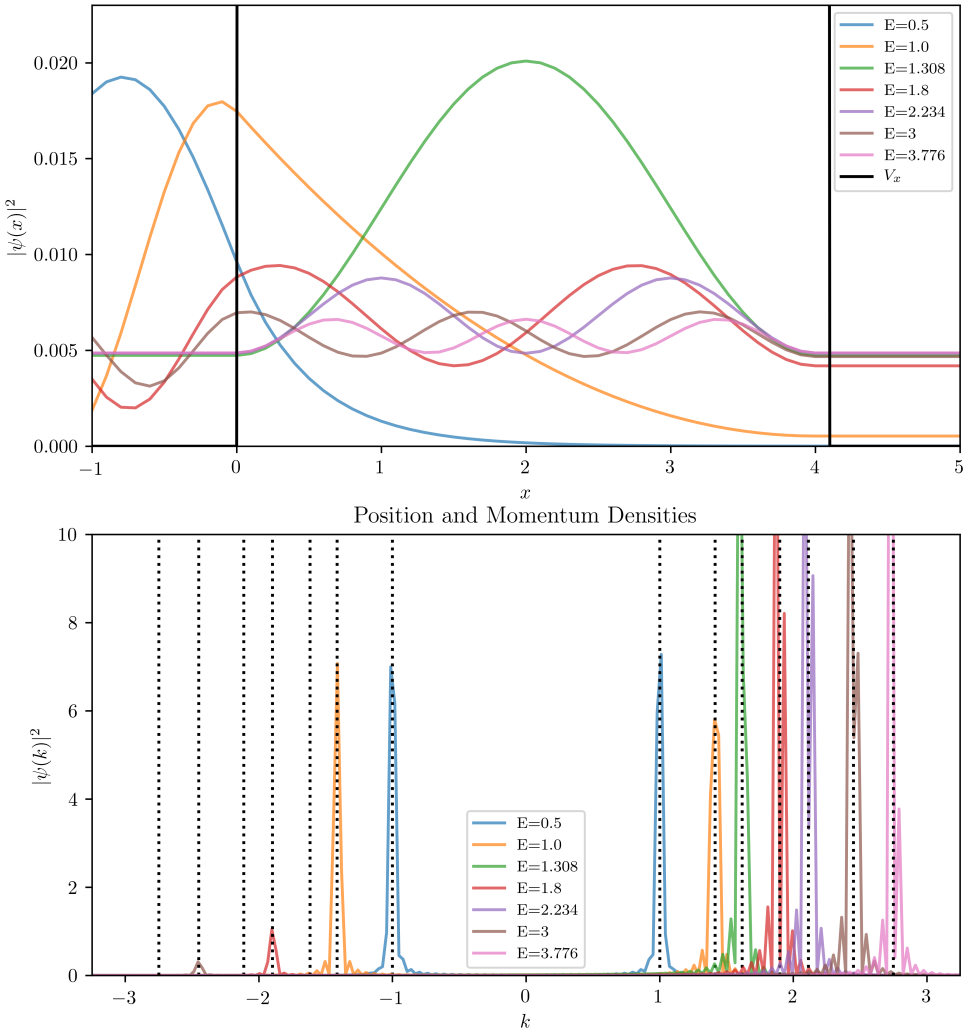


Fig. 4.10 – Position and Momentum Densities for the Finite Barrier. $M = 1, L = 4, U = 1, E = [0.5, 1.0, \frac{(\frac{\pi}{4})^2}{2} + 1, 1.8, \frac{(\frac{\pi}{2})^2}{2} + 1, 3.0, \frac{(\frac{3\pi}{4})^2}{2} + 1]$ are chosen.

transmission. $E = \frac{(\frac{\pi}{4})^2}{2} + 1$ shows a higher Region 3 amplitude than $E = 1.8$ and a single high peak in Region 2, while $E = \frac{(\frac{\pi}{2})^2}{2} + 1$ shows two equal but smaller peaks and $E = \frac{(\frac{3\pi}{4})^2}{2} + 1$ shows three equal but even smaller peaks.
(b) For non-resonant transmission, two peaks (at $k = \pm\sqrt{2ME}$) are observed. For resonances, a single peak is observed at $k = \sqrt{2ME}$.

4.2.2 Resonant States

The incident, reflected and transmitted components of the wavefunction are:

$$\psi_i(x) = Ae^{ikx}$$
$$\psi_r(x) = Be^{-ikx}$$
$$\psi_t(x) = Fe^{ikx}$$

The reflection and transmission coefficients, R and T are defined as:

$$R = -\frac{j_r}{j_i}$$

and

$$T = \frac{j_t}{j_i}$$

where j_i, j_r and j_t are the incident, reflected and transmitted probability currents.

Thus,

$$R = -\frac{\psi_r(x)\frac{d}{dx}(\psi_r{}^*(x)) - \psi_r{}^*(x)\frac{d}{dx}(\psi_r(x))}{\psi_i(x)\frac{d}{dx}(\psi_i{}^*(x)) - \psi_i{}^*(x)\frac{d}{dx}(\psi_i(x))} = \frac{|B|^2}{|A|^2} \tag{4.27}$$

and

$$T = \frac{\psi_t(x)\frac{d}{dx}(\psi_t{}^*(x)) - \psi_t{}^*(x)\frac{d}{dx}(\psi_t(x))}{\psi_i(x)\frac{d}{dx}(\psi_i{}^*(x)) - \psi_i{}^*(x)\frac{d}{dx}(\psi_i(x))} = \frac{|F|^2}{|A|^2} \tag{4.28}$$

Note that $R + T = 1$. Resonant transmission is the phenomena of $T = 1$, i.e., $R = 0$.

Case I: $E > U$

$$R = \frac{(k^2 - \alpha^2)^2 \sin^2 \alpha L}{4k^2\alpha^2 \cos^2 \alpha L + (k^2 + \alpha^2)^2 \sin^2 \alpha L}$$

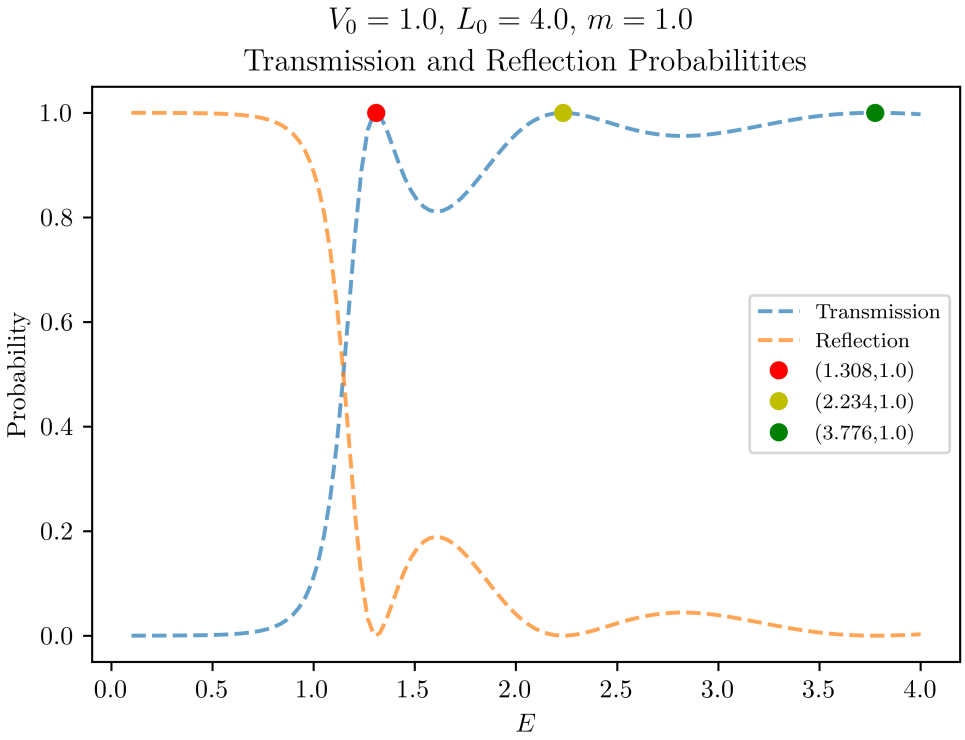


Fig. 4.11 – Transmission Probabilities of the Finite Barrier for $M = 1.0, L = 4.0, U = 1.0$.

$$= \frac{(k^2 - \alpha^2)^2 \sin^2 \alpha L}{4k^2\alpha^2 + (k^2 - \alpha^2)^2 \sin^2 \alpha L}$$

Hence,

$$T = \frac{4k^2\alpha^2}{4k^2\alpha^2 + (k^2 - \alpha^2)^2 \sin^2 \alpha L}$$

Resonant transmission:

$$\sin \alpha L = 0$$

$$\Rightarrow \alpha = \frac{n\pi}{L}, n \in \mathbb{Z}^+$$

$$\Rightarrow k^2 - 2MU = (\frac{n\pi}{L})^2$$

$$\Rightarrow k = \sqrt{(\frac{n\pi}{L})^2 + 2MU}, n \in \mathbb{Z}^+ \tag{4.29}$$

Case II: $E = U$

$$R = \frac{k^2 L^2}{k^2 L^2 + 4}$$

Hence,

$$T = \frac{4}{k^2 L^2 + 4}$$

Resonant transmission: $kL = 0$ which is absurd.

Case III: $E < U$

$$R = \frac{(k^2 + \alpha^2)^2 \sinh^2 \alpha L}{4k^2\alpha^2 \cosh^2 \alpha L + (k^2 - \alpha^2)^2 \sinh^2 \alpha L}$$

$$= \frac{(k^2 + \alpha^2)^2 \sinh^2 \alpha L}{4k^2\alpha^2 + (k^2 + \alpha^2)^2 \sinh^2 \alpha L}$$

Hence,

$$T = \frac{4k^2\alpha^2}{4k^2\alpha^2 + (k^2 + \alpha^2)^2 \sinh^2 \alpha L}$$

Resonant transmission:

$$\sinh \alpha L = 0$$

$$\Rightarrow \alpha = \frac{in\pi}{L}, n \in \mathbb{Z}^+$$

$$\Rightarrow 2MU - k^2 = (\frac{in\pi}{L})^2$$

$$\Rightarrow k = \sqrt{2MU - (\frac{in\pi}{L})^2}$$

$$= \sqrt{2MU + (\frac{n\pi}{L})^2}$$

which is absurd since $E < U$.

Figure 4.11 shows the transmission probabilities of the finite barrier for $M = 1.0, L = 4.0, U = 1.0$. The peaks for transmission probability are observed at $E = [\frac{(\frac{\pi}{4})^2}{2} + 1, \frac{(\frac{\pi}{2})^2}{2} + 1, \frac{(\frac{3\pi}{4})^2}{2} + 1]$. They become broader with higher energy.

4.2.3 Uncertainty Measures

Figure 4.12 shows the position (a) and momentum (b) standard deviations and the Heisenberg product (c) for the finite barrier with $E = (0, 4), M = 1.0, L = 4.0, U = 1.0$. $E = [\frac{(\frac{\pi}{4})^2}{2} + 1, \frac{(\frac{\pi}{2})^2}{2} + 1, \frac{(\frac{3\pi}{4})^2}{2} + 1]$ which are known to show resonant transmission, are highlighted.
(a) The resonances lie close to σ_x maxima but the peaks are broad and short, making them hard to distinguish.
(b) The first resonance shows σ_p minima while the rest are close to minima but the anti-peaks are not clearly defined.
(c) Similar to σ_p, the minima of the first resonance is distinguishable while the others are not.

Figure 4.13 shows the position (a) and momentum (b) Shannon entropies and the entropy sum (c) for the finite barrier with $E = (0, 4), M = 1.0, L = 4.0, U = 1.0$. $E = [\frac{(\frac{\pi}{4})^2}{2} + 1, \frac{(\frac{\pi}{2})^2}{2} + 1, \frac{(\frac{3\pi}{4})^2}{2} + 1]$ which are known to show resonant transmission, are highlighted.
(a) The resonances lie on S_x maxima and though the peaks are broad, the first two are distinguishable.
(b) The first resonance shows S_k minima while the rest are unremarkable.
(c) Similar to S_k, the minima of the first resonance is distinguishable while the others are not.

$V_0 = 1.0,\ L_0 = 4.0,\ m = 1.0$

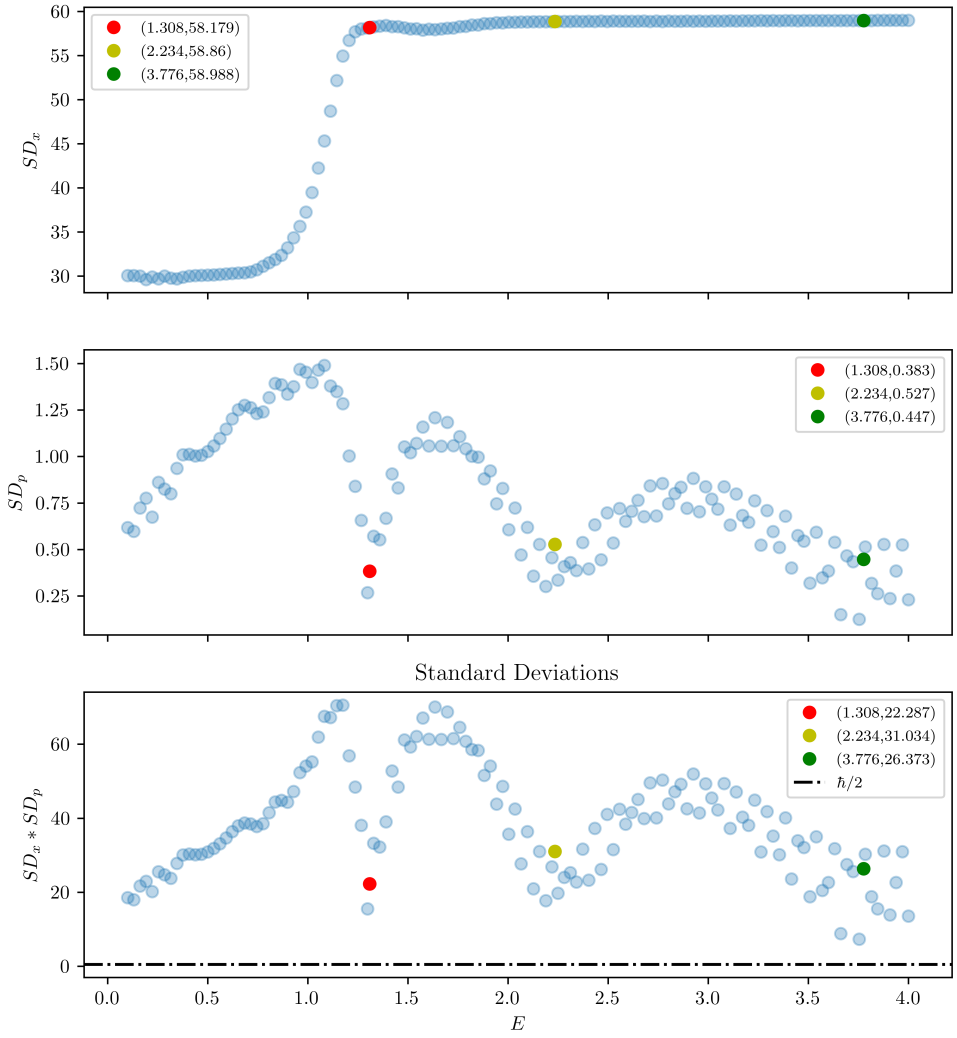


Fig. 4.12 – Standard Deviations for the Finite Barrier with $E = (0, 4), M = 1.0, L = 4.0, U = 1.0$.

$V_0 = 1.0,\ L_0 = 4.0,\ m = 1.0$

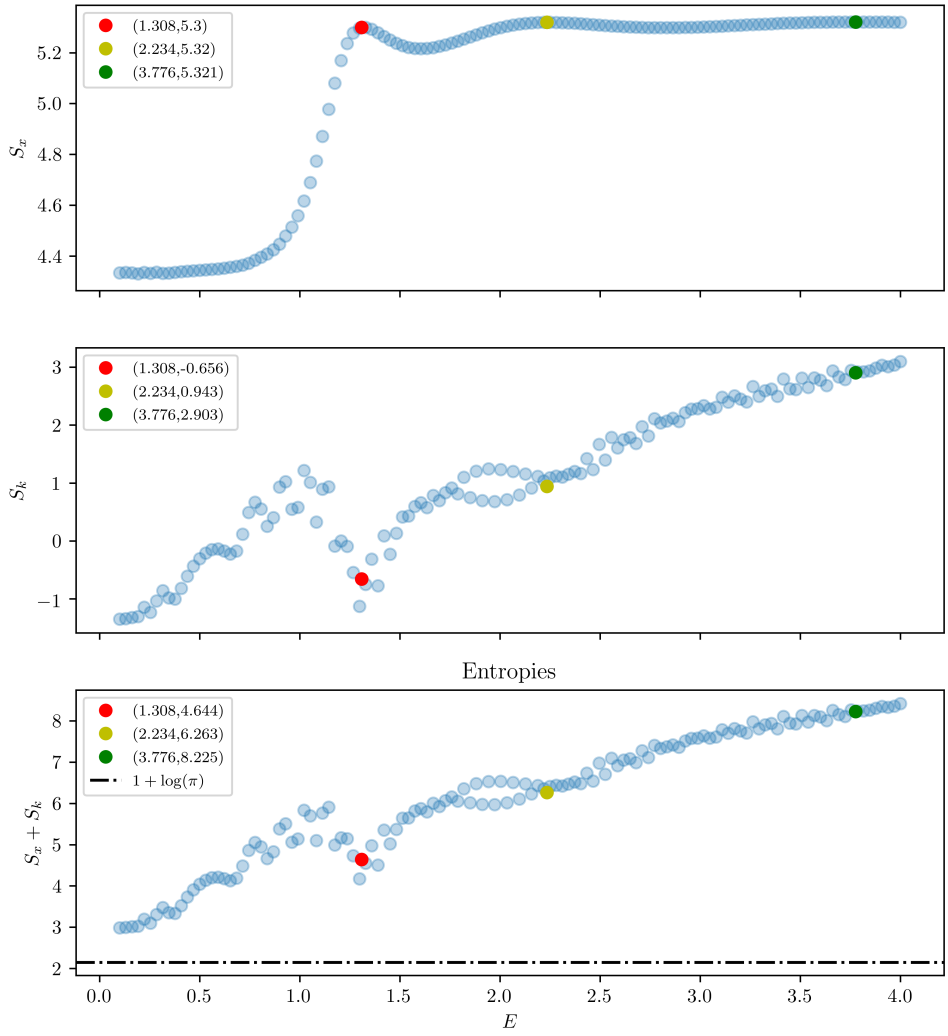


Fig. 4.13 – Shannon Entropies for the Finite Barrier with $E = (0, 4), M = 1.0, L = 4.0, U = 1.0$.

4.2.4 Propagation of Gaussian Wavepacket

Figure 4.14 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the finite barrier with $E = 3.0$ for $M = L = U = 1.0$.
(a) The wavelet undergoes significant deformation in the second and third frames, and there is a noticeable reflection in the fourth frame.
(b) In the third and fourth frames, a small peak is seen developing at $k = -\sqrt{2ME}$. The tall peak at $k = \sqrt{2ME}$ reduces slightly.

Figure 4.15 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the finite barrier with $E = \frac{\pi^2}{2} + 1$ for $M = L = U = 1.0$.
(a) The wavelet undergoes minimal deformation in the second and third frames, and no reflection can be seen in the fourth frame.
(b) From the first through to the fourth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

Figure 4.16 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the finite barrier with $E = 9.7$ for $M = L = U = 1.0$.
(a) The wavelet undergoes noticeable deformation in the second and third frames, but no reflection can be seen in the fourth frame.
(b) From the first through to the fourth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

Figure 4.17 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the finite barrier with $E = 13.4$ for $M = L = U = 1.0$.
(a) The wavelet undergoes slight deformation in the second and third frames, but no reflection can be seen in the fourth frame.
(b) From the first through to the fourth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

Figure 4.18 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the finite barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $M = L = U = 1.0$.
(a) The wavelet undergoes no deformation at all in the second and third frames, and no reflection can be seen in the fourth frame.
(b) From the first through to the fourth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

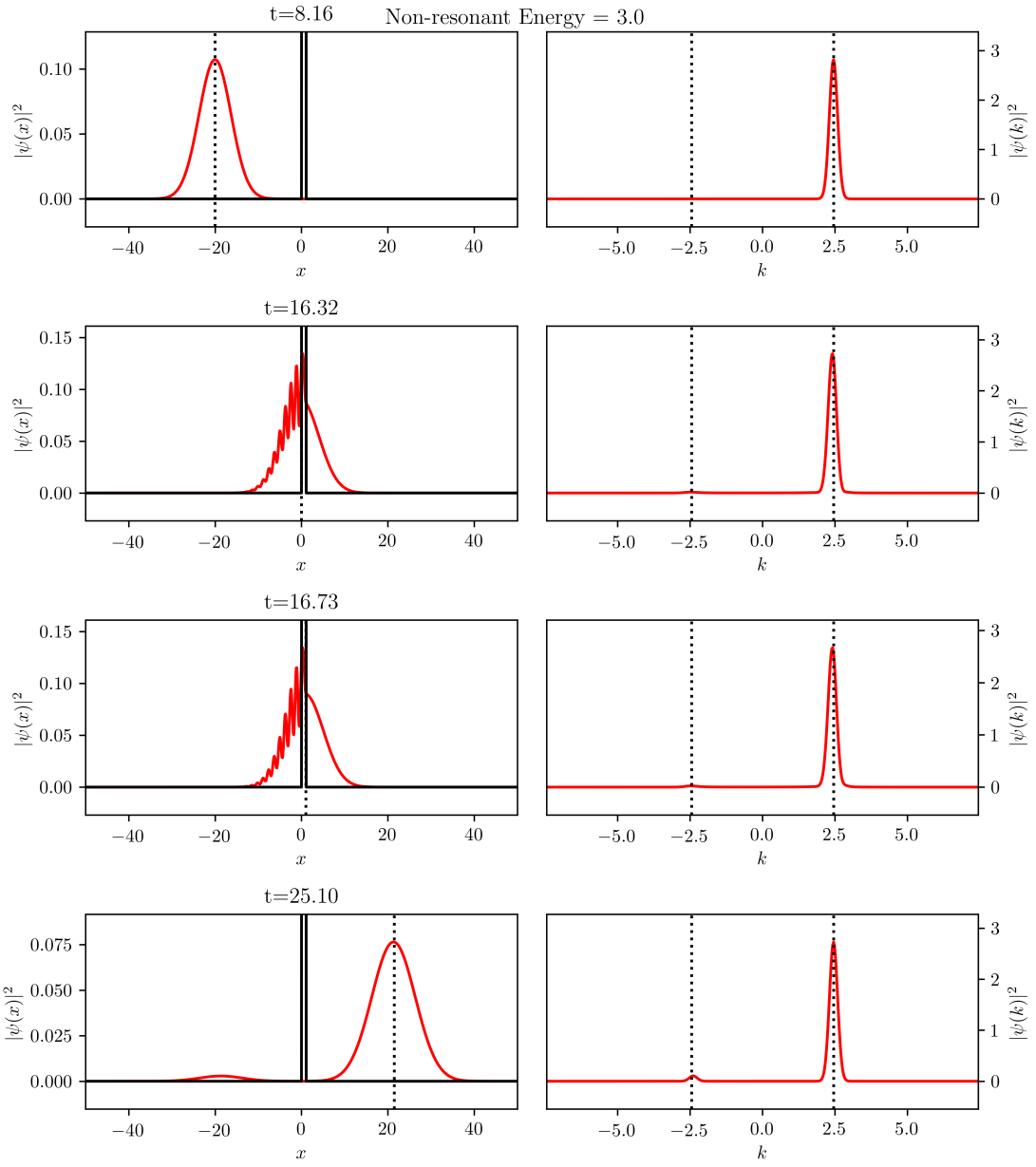


Fig. 4.14 – Position and Momentum Densities of wavelet through the Finite Barrier with $E = 3.0$ for $M = L = U = 1.0$.

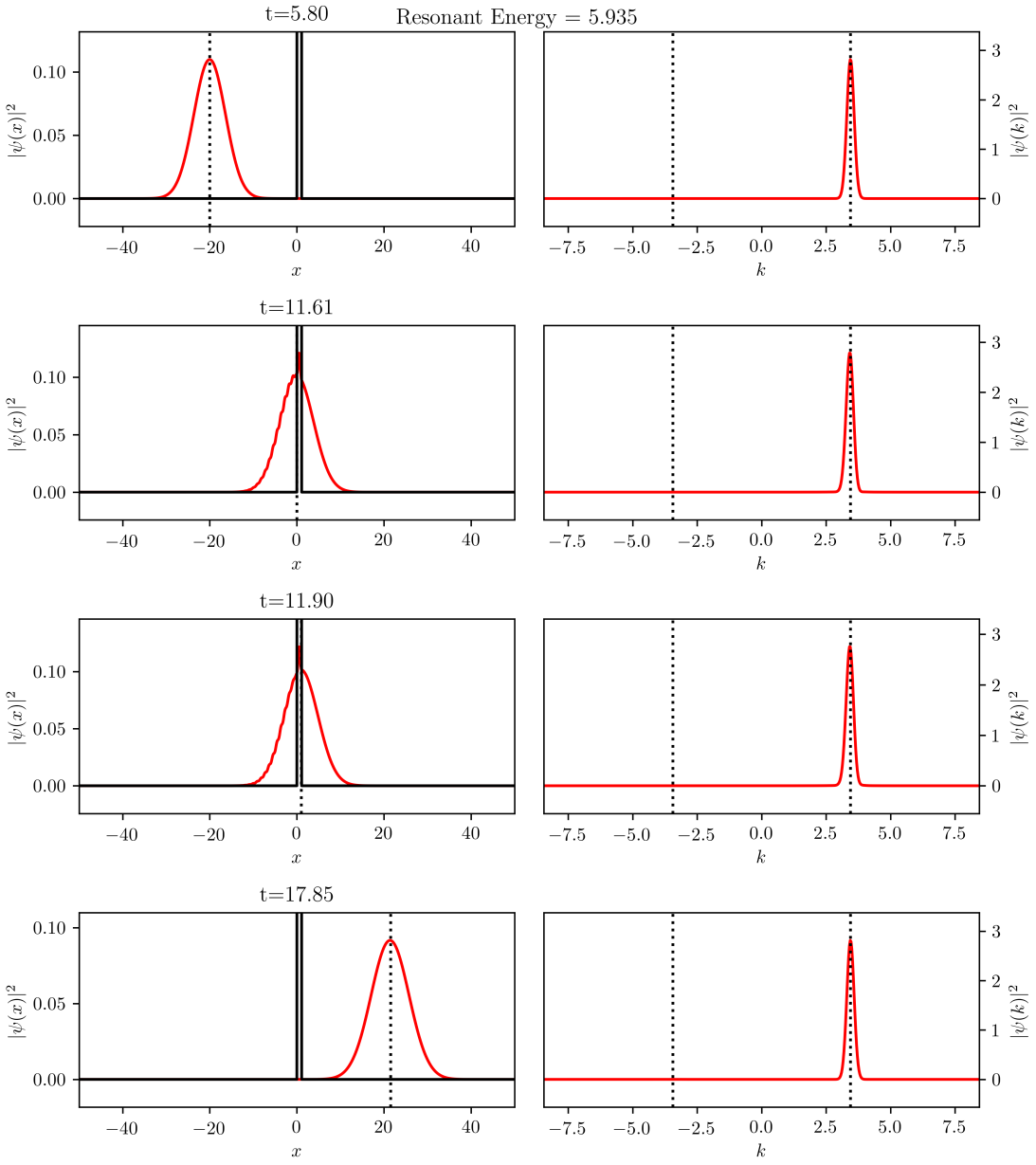


Fig. 4.15 – Position and Momentum Densities of wavelet through the Finite Barrier with $E = \frac{\pi^2}{2} + 1$ for $M = L = U = 1.0$.

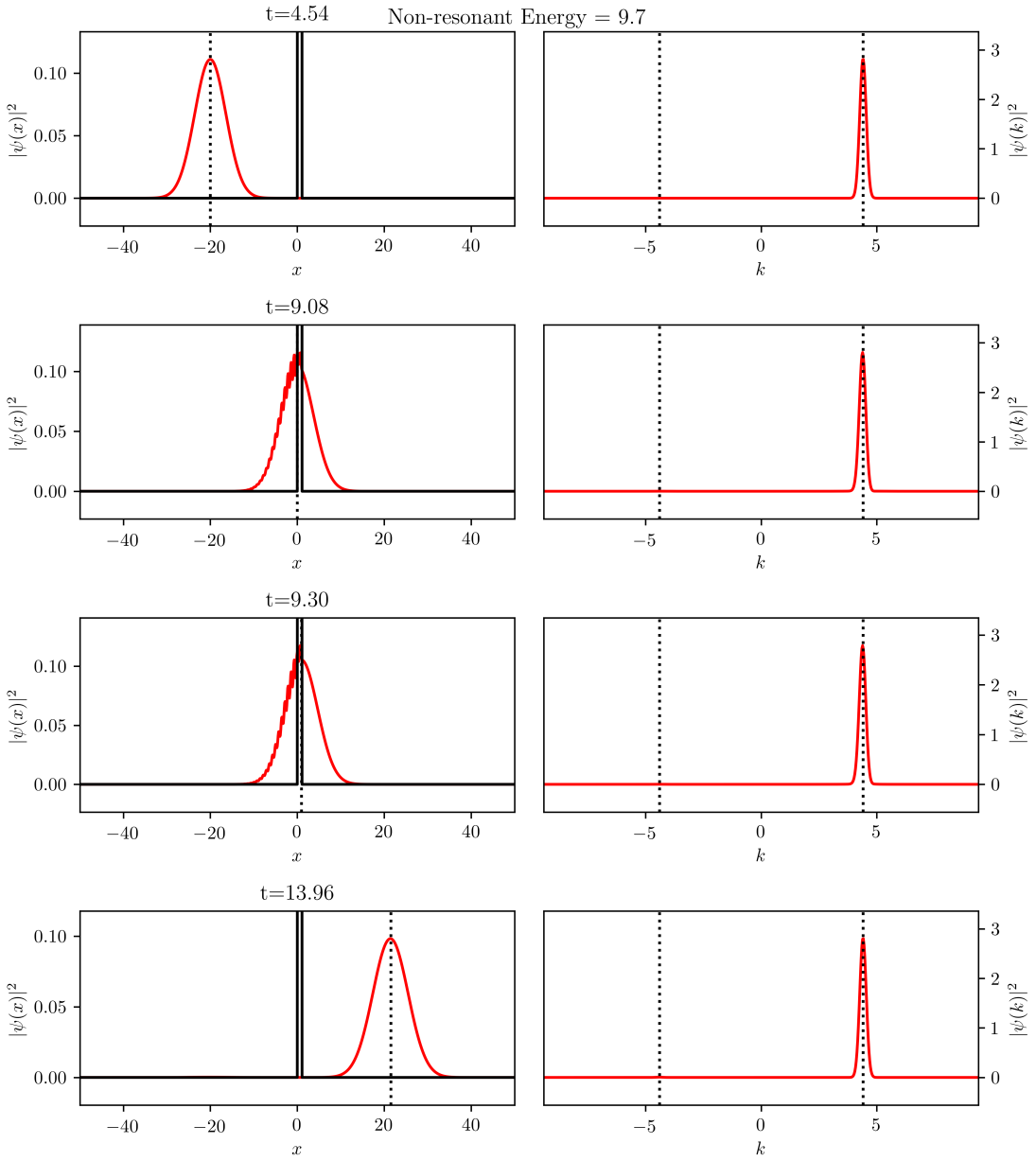


Fig. 4.16 – Position and Momentum Densities of wavelet through the Finite Barrier with $E = 9.7$ for $M = L = U = 1.0$.

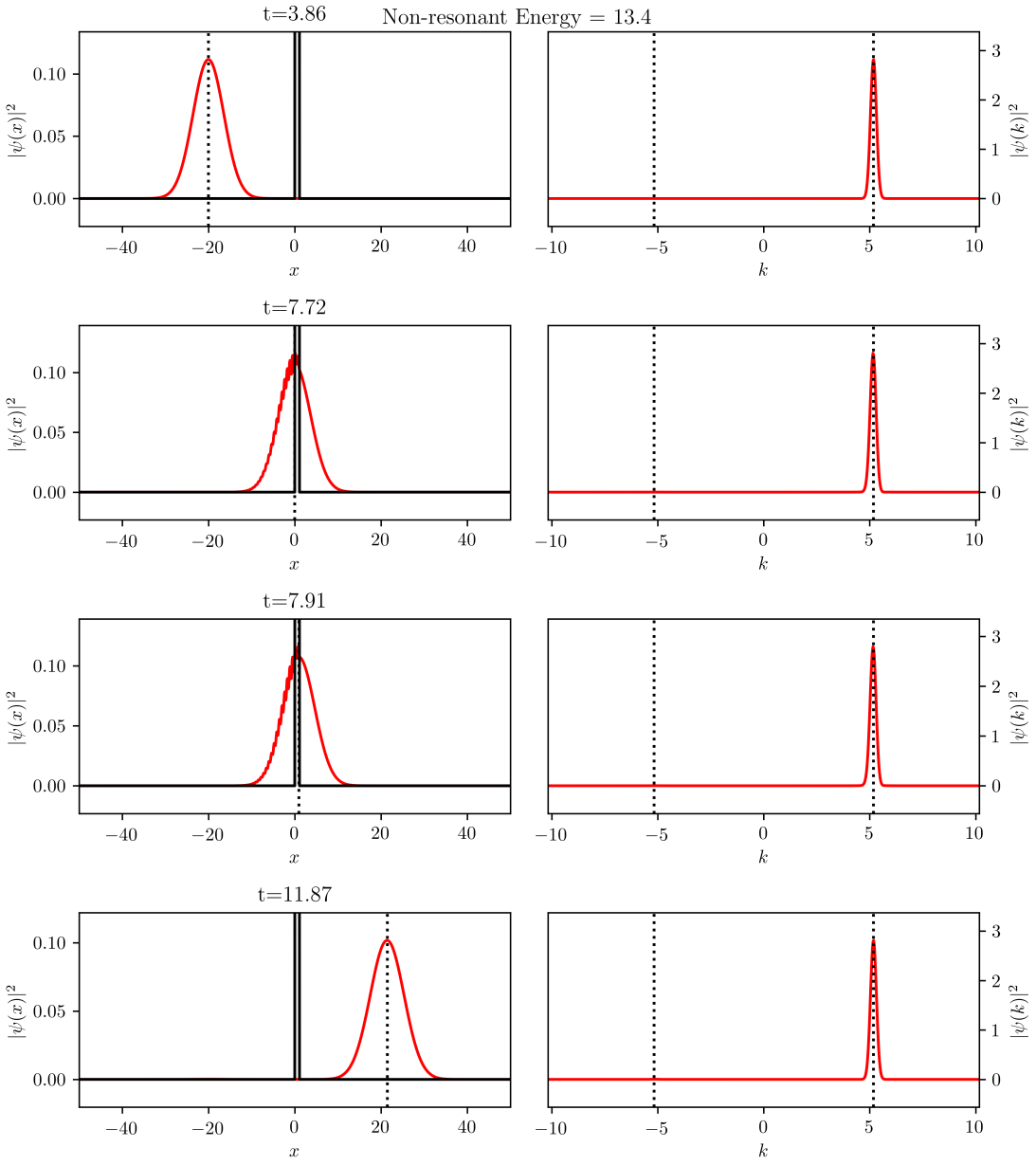


Fig. 4.17 – Position and Momentum Densities of wavelet through the Finite Barrier with $E = 13.4$ for $M = L = U = 1.0$.

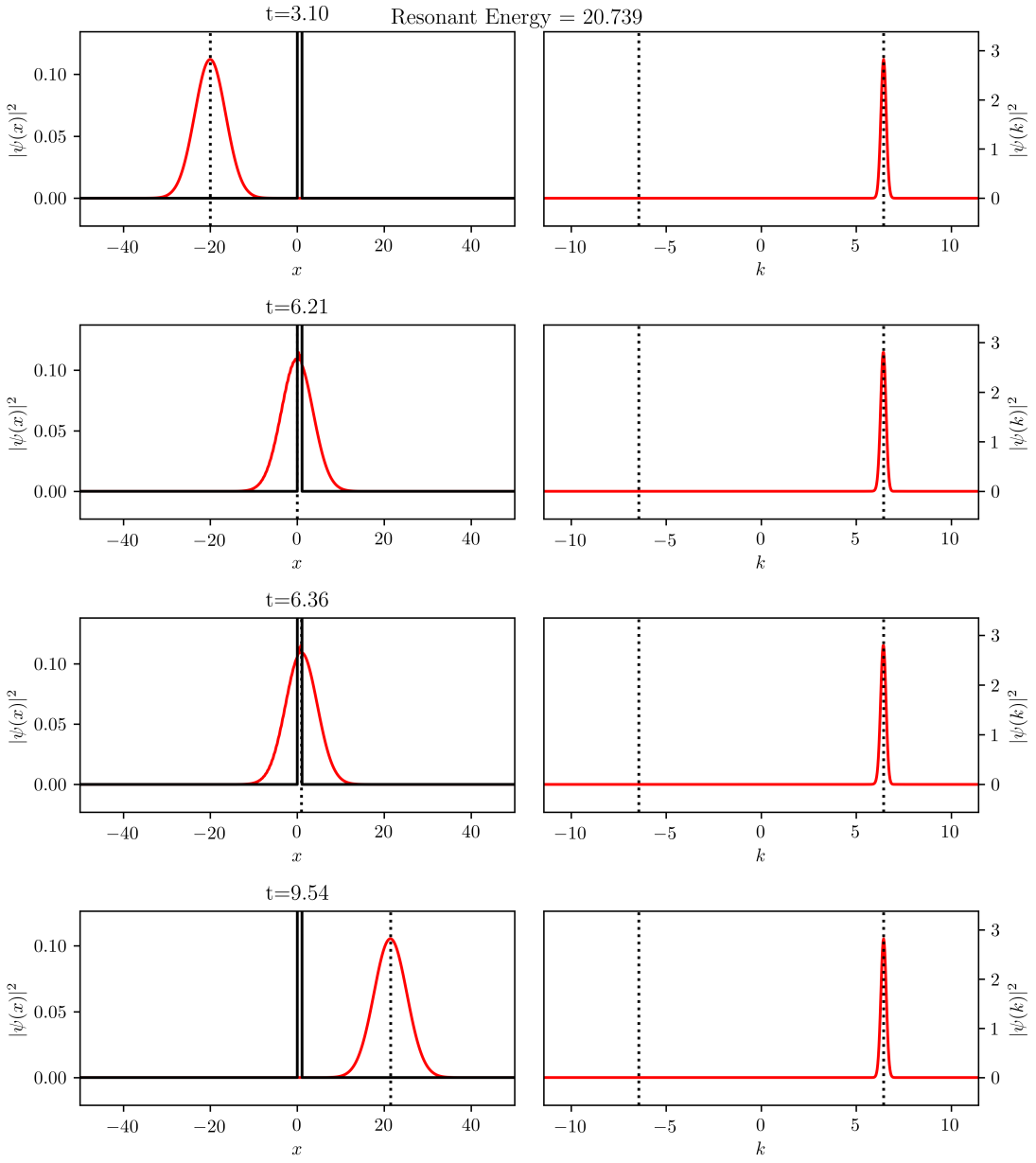


Fig. 4.18 – Position and Momentum Densities of wavelet through the Finite Barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $M = L = U = 1.0$.

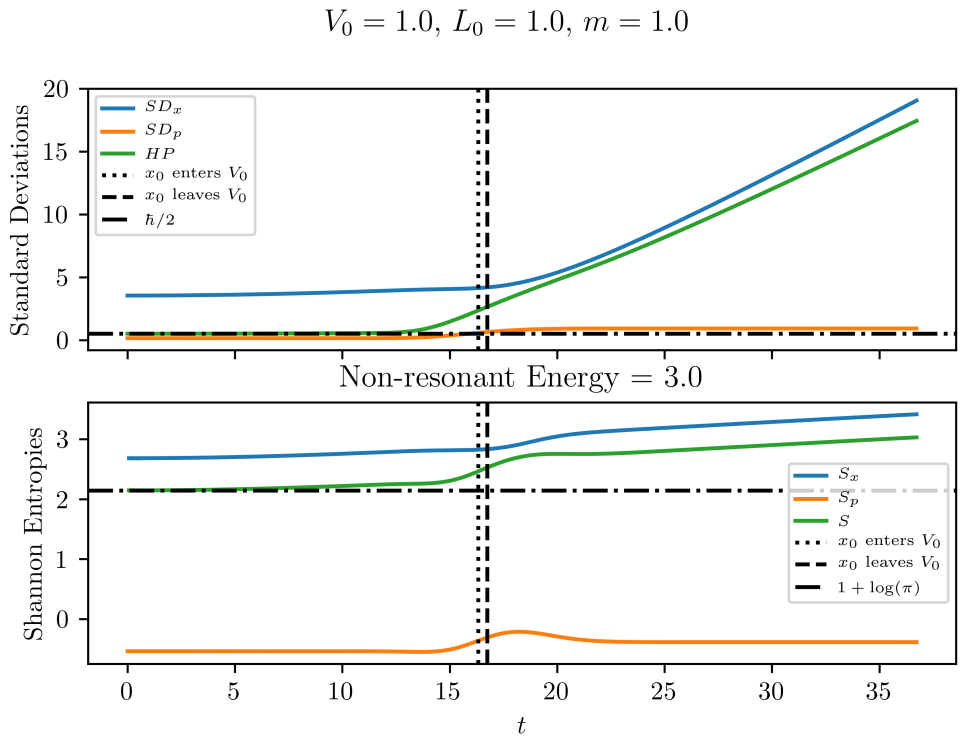


Fig. 4.19 – Uncertainties of wavelet through the Finite Barrier with $E = 3.0$ for $M = L = U = 1.0$.

Figure 4.19 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the finite barrier with $E = 3.0$ for $M = L = U = 1.0$.
(a) σ_x and consequently, the product blow up post-interaction.
(b) S_x and hence, the sum experience a jump post-interaction but the latter remains around the lower bound.

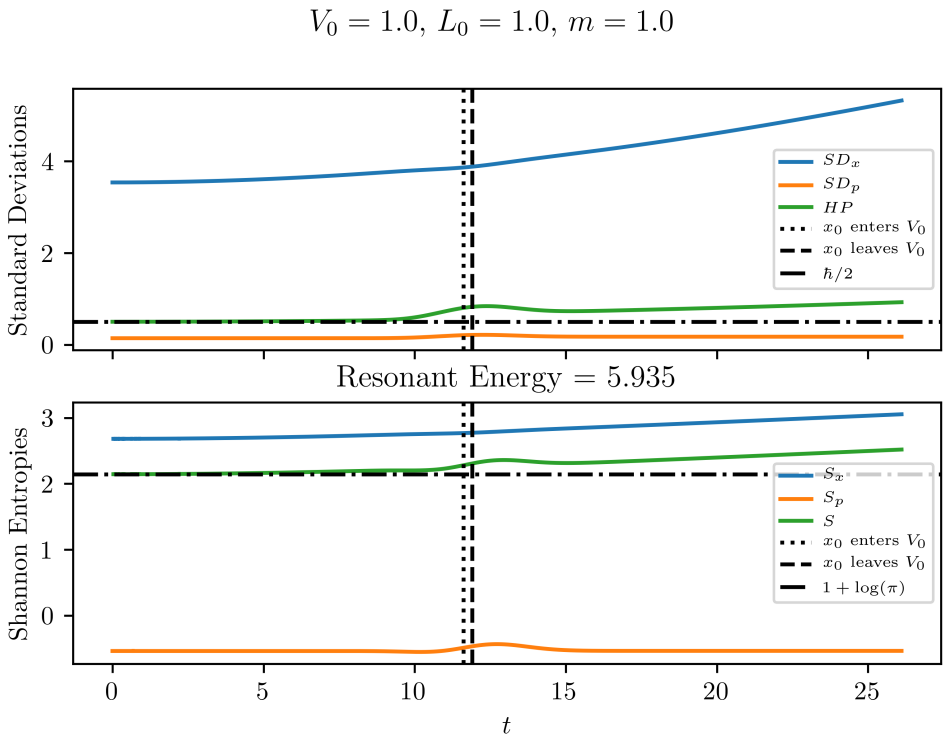


Fig. 4.20 – Uncertainties of wavelet through the Finite Barrier with $E = \frac{\pi^2}{2} + 1$ for $M = L = U = 1.0$.

Figure 4.20 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the finite barrier with $E = \frac{\pi^2}{2} + 1$ for $M = L = U = 1.0$.
(a) Though σ_x increases a bit post-interaction, the product remains around the lower bound.
(b) S_x and hence, the sum experience a slight jump post-interaction but the latter remains around the lower bound.

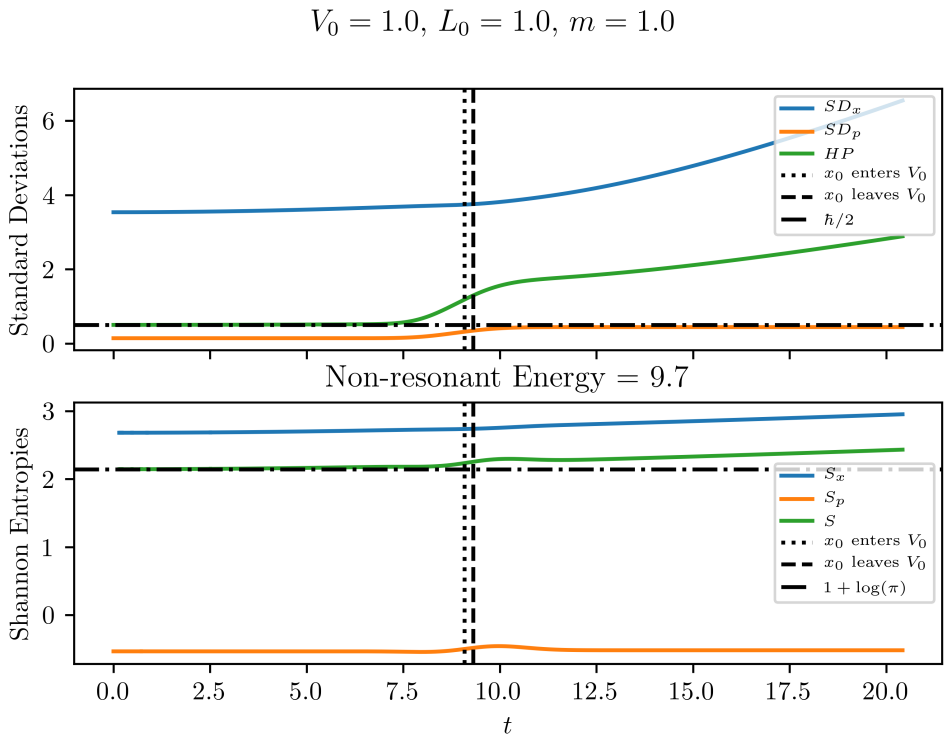


Fig. 4.21 – Uncertainties of wavelet through the Finite Barrier with $E = 9.7$ for $M = L = U = 1.0$.

Figure 4.21 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the finite barrier with $E = 9.7$ for $M = L = U = 1.0$.
(a) σ_x and consequently, the product hike up significantly post-interaction.
(b) S_x and hence, the sum again experience a slight jump post-interaction but the latter remains around the lower bound.

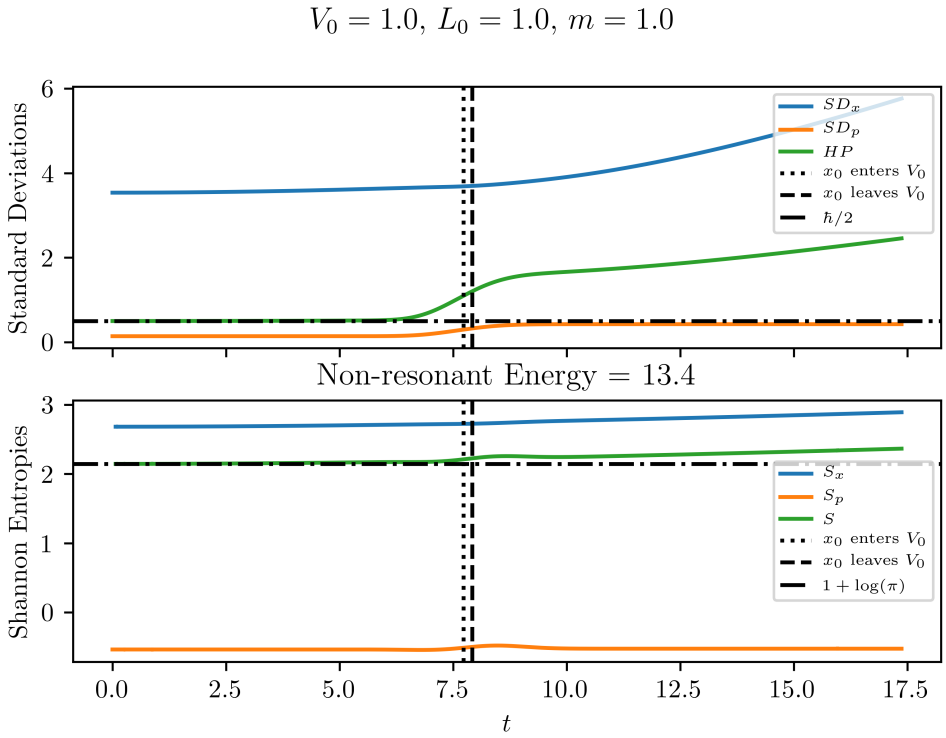


Fig. 4.22 – Uncertainties of wavelet through the Finite Barrier with $E = 13.4$ for $M = L = U = 1.0$.

Figure 4.22 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the finite barrier with $E = 13.4$ for $M = L = U = 1.0$.
(a) σ_x and consequently, the product increase significantly post-interaction.
(b) S_x and hence, the sum experience a very slight jump post-interaction but the latter remains around the lower bound.

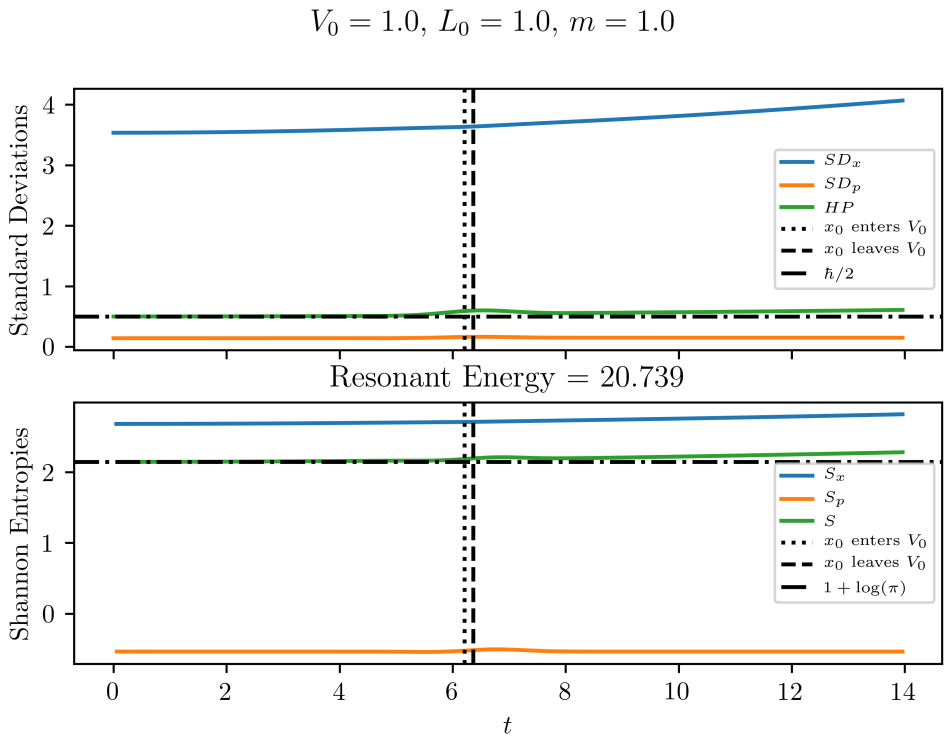


Fig. 4.23 – Uncertainties of wavelet through the Finite Barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $M = L = U = 1.0$.

Figure 4.23 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the finite barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $M = L = U = 1.0$.
(a) σ_x and the product remain almost constant, the latter very close to the lower bound.
(b) S_x and the sum remain almost constant, the latter again very close to the lower bound.

4.2.5 Error Analysis

4.2.5.1 Stationary Wavefunctions

Mass of the particle $M = 1.0$
Length of the barrier $L = 4.0$
Height of the barrier $U = 1.0$

A total of N samples is taken with spacing Δx for the range of integration in x which is symmetric about 0.

For k, again N samples are taken symmetrically about 0. However, the k space entirely comprises of a Dirac delta or comb function, so delta function normalization is implemented around the peaks with a radius of B.

Table 4.2 shows the error analysis of stationary wavefunctions for the finite barrier. $N = 2^{11}, \Delta x = 0.1, B = 0.5$ are chosen.

Tab. 4.2 – Error Analysis of Stationary Wavefunctions for the Finite Barrier

E	$\int \lvert\psi(x)\rvert^2 dx$	$\int_{\text{around} \pm k_0} \lvert\psi(k)\rvert^2 dk$
0.5	1.0	1.0000000000000004
1.0	1.0	1.0
$\frac{(\frac{\pi}{4})^2}{2} + 1$	1.0	0.9999999999999998
1.8	1.0	1.0000000000000002
$\frac{(\frac{\pi}{2})^2}{2} + 1$	1.0	0.9999999999999986
3.0	0.9999999999999999	0.999999999999999
$\frac{(\frac{3\pi}{4})^2}{2} + 1$	1.0	0.9999999999999987

4.2.5.2 Gaussian Wavepacket

Mass of the particle $M = 1.0$
Length of the barrier $L = 1.0$
Height of the barrier $U = 1.0$

A total of N samples is taken with spacing Δx for the range of integration in x which is symmetric about 0.

For k, again N samples are taken symmetrically about 0 but with spacing $\Delta p = \frac{2\pi}{N\Delta x}$

The position and momentum norms of the wavelet are calculated for the time instant of the fourth frame post-interaction, so the time varies for different energies.

Table 4.3 shows the error analysis of the Gaussian wavepacket through the finite barrier. $N = 2^{11}, \Delta x = 0.1$ are chosen.

Tab. 4.3 – Error Analysis of Gaussian Wavepacket through the Finite Barrier

E	$\int \lvert\psi(x)\rvert^2 dx$	$\int \lvert\psi'(k)\rvert^2 dk$
3.0	1.0000000000000924	1.0000000000000209
$\frac{(\pi)^2}{2} + 1$	1.000000000000061	1.000000000000003
9.7	1.000000000000034	0.9999999999999769
13.4	1.0000000000000009	0.9999999999999442
$\frac{(2\pi)^2}{2} + 1$	1.0000000000000362	0.9999999999999798

4.3 Symmetric Double Barrier

$$U(x) = \begin{cases} U_1 & x \in [a, a + L_1] \\ U_2 & x \in [a + L_1 + d, a + L_1 + d + L_2] \\ 0 & \text{elsewhere} \end{cases}$$

Since the double barrier is symmetric, $L_1 = L_2, U_1 = U_2$.

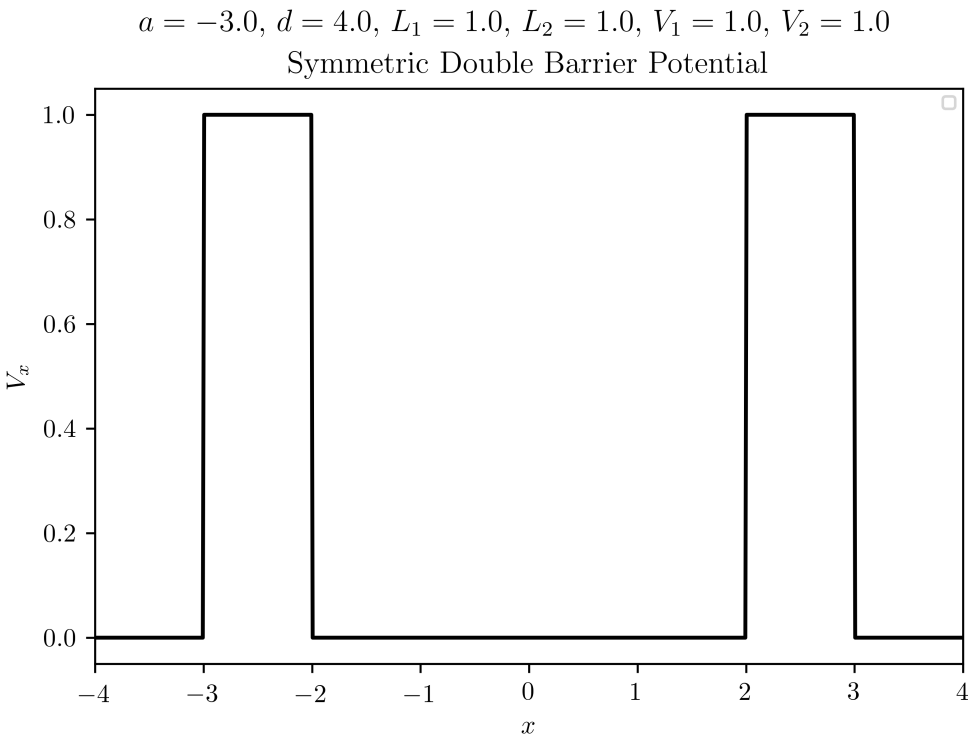


Fig. 4.24 – Potential for a Symmetric Double Barrier for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Figure 4.24 shows the potential for a symmetric double barrier for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Let us divide the system into five regions:

1. $x < -\frac{d}{2} - L_1$: Left of the first barrier
2. $-\frac{d}{2} - L_1 \leq x \leq -\frac{d}{2}$: Inside the first barrier
3. $-\frac{d}{2} < x < \frac{d}{2}$: Between the barriers
4. $\frac{d}{2} \leq x \leq \frac{d}{2} + L_2$: Inside the second barrier
5. $\frac{d}{2} + L_2 < x$: Right of the second barrier

4.3.1 Propagation of Gaussian Wavepacket

Figure 4.25 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the symmetric double barrier with $E = 3.0$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.
(a) The wavelet undergoes significant deformation in the second, third and fourth frames, and there is a noticeable reflection in the fifth frame.
(b) From the third frame onward, a small peak is seen developing at $k = -\sqrt{2ME}$. The tall peak at $k = \sqrt{2ME}$ reduces slightly.

Figure 4.26 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the symmetric double barrier with $E = \frac{\pi^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.
(a) The wavelet undergoes minimal deformation in the second, third and fourth frames, and no reflection can be seen in the fifth frame.
(b) From the first through to the fifth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

Figure 4.27 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the symmetric double barrier with $E = 9.7$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.
(a) The wavelet undergoes noticeable deformation in the second, third and fourth frames, and no reflection can be seen in the fifth frame.
(b) From the first through to the fifth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

Figure 4.28 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the symmetric double barrier with $E = 13.4$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.
(a) The wavelet undergoes very slight deformation in the second, third and fourth frames, and no reflection can be seen in the fifth frame.
(b) From the first through to the fifth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

Figure 4.29 shows the position (a) and momentum (b) densities of a Gaussian wavelet through the symmetric double barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.
(a) The wavelet undergoes no deformation at all in the second, third and fourth frames, and no reflection can be seen in the fifth frame.
(b) From the first through to the fifth frame, the tall peak at $k = \sqrt{2ME}$ remains unchanged and no other peaks develop.

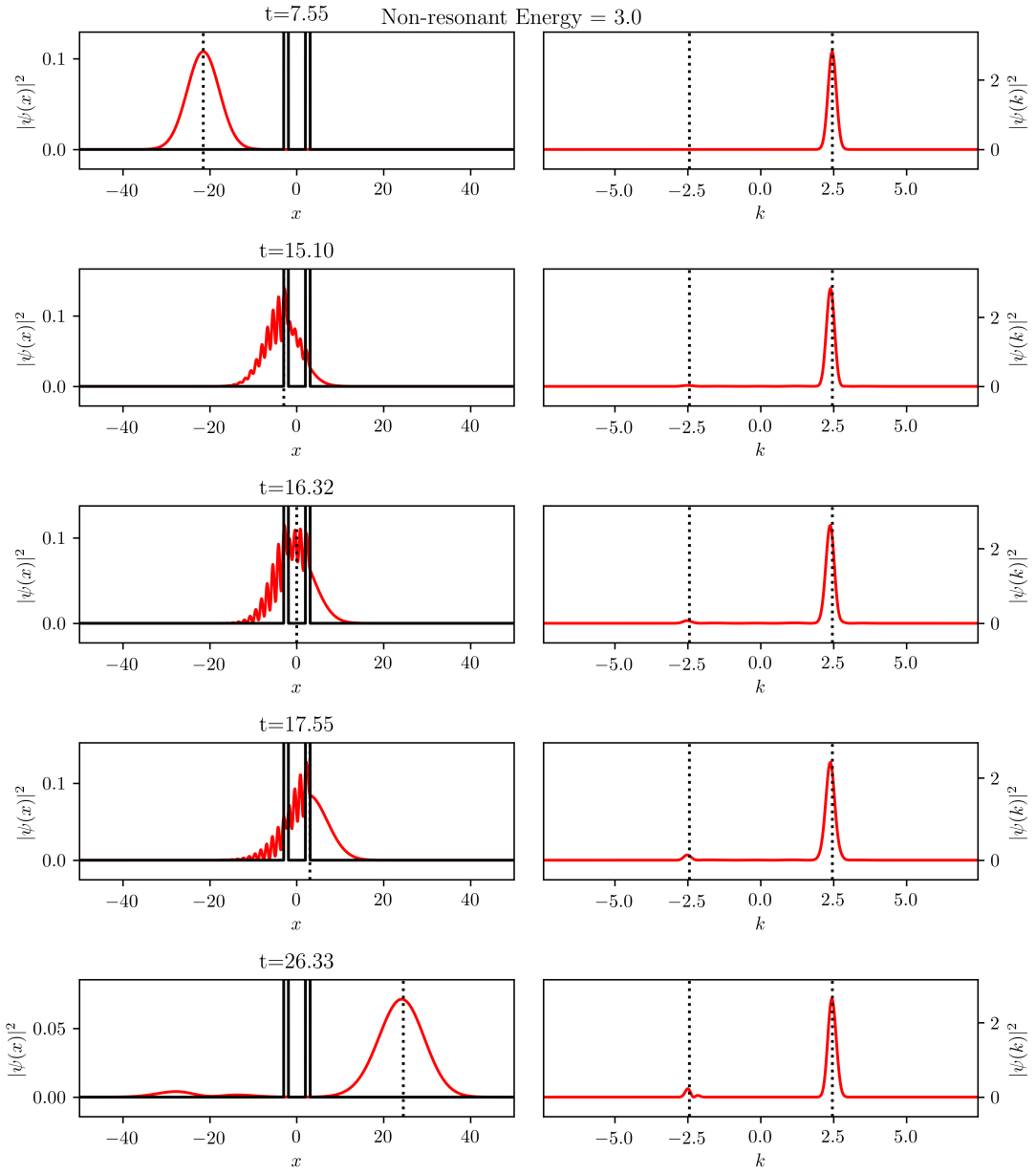


Fig. 4.25 – Position and Momentum Densities of wavelet through the Symmetric Double Barrier with $E = 3.0$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

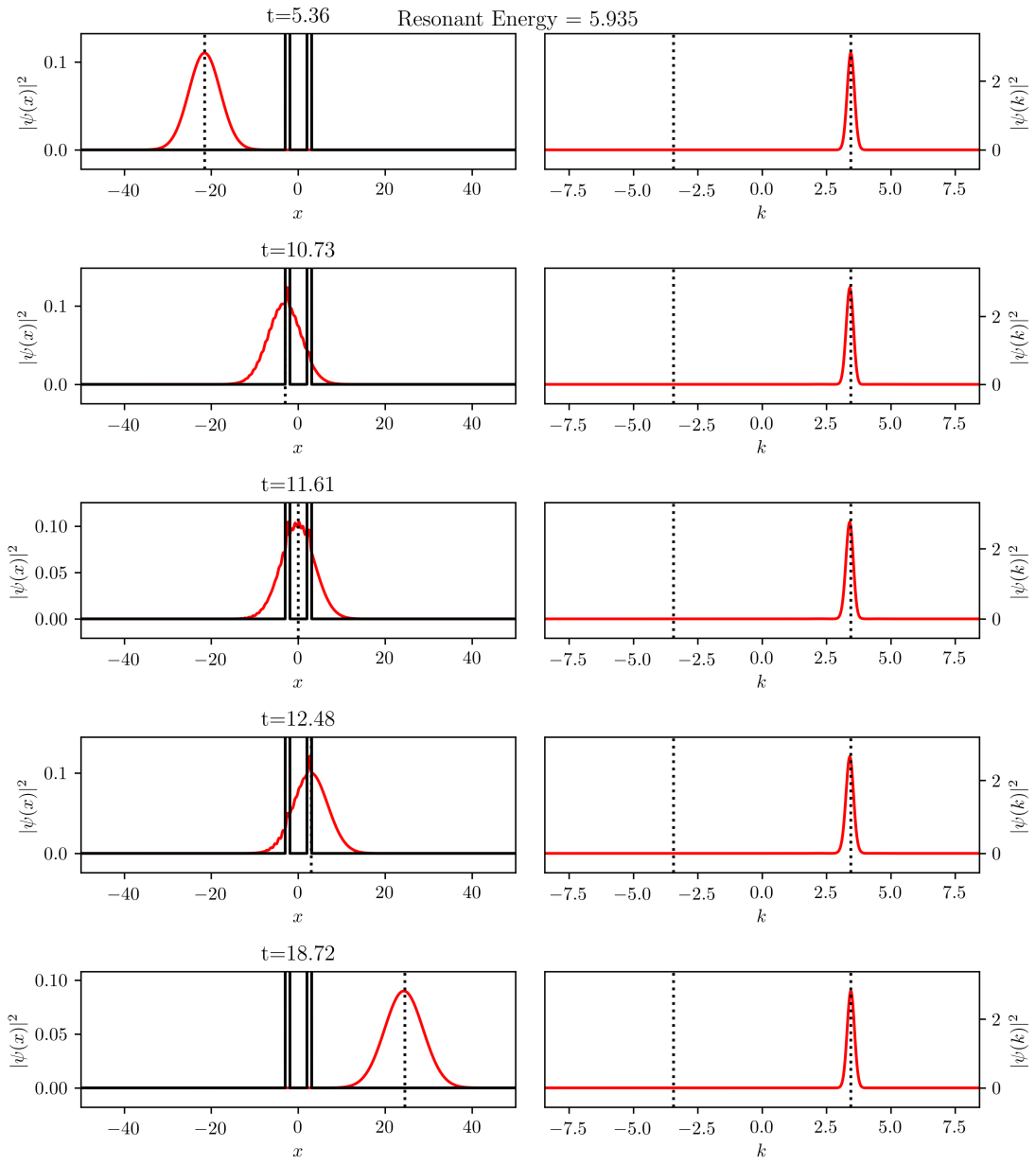


Fig. 4.26 – Position and Momentum Densities of wavelet through the Symmetric Double Barrier with $E = \frac{\pi^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

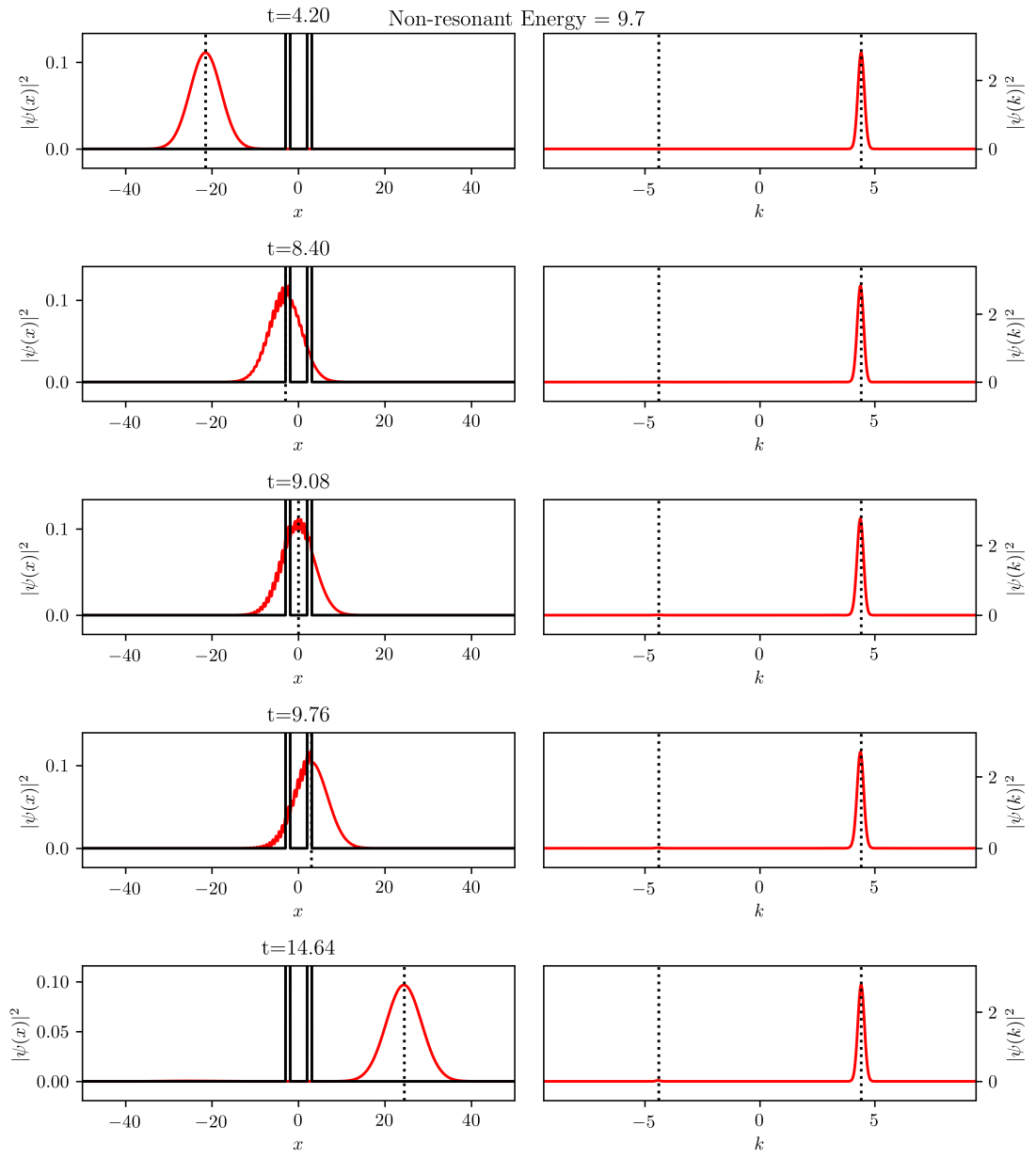


Fig. 4.27 – Position and Momentum Densities of wavelet through the Symmetric Double Barrier with $E = 9.7$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

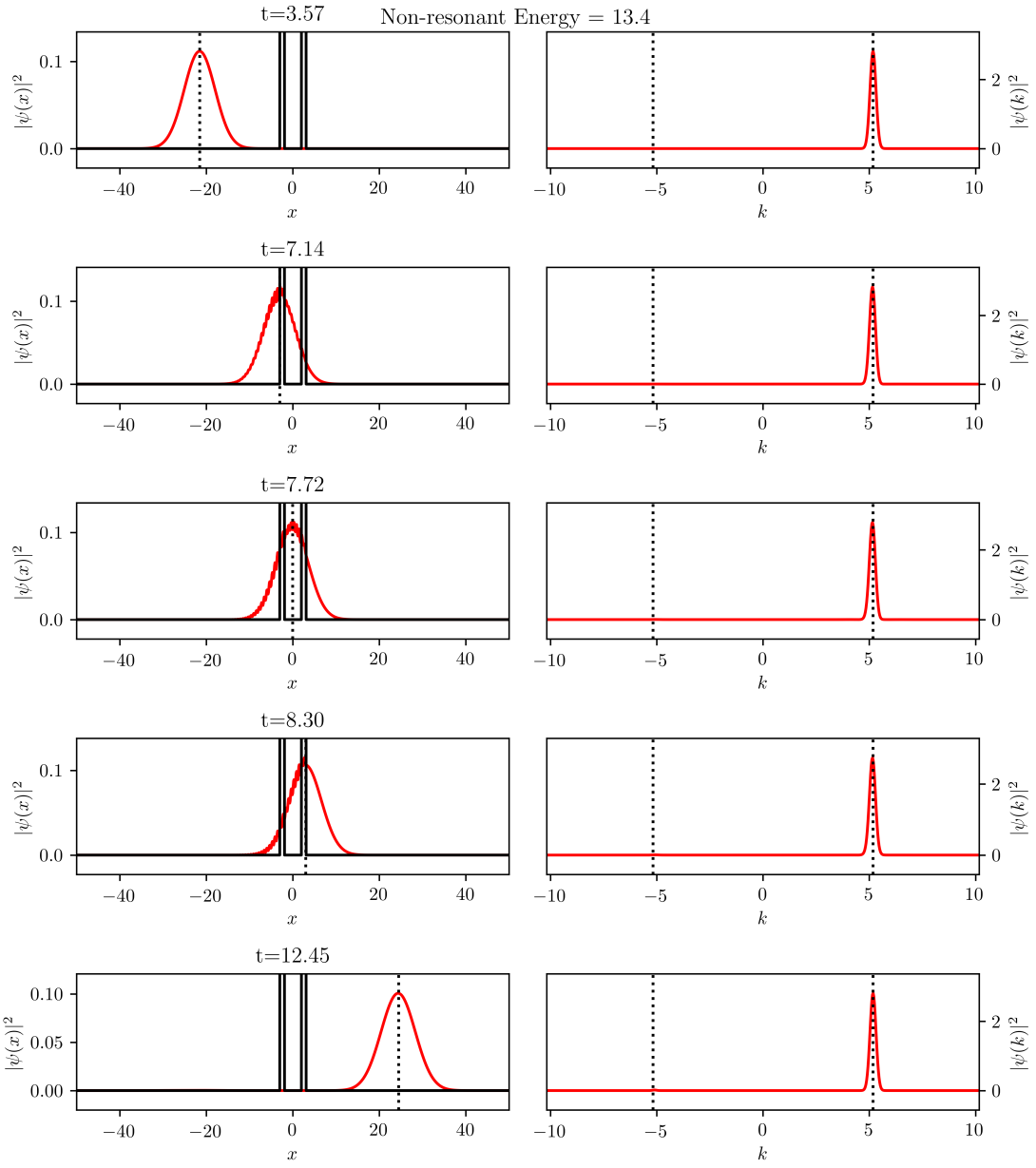


Fig. 4.28 – Position and Momentum Densities of wavelet through the Symmetric Double Barrier with $E = 13.4$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

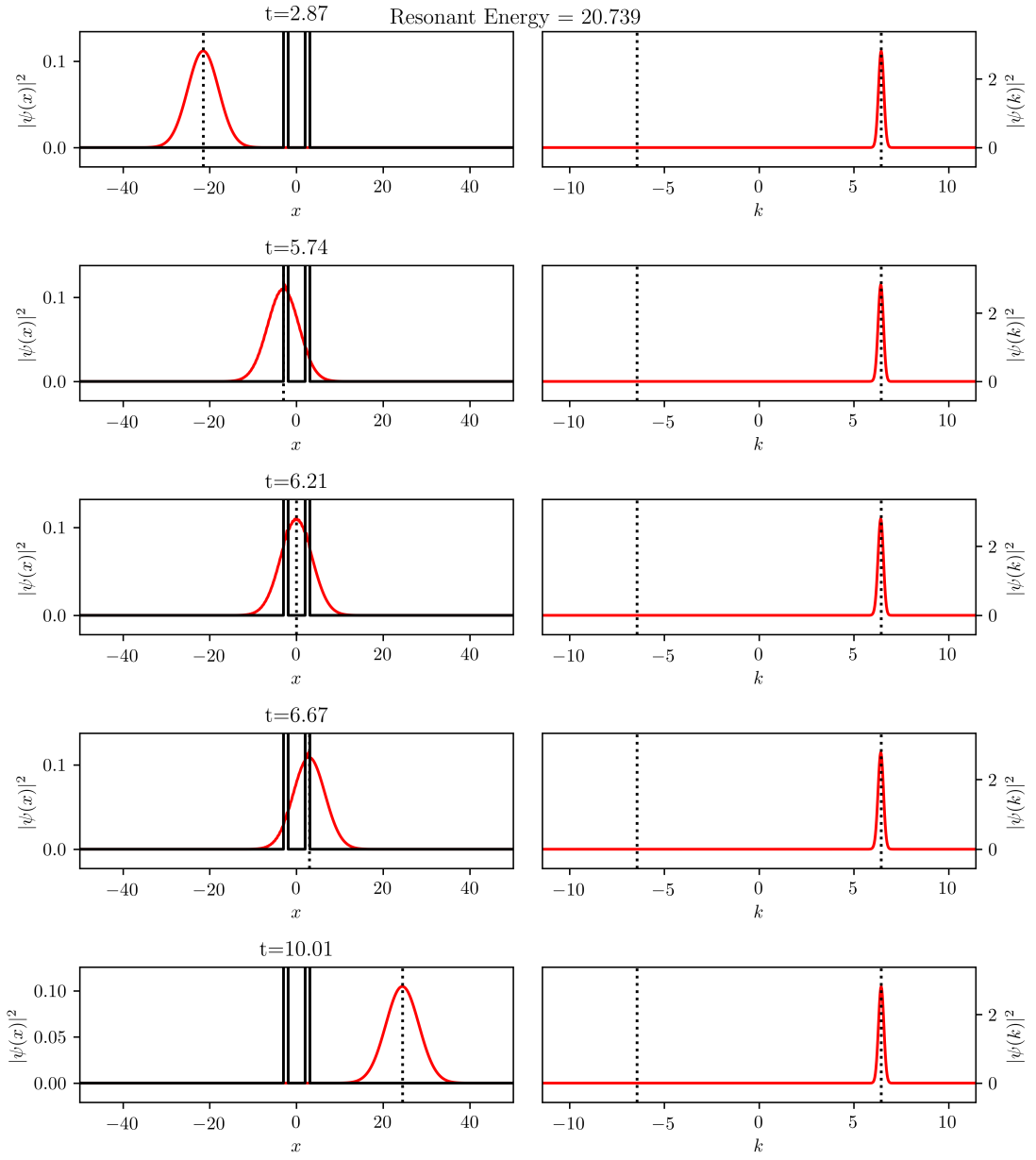


Fig. 4.29 – Position and Momentum Densities of wavelet through the Symmetric Double Barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

$V_1 = 1.0,\ V_2 = 1.0,\ L_1 = 1.0,\ L_2 = 1.0,\ d = 4.0,\ m = 1.0$

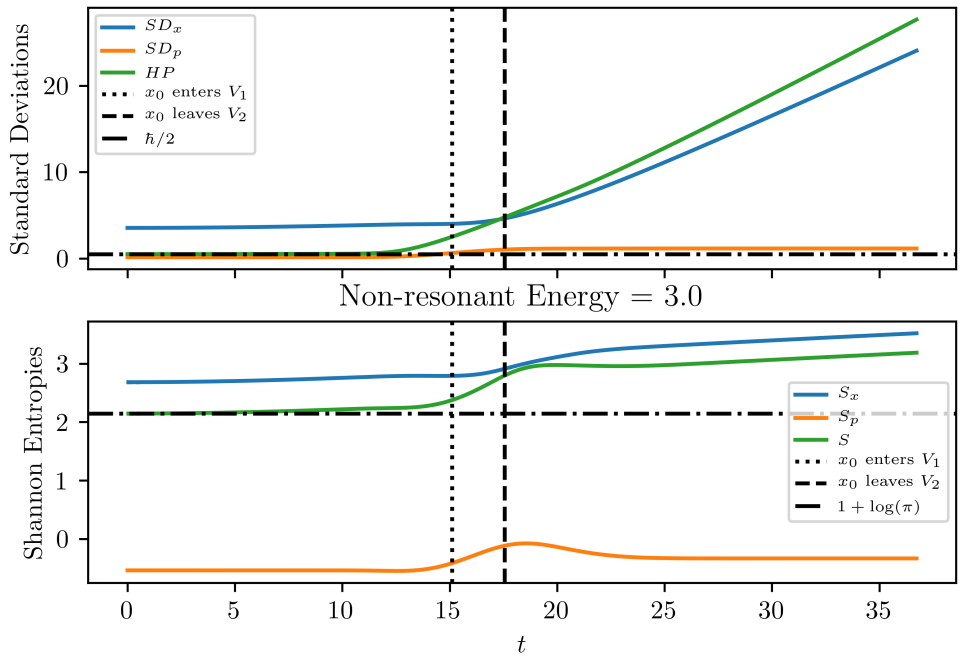


Fig. 4.30 – Uncertainties of wavelet through the Symmetric Double Barrier with $E = 3.0$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Figure 4.30 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the symmetric double barrier with $E = 3.0$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

(a) σ_x and consequently, the product blow up post-interaction.

(b) S_x and hence, the sum experience a jump post-interaction but the latter remains around the lower bound.

$V_1 = 1.0,\ V_2 = 1.0,\ L_1 = 1.0,\ L_2 = 1.0,\ d = 4.0,\ m = 1.0$

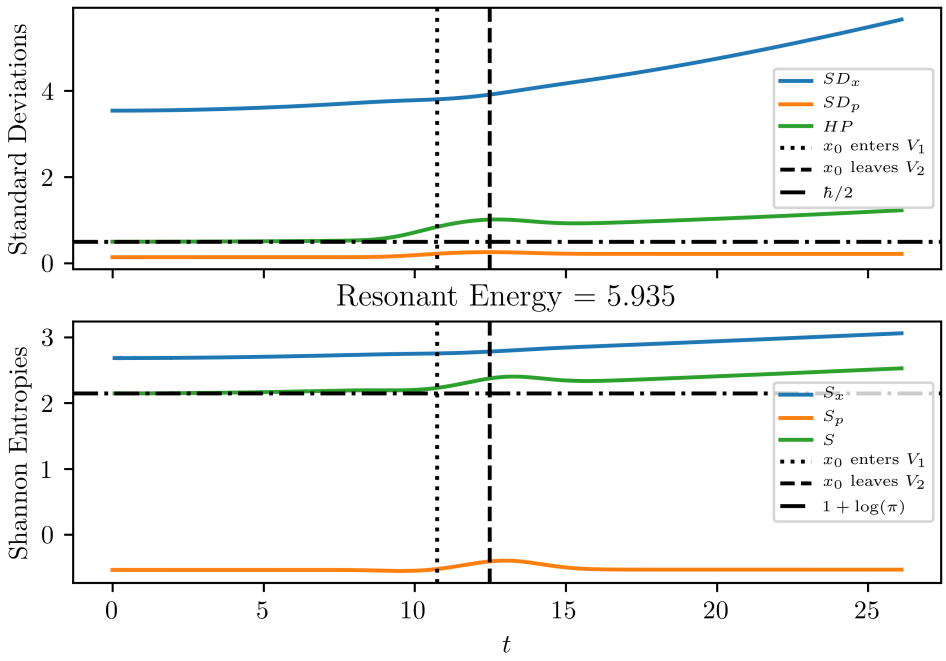


Fig. 4.31 – Uncertainties of wavelet through the Symmetric Double Barrier with $E = \frac{\pi^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Figure 4.31 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the symmetric double barrier with $E = \frac{\pi^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

(a) Though σ_x increases a bit post-interaction, the product remains around the lower bound.

(b) S_x and hence, the sum experience a slight jump post-interaction but the latter remains around the lower bound.

$$V_1 = 1.0,\ V_2 = 1.0,\ L_1 = 1.0,\ L_2 = 1.0,\ d = 4.0,\ m = 1.0$$

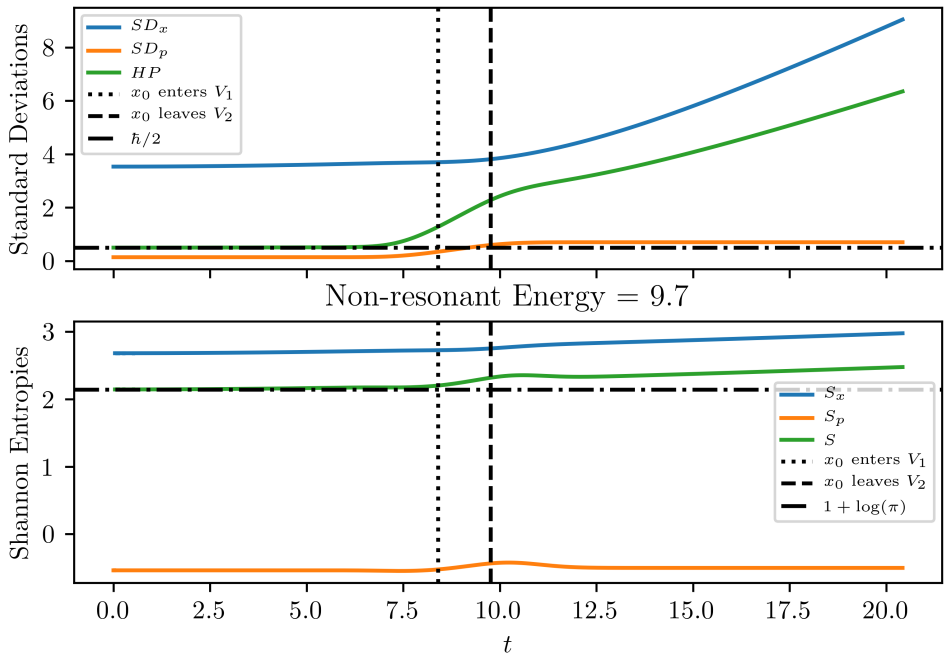


Fig. 4.32 – Uncertainties of wavelet through the Symmetric Double Barrier with $E = 9.7$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Figure 4.32 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the symmetric double barrier with $E = 9.7$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

(a) σ_x and consequently, the product hike up significantly post-interaction.

(b) S_x and hence, the sum again experience a slight jump post-interaction but the latter remains around the lower bound.

$$V_1 = 1.0,\ V_2 = 1.0,\ L_1 = 1.0,\ L_2 = 1.0,\ d = 4.0,\ m = 1.0$$

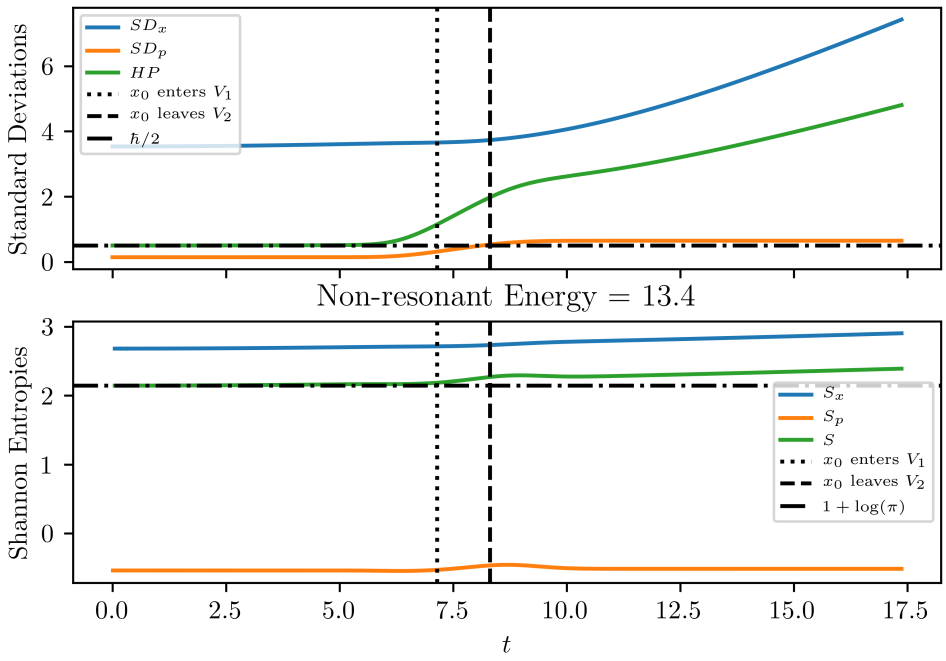


Fig. 4.33 – Uncertainties of wavelet through the Symmetric Double Barrier with $E = 13.4$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Figure 4.33 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the symmetric double barrier with $E = 13.4$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

(a) σ_x and consequently, the product increase significantly post-interaction.

(b) S_x and hence, the sum experience a very slight jump post-interaction but the latter remains around the lower bound.

$V_1 = 1.0,\ V_2 = 1.0,\ L_1 = 1.0,\ L_2 = 1.0,\ d = 4.0,\ m = 1.0$

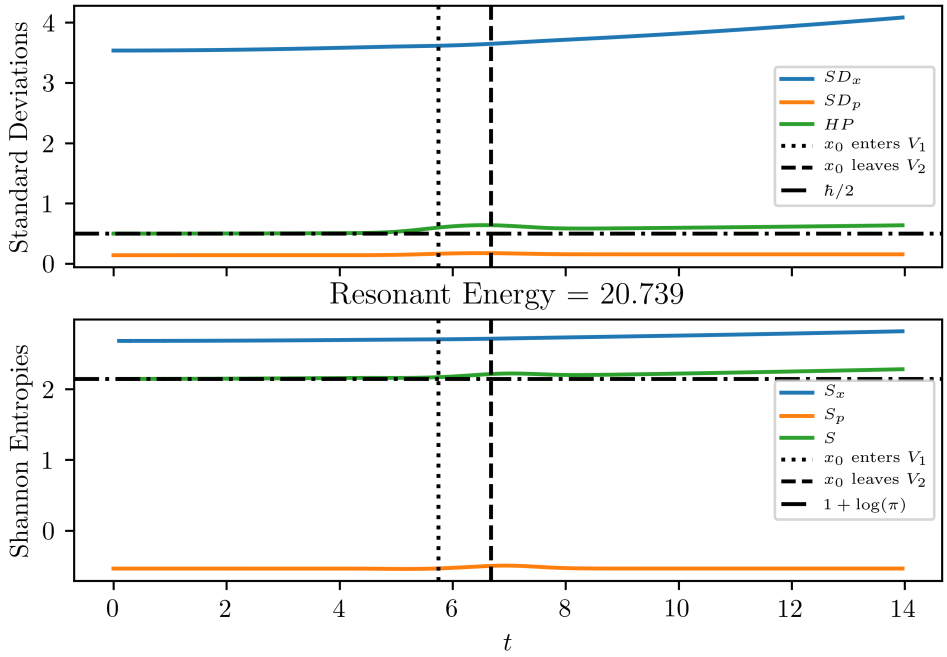


Fig. 4.34 – Uncertainties of wavelet through the Symmetric Double Barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.

Figure 4.34 shows the standard deviations (a) and Shannon entropies (b) of the Gaussian wavelet through the symmetric double barrier with $E = \frac{(2\pi)^2}{2} + 1$ for $d = L_1 = L_2 = U_1 = U_2 = 1.0$.
(a) σ_x and the product remain almost constant, the latter very close to the lower bound.
(b) S_x and the sum remain almost constant, the latter again very close to the lower bound.

4.3.2 Error Analysis

Mass of the particle $M = 1.0$
Lengths of the barriers $L_1 = L_2 = 4.0$
Heights of the barriers $U_1 = U_2 = 1.0$

A total of N samples is taken with spacing Δx for the range of integration in x which is symmetric about 0.

For k, again N samples are taken symmetrically about 0 but with spacing $\Delta p = \frac{2\pi}{N\Delta x}$

The position and momentum norms of the wavelet are calculated for the time instant of the fifth frame post-interaction, so the time varies for different energies.

Table 4.4 shows the error analysis of the Gaussian wavepacket through the symmetric double barrier. $N = 2^{11}, \Delta x = 0.1$ are chosen.

Tab. 4.4 – Error Analysis of Gaussian Wavepacket through the Symmetric Double Barrier

E	$\int \lvert\psi(x)\rvert^2 dx$	$\int \lvert\psi'(k)\rvert^2 dk$
3.0	1.000000000000089	0.9999999999997371
$\frac{(\pi)^2}{2} + 1$	1.0000000000000606	1.0000000000000022
9.7	1.0000000000000364	0.9999999999999795
13.4	1.0	0.9999999999999434
$\frac{(2\pi)^2}{2} + 1$	1.0000000000000406	0.999999999999984

4.4 Conclusion

Stationary wavefunctions of the finite well:

- The stationary wavefunctions are sinusoidal in nature, mostly with multiple peaks. This makes Shannon entropy a more stable choice of localization. This is partly offset for the position space since the distribution there is almost entirely confined to the well. This causes both σ_x and S_x to remain comparatively low. However, in the momentum space, the distribution is not as localized, causing σ_p to blow up. S_k, however, remains stable.

- Despite the above, there is an overwhelming trend in the momentum space of anti-symmetric bound states being a lot more localized than symmetric bound states, so much so that $n = 2$ for all U almost saturate the lower bounds for both Heisenberg product and entropy sum.

- Discounting the above, there is a general trend of gradual delocalization in both spaces (considering symmetric and anti-symmetric states separately for momentum) with increasing quantum number. This makes more sense since higher states have more spread towards the extrema and consist of more lobes.

Stationary wavefunctions of the finite barrier:

- The stationary wavefunctions are sinusoidal in nature, thus again Shannon entropy is a more stable choice of localization. In this case, it is partly offset for the momentum space since the distribution there comprises of a Dirac delta or comb function. This causes both σ_p and S_k to remain comparatively low. However, in the position space, the distribution is delocalized, causing σ_x to blow up. S_x, however, remains stable.

- The resonances lie very close to both σ_x and S_x maxima. But due to the above unreliability of σ_x, the peaks are very broad and hard to distinguish. In contrast, the S_x peaks are distinguishable up to the first two states with sufficient reliability.

- Through momentum uncertainties, only the first resonance can be identified with any reliability as a σ_p and a S_k minima. Resonances beyond are not distinguishable in σ_p and present too broad anti-peaks in S_k.

Gaussian wavelet propagation through the finite barrier and the symmetric double barrier:

- The Gaussian function is a good fit for standard deviation and not as much for Shannon entropy. This is because of the single peak as well as the fact that standard deviation is more sensitive to outliers, essential to studying the minute deformations in the Gaussian wavelet.
- For resonances, the wavelet shows minimal deformation while passing through the barrier(s), as compared to the non-resonant energies. The low non-resonant energies also show a noticeable reflected component.
- While Shannon entropies remain uneventfully low, σ_x and consequently, the Heisenberg product increase significantly post-interaction for non-resonant energies. However, for resonances, they remain rather low in comparison. This makes standard deviation an excellent indicator of resonance.

4.5 Future Avenues

In this work, the uncertainty measures of Shannon entropy and standard deviation are found to sufficiently characterize resonant phenomena in the time-independent and the time-dependent settings, respectively, in the case of the finite rectangular barrier for the former and the finite as well as the double barriers for the latter. Additionally, both measures are also found to reliably ascertain the parity of bound states in the case of the finite well.

It is only natural to want to test if the above extends to related potentials. The time-independent setting for the double barrier remains to be analyzed. The resonances in the continuum above the finite well can also be studied through these uncertainty measures. Thereafter, one can move onto the double well and then the general cases of n-well and the n-barrier. Finally, for a more accurate portrayal of a real quantum system, the cases of smoothly varying potentials, both positive and negative, can be considered.

Uncertainty measures and their bounds represent fundamental limitations of our possible knowledge of the quantum world. In this light, the above observations have important theoretical implications about the physical nature of the model quantum systems in study. Moreover, since these model quantum systems closely resemble their real counterparts in properties and find multifarious modern applications, said revelations may extend to real quantum systems. Further research is crucial to understanding the significance of the observations in this work and the overall scope of their impact.

Appendix

Listing 4.1 – This is a Python program for solving the input probabilities of a 6-sided biased dice with mean 3:

```
from matplotlib import pyplot as pl
import numpy as np
from scipy.optimize import minimize
import itertools

# declaring parameters

num_sides = 6
mean = 3

# constraint functions

def f(x):
    return - x*np.log(x)

def sum_form(p):
    s = 0
    for pi in p:
        s += - f(pi)
    return s

# optimizing w.r.t. constraints

cons1 = ({'type': 'eq', 'fun': lambda p: np.sum(p) - 1.})
bnds = tuple([(0, None)]*(num_sides))
sides = np.arange(0, num_sides)+1
cons2 = ({'type': 'eq', 'fun': lambda p: sides.dot(p) - mean})
```

```
ini_guess = np.array([1./num_sides]*num_sides)
sol = minimize(sum_form, ini_guess, bounds=bnds,
        constraints=[cons1, cons2], options={'maxiter':10001})

# print and plot input probabilities

print sol.x
pl.plot(np.array(range(1,7)),np.array(sol.x),'--')
pl.scatter(np.array(range(1,7)),np.array(sol.x),color='black')
pl.axhline(y=1./6., c='k', ls='-.', label=r'Mean$=3.5$')
pl.xlabel("Side Number $n$")
pl.ylabel("Probability $p_n$ of side $n$")
pl.title(r'$N={}$, Mean$={}$'.format(num_sides,mean))
pl.legend()
pl.savefig('BD.png', dpi=600)
pl.clf()
```

Listing 4.2 – This is the master Python program defining classes that compute uncertainty measures for the time-independent as well as the time-dependent cases:

```
import numpy as np
from scipy.fftpack import fft,ifft
from scipy.integrate import simps

class INDEP(object):
    """
    computes uncertainties for analytical solutions of the time-independent Schrodinger equation
    """
    def __init__(self, x, psi_x0, m=1.0):
        """
        Parameters
        ----------
        x : array, float
            length-N array of evenly spaced spatial coordinates
        psi_x0 : array, complex
            length-N array of the stationary wave function
        m : float
            particle mass
        """
```

```
        # Validation of array inputs
        self.x, psi_x0 = map(np.asarray, (x, psi_x0))
        N = self.x.size
        assert self.x.shape == (N,)
        assert psi_x0.shape == (N,)

        # Set internal parameters
        self.m = m
        self.N = len(x)
        self.dx = self.x[1] - self.x[0]
        self.dk = 2 * np.pi / (self.N * self.dx)
        self.k = self.dk * (np.arange(self.N) - 0.5 * self.N)
        self.psi_x = psi_x0
        self.hbar = 1.0

        self.compute_k_from_x()

        # attributes used for calculations
        self.psi_x_line = None
        self.psi_k_line = None
        self.D_x_line=None
        self.D_k_line=None

        self.S_x_line = []
        self.S_k_line = []
        self.S_line = []

        self.SD_x_line = []
        self.SD_p_line = []
        self.HP_line = []

        self.calc()

    def _set_psi_x(self, psi_x):
        self.psi_mod_x = (psi_x * np.exp(-1j * self.k[0] * self.x)
                          * self.dx / np.sqrt(2 * np.pi))
    def _get_psi_x(self):
        return (self.psi_mod_x * np.exp(1j * self.k[0] * self.x)
                * np.sqrt(2 * np.pi) / self.dx)
```

```
    def _set_psi_k(self, psi_k):
        self.psi_mod_k = psi_k * np.exp(1j * self.x[0]
                                        * self.dk * np.arange(self.N))

    def _get_psi_k(self):
        return self.psi_mod_k * np.exp(-1j * self.x[0] *
                                        self.dk * np.arange(self.N))

    psi_x = property(_get_psi_x, _set_psi_x)
    psi_k = property(_get_psi_k, _set_psi_k)

    def compute_k_from_x(self):
        self.psi_mod_k = fft(self.psi_mod_x)

    def calc(self):

        # attributes used for densities

        self.D_x_line = np.absolute(self.psi_x)*np.absolute(self.psi_x)
        self.D_k_line = np.absolute(self.psi_k)*np.absolute(self.psi_k)

        # attributes used for shannon entropy

        self.S_x=simps(-abs(self.psi_x)*abs(self.psi_x)*np.log(abs(self.psi_x)*abs(self.psi_x)),self.x)
        self.S_k=simps(-abs(self.psi_k)*abs(self.psi_k)*np.log(abs(self.psi_k)*abs(self.psi_k)),self.k)

        # attributes used for standard deviation

        E_x=simps(abs(self.psi_x)*abs(self.psi_x)*self.x,self.x)
        E_x2=simps(abs(self.psi_x)*abs(self.psi_x)*self.x*self.x,self.x)
        self.SD_x=(E_x2-(E_x)**2.0)**0.5
        E_p=self.hbar*simps(abs(self.psi_k)*abs(self.psi_k)*self.k,self.k)
        E_p2=self.hbar*self.hbar*simps(abs(self.psi_k)*abs(self.psi_k)*self.k*self.k,self.k)
        self.SD_p=(E_p2-(E_p)**2.0)**0.5

##############################################################################

class DEP(object):
```

```
"""
implements a numerical solution of the time-dependent Schrodinger equation
for an arbitrary potential and computes uncertainties
"""
def __init__(self, x, psi_x0, V_x, v0, m=1.0):
    """
    Parameters
    ----------
    x : array, float
        length-N array of evenly spaced spatial coordinates
    psi_x0 : array, complex
        length-N array of the initial wave function
    V_x : array, float
        length-N array giving the potential at each x
    m : float
        particle mass
    v0 : float
            wavepacket velocity
    """
    # validation of array inputs
    self.x, psi_x0, self.V_x = map(np.asarray, (x, psi_x0, V_x))
    N = self.x.size
    assert self.x.shape == (N,)
    assert psi_x0.shape == (N,)
    assert self.V_x.shape == (N,)

    # set internal parameters
    self.m = m
    self.v = v0
    self.t = 0.0
    self.dt_ = None
    self.N = len(x)
    self.dx = self.x[1] - self.x[0]
    self.dk = 2 * np.pi / (self.N * self.dx)
    self.k = self.dk * (np.arange(self.N) - 0.5 * self.N)
    self.b = np.ones(N,)
    self.psi_x = psi_x0
    self.hbar = 1.0
```

```
        self.compute_k_from_x()

        # variables which hold steps in evolution of the wavelet
        self.x_evolve_half = None
        self.x_evolve = None
        self.k_evolve = None

        # attributes used for calculations
        self.psi_x_line = None
        self.psi_k_line = None
        self.V_x_line = None
        self.D_x_line=None
        self.D_k_line=None

        self.S_x_line = []
        self.S_k_line = []
        self.S_line = []

        self.SD_x_line = []
        self.SD_p_line = []
        self.HP_line = []

        self.x_arr=[]
        self.t_arr=[]

        self.compute_density()

    def _set_psi_x(self, psi_x):
        self.psi_mod_x = (psi_x * np.exp(-1j * self.k[0] * self.x)
                          * self.dx / np.sqrt(2 * np.pi))
    def _get_psi_x(self):
        return (self.psi_mod_x * np.exp(1j * self.k[0] * self.x)
                * np.sqrt(2 * np.pi) / self.dx)

    def _set_psi_k(self, psi_k):
        self.psi_mod_k = psi_k * np.exp(1j * self.x[0]
                                        * self.dk * np.arange(self.N))

    def _get_psi_k(self):
```

```
        return self.psi_mod_k * np.exp(-1j * self.x[0] *
                                       self.dk * np.arange(self.N))

    def _get_dt(self):
        return self.dt_

    def _set_dt(self, dt):
        if dt != self.dt_:
            self.dt_ = dt
            self.x_evolve_half = self.b*np.exp(-0.5 * 1j * self.V_x
                                               / self.hbar * dt )
            self.x_evolve = self.x_evolve_half * self.x_evolve_half
            self.k_evolve = np.exp(-0.5 * 1j * (self.k * self.k) * dt * self.hbar /
                                   self.m )

    psi_x = property(_get_psi_x, _set_psi_x)
    psi_k = property(_get_psi_k, _set_psi_k)
    dt = property(_get_dt, _set_dt)

    def compute_density(self):
        self.D_x_line = abs(self.psi_x)*abs(self.psi_x)
        self.D_k_line = abs(self.psi_k)*abs(self.psi_k)

    def compute_k_from_x(self):
        self.psi_mod_k = fft(self.psi_mod_x)

    def compute_x_from_k(self):
        self.psi_mod_x = ifft(self.psi_mod_k)


    def time_step(self, dt, Nsteps = 1):
        """
        performs discrete time evolution

        Parameters
        ----------
        dt : float
            time interval of integration
        Nsteps : float, optional
```

```
        Number of intervals to compute.
        Total change in time = dt * Nsteps
    """
    self.dt = dt
    if Nsteps > 0:
        self.psi_mod_x *= self.x_evolve_half
    for i in xrange(Nsteps - 1):
        self.compute_k_from_x()
        self.psi_mod_k *= self.k_evolve
        self.compute_x_from_k()
        self.psi_mod_x *= self.x_evolve

    self.compute_k_from_x()
    self.psi_mod_k *= self.k_evolve

    self.compute_x_from_k()
    self.psi_mod_x *= self.x_evolve_half

    self.compute_k_from_x()
    self.compute_density()

    # attributes used for shannon entropy

    S_x=simps(-abs(self.psi_x)*abs(self.psi_x)*np.log(abs(self.psi_x)*abs(self.psi_x)),self.x)
    S_k=simps(-abs(self.psi_k)*abs(self.psi_k)*np.log(abs(self.psi_k)*abs(self.psi_k)),self.k)
    self.S_x_line.append(S_x)
    self.S_k_line.append(S_k)
    self.S_line.append(S_x+S_k)

    # attributes used for standard deviation

    E_x=simps(abs(self.psi_x)*abs(self.psi_x)*self.x,self.x)
    E_x2=simps(abs(self.psi_x)*abs(self.psi_x)*self.x*self.x,self.x)
    SD_x=(E_x2-(E_x)**2.0)**0.5
    E_p=self.hbar*simps(abs(self.psi_k)*abs(self.psi_k)*self.k,self.k)
    E_p2=self.hbar*self.hbar*simps(abs(self.psi_k)*abs(self.psi_k)*self.k*self.k,self.k)
    SD_p=(E_p2-(E_p)**2.0)**0.5
    self.SD_x_line.append(SD_x)
    self.SD_p_line.append(SD_p)
```

```
            self.HP_line.append(SD_x*SD_p)

            self.x_arr.append(self.t*self.v)
            self.t_arr.append(self.t)
            self.t += dt * Nsteps

##############################################################################
# Utility functions

def norm(wave_fn,x):
    """
    returns norm of a wave function

    Parameters
    ----------
    wave_fn : array
            length-N array of the wavefunction
    x : array
            length-N array w.r.t. which norm is calculated
    """
    return np.sqrt(simps(np.absolute(wave_fn)**2.,x))

def normalize(wave_fn,x):
    """
    returns normalized wave function

    Parameters
    ----------
    wave_fn : array
            length-N array of the wavefunction
    x : array
            length-N array w.r.t. which norm is calculated
    """
    return wave_fn/np.sqrt(simps(np.absolute(wave_fn)**2.,x))

def gauss_x(x, a, x0, k0):
    """
    a gaussian wave packet of width a, centered at x0, with mean momentum k0
    """
```

```
    return ((a * np.sqrt(np.pi)) ** (-0.5)
            * np.exp(-0.5 * ((x - x0) * 1. / a) ** 2 + 1j * x * k0))

def theta(x):
    """
    returns 0 if x<=0, and 1 if x>0
    """
    x = np.asarray(x)
    y = np.zeros(x.shape)
    y[x > 0] = 1.0
    return y


def well(x, a, w, h):
    """
    a finite well of width w and height h, starting from a
    """
    return h-h*(theta(x-a)-theta(x-a-w))

def barrier(x, a, w, h):
    """
    a rectangular barrier of width w and height h, starting from a
    """
    return h*(theta(x-a)-theta(x-a-w))


def double_barrier(x, a, d, w1, w2, h1, h2):
    """
    two rectangulars barrier of widths w1 and w2, and heights h1 and h2,
    with separation d, starting from a
    """
    return h1*(theta(x-a)-theta(x-a-w1))+h2*(theta(x-a-w1-d)-theta(x-a-w1-d-w2))
```

Listing 4.3 – This is a Python program for plotting the stationary densities and uncertainty measures of a particle in a finite well:

```
from matplotlib import pyplot as pl
from matplotlib import rc
rc('font',**{'family':'Latin␣Modern␣Roman␣Demi','sans-serif':['Helvetica']})
rc('text', usetex=True)
```

```
pl.rc('legend', fontsize=8)
pl.rcParams['figure.figsize']=[8,9]
import numpy as np
from cmath import exp, sin, cos
from operator import add
from master import INDEP,norm,normalize,well

##################################################################
# specify constants

hbar = 1.0 # planck's constant
m = 0.5 # particle mass

# specify range in x coordinate

N = 2 ** 11
dx = 0.02
x = dx * (np.arange(N) - 0.5 * N)

# specify potential

L0 = 2.0
V0 = 65.0
a = -L0/2.
V_x = well(x,a,L0,V0)

# determine bound states

def kset(m,L,U):
        x = np.linspace((10)**(-7), np.sqrt(U)-(10)**(-7),N)
        f = np.array([np.sqrt(2.*m*U-i**2.) for i in x])
        g = np.array([i*np.tan(i*L/2.) for i in x])
        h = np.array([-i/np.tan(i*L/2.) for i in x])
        idxs = np.argwhere(np.diff(np.sign(f-g))).flatten()
        idxa = np.argwhere(np.diff(np.sign(f-h))).flatten()
        idks = []
        for j in range(len(idxs)):
                if j%2 == 0:
                        idks.append(idxs[j])
```

```
        idka = []
        for j in range(len(idxa)):
                if j%2 == 0:
                        idka.append(idxa[j])
        ks = []
        for i in idks:
                ks.append(x[i])
        ka = []
        for i in idka:
                ka.append(x[i])
        kt = []
        for k in ks:
                kt.append(k)
        kt.extend(ka)
        kt.sort()
        return ks,ka

def ks(m,L,U):
        ks,ka = kset(m,L,U)
        return np.array(ks)

def ka(m,L,U):
        ks,ka = kset(m,L,U)
        return np.array(ka)

def kt(m,L,U):
        ks,ka = kset(m,L,U)
        kt = []
        for k in ks:
                kt.append(k)
        kt.extend(ka)
        kt.sort()
        return np.array(kt)

# specify stationary state wavefunctions

def psi_s(m,L,U,k):
        alpha = np.sqrt(2.*m*U-k**2.)
        def f1(x):
```

```
                return exp(alpha*L/2.)*cos(k*L/2.)*exp(alpha*x)
        def f2(x):
                return cos(k*x)
        def f3(x):
                return exp(alpha*L/2.)*cos(k*L/2.)*exp(-alpha*x)
        def f(x):
                        if x <= -L/2.:
                                return f1(x)
                        elif -L/2. < x < L/2.:
                                return f2(x)
                        else:
                                return f3(x)
        return normalize([f(i) for i in x],x)

def psi_a(m,L,U,k):
        alpha = np.sqrt(2.*m*U-k**2.)
        def f1(x):
                return -exp(alpha*L/2.)*sin(k*L/2.)*exp(alpha*x)
        def f2(x):
                return sin(k*x)
        def f3(x):
                return exp(alpha*L/2.)*sin(k*L/2.)*exp(-alpha*x)
        def f(x):
                        if x <= -L/2.:
                                return f1(x)
                        elif -L/2. < x < L/2.:
                                return f2(x)
                        else:
                                return f3(x)
        return normalize([f(i) for i in x],x)

# entropy of bound states

def Sx(m,L,U):
        Sx = []
        for k in kt(m,L,U):
                if k in ks(m,L,U):
                        S = INDEP(x=x,psi_x0=psi_s(m,L0,V0,k),m=m)
                        Sx.append(S.S_x)
```

```
                else:
                        S = INDEP(x=x,psi_x0=psi_a(m,L0,V0,k),m=m)
                        Sx.append(S.S_x)
        return np.array(Sx)

def Sk(m,L,U):
        Sk = []
        for k in kt(m,L,U):
                if k in ks(m,L,U):
                        S = INDEP(x=x,psi_x0=psi_s(m,L0,V0,k),m=m)
                        Sk.append(S.S_k)
                else:
                        S = INDEP(x=x,psi_x0=psi_a(m,L0,V0,k),m=m)
                        Sk.append(S.S_k)
        return np.array(Sk)

# Standard Deviation of bound states

def SDx(m,L,U):
        SDx = []
        for k in kt(m,L,U):
                if k in ks(m,L,U):
                        S = INDEP(x=x,psi_x0=psi_s(m,L0,V0,k),m=m)
                        SDx.append(S.SD_x)
                else:
                        S = INDEP(x=x,psi_x0=psi_a(m,L0,V0,k),m=m)
                        SDx.append(S.SD_x)
        return np.array(SDx)

def SDp(m,L,U):
        SDp = []
        for k in kt(m,L,U):
                if k in ks(m,L,U):
                        S = INDEP(x=x,psi_x0=psi_s(m,L0,V0,k),m=m)
                        SDp.append(S.SD_p)
                else:
                        S = INDEP(x=x,psi_x0=psi_a(m,L0,V0,k),m=m)
                        SDp.append(S.SD_p)
        return np.array(SDp)
```

```
##########################################################################
# lists for plotting

Ulist = [5.,15.,25.,40.,65.]

# error analysis
print 'Computing␣Norms...'

for k in ks(m,L0,V0):
        S = INDEP(x=x,psi_x0=psi_s(m,L0,V0,k),m=m)
        print norm(S.psi_x,S.x),norm(S.psi_k,S.k)

for k in ka(m,L0,V0):
        S = INDEP(x=x,psi_x0=psi_a(m,L0,V0,k),m=m)
        print norm(S.psi_x,S.x),norm(S.psi_k,S.k)

# plot potential
print 'Plotting␣Potential...'

fig = pl.figure()
xaxis=np.linspace(-N*dx/2,N*dx/2,8*N)
pl.xlim(-L0/2.-0.5,L0/2.+0.5)
pl.plot(xaxis,well(xaxis,a,L0,V0),color='black')
pl.xlabel(r"$x$", fontweight ='bold')
pl.ylabel(r"$V_x$", fontweight ='bold')
pl.suptitle(r'$V_0={}$,␣$L_0={}$'.format(V0,L0))
pl.title(r'Finite␣Well␣Potential',fontweight='bold')
pl.legend()
pl.savefig('plots/well/wV.png', dpi=600)
pl.clf()

# plot densities
print 'Plotting␣Densities...'

fig, (ax31, ax32) = pl.subplots(2)
fig.suptitle(r'$V_0={}$,␣$L_0={}$,␣$m={}$,␣$n={}$'.format(V0,L0,m,[(1+2*r)
for r in range(len(ks(m,L0,V0)))]))
ax31.set_xlim(-L0/2.-0.5,L0/2.+0.5)
```

```
ax31.set_ylim(0.,1.0)
ax32.set_xlim(−10.,10.)
for k in ks(m,L0,V0):
        S = INDEP(x=x,psi_x0=psi_s(m,L0,V0,k),m=m)
        ax31.plot(np.array(S.x),np.array(S.D_x_line),label='k'+'='+str(np.round(k,3)),alpha=0.7)
        ax32.plot(np.array(S.k),np.array(S.D_k_line),label='k'+'='+str(np.round(k,3)),alpha=0.7)
ax31.plot(np.array(S.x),np.array(V_x),color='black',label=r'$V_x$')
ax31.set_xlabel(r"$x$")
ax31.set_ylabel(r"$|\psi(x)|^2$")
ax31.legend(fontsize=8)
ax32.set_xlabel(r"$k$")
ax32.set_ylabel(r"$|\psi(k)|^2$")
ax32.legend(fontsize=8)
pl.title(r'Symmetric␣Position␣and␣Momentum␣Densities',fontweight='bold')
pl.savefig('plots/well/wDs.png', dpi=600)
pl.clf()

fig, (ax01, ax02) = pl.subplots(2)
fig.suptitle(r'$V_0={}$,␣$L_0={}$,␣$m={}$,␣$n={}$'.format(V0,L0,m,[(2+2*r)
for r in range(len(ka(m,L0,V0)))]))
ax01.set_xlim(−L0/2.−0.5,L0/2.+0.5)
ax01.set_ylim(0.,1.0)
ax02.set_xlim(−10,10)
for k in ka(m,L0,V0):
        S = INDEP(x=x,psi_x0=psi_a(m,L0,V0,k),m=m)
        ax01.plot(np.array(S.x),np.array(S.D_x_line),label='k'+'='+str(np.round(k,3)),alpha=0.7)
        ax02.plot(np.array(S.k),np.array(S.D_k_line),label='k'+'='+str(np.round(k,3)),alpha=0.7)
ax01.plot(np.array(S.x),np.array(V_x),color='black',label=r'$V_x$')
ax01.set_xlabel(r"$x$")
ax01.set_ylabel(r"$|\psi(x)|^2$")
ax01.legend(fontsize=8)
ax02.set_xlabel(r"$k$")
ax02.set_ylabel(r"$|\psi(k)|^2$")
ax02.legend(fontsize=8)
pl.title(r'Anti−symmetric␣Position␣and␣Momentum␣Densities',fontweight='bold')
pl.savefig('plots/well/wDa.png', dpi=600)
pl.clf()

# plotting bound states
```

```
print 'Plotting Bound States...'

for U in Ulist:
        xa = np.linspace((10)**(-7),np.sqrt(2*m*U)-(10)**(-7),N)
        y = []
        for x1 in xa:
                y.append(np.sqrt(2*m*U-x1**2))
        pl.plot(xa,y,'--',label='U'+'='+str(U),alpha=0.5)
        pl.scatter(kt(m,L0,U),np.array([np.sqrt(2.*m*U-k**2.) for k in kt(m,L0,U)]))
xb = np.linspace((10)**(-7),np.sqrt(Ulist[-1])-(10)**(-7),N)
threshold = Ulist[-1]+(10)**(-7)
f = xb*np.tan(xb*L0/2.)
g = -xb/np.tan(xb*L0/2.)
f[f>=threshold] = np.inf
f[f<=-threshold] = np.inf
g[g>=threshold] = np.inf
g[g<=-threshold] = np.inf
pl.plot(xb,f,label = 'sym',alpha=0.7)
pl.plot(xb,g,label = 'anti-sym',alpha=0.7)
pl.xlim(0,np.sqrt(Ulist[-1])+(10)**(-7))
pl.ylim(0,np.sqrt(Ulist[-1])+(10)**(-7))
pl.xlabel(r"$k$", fontweight ='bold')
pl.ylabel('$\\alpha$', fontweight ='bold')
pl.title(r'Bound States',fontweight='bold')
pl.suptitle(r'$L_0={}$, $m={}$'.format(L0,m))
pl.legend()
pl.savefig('plots/well/wBS.png', dpi=600)
pl.clf()

# plot entropies
print 'Plotting Entropies...'

fig, (ax11, ax12, ax13) = pl.subplots(3,sharex=True)
fig.suptitle(r'$L_0={}$, $m={}$'.format(L0,m))
for U in Ulist:
        lvl = [(1+r) for r in range(len(kt(m,L0,U)))]
        A = Sx(m,L0,U)
        B = Sk(m,L0,U)
        C = np.array(list(map(add,list(A),list(B))))
```

```
            ax11.plot(lvl,A,'--',alpha=0.45)
            ax11.scatter(lvl,A,label='U'+'='+str(U))
            ax12.plot(lvl,B,'--',alpha=0.45)
            ax12.scatter(lvl,B,label='U'+'='+str(U))
            ax13.plot(lvl,C,'--',alpha=0.45)
            ax13.scatter(lvl,C,label='U'+'='+str(U))
ax13.axhline(y=1+np.log(np.pi), c='k', ls='-.', label=r'$1+\log(\pi)$')
ax11.set_ylabel(r"$S_x$")
ax11.legend(fontsize=8)
ax12.set_ylabel(r"$S_k$")
ax12.legend(fontsize=8)
ax13.set_ylabel(r"$S_x + S_k$")
ax13.legend(fontsize=8)
ax13.set_xlabel(r"$n$")
pl.title(r'Shannon Entropies',fontweight='bold')
pl.savefig('plots/well/wE.png', dpi=600)
pl.clf()

# plot entropy barchart
print 'Plotting Entropy Barchart...'

barWidth = 0.3
fig = pl.subplots(figsize =(12, 8))
lvl = [(1+r) for r in range(len(kt(m,L0,V0)))]
br1 = lvl
br2 = [br + barWidth for br in br1]
br3 = [br + barWidth for br in br2]
A1 = Sx(m,L0,V0)
B1 = Sk(m,L0,V0)
C1 = np.array(list(map(add,list(A1),list(B1))))
p1 = pl.bar(br1,A1, color ='blue', width = barWidth, label = r'$S_x$',alpha=0.7)
p2 = pl.bar(br2,B1, color ='green', width = barWidth, label = r'$S_k$',alpha=0.7)
p3 = pl.bar(br3,C1, color ='red', width = barWidth, label = r'$S$',alpha=0.7)
pl.axhline(y=1+np.log(np.pi), c='k', ls='-.', label=r'$1+\log(\pi)$')
pl.xlabel(r"$n$", fontweight ='bold')
pl.ylabel(r"Entropies", fontweight ='bold')
pl.suptitle(r'$L_0={}$, $m={}$, $V0={}$'.format(L0,m,V0))
pl.title(r'Shannon Entropies',fontweight='bold')
pl.xticks([r + barWidth for r in lvl],lvl)
```

```
pl.legend()
pl.savefig('plots/well/wEbar.png', dpi=600)
pl.clf()

# plot standard deviations
print 'Plotting Standard Deviations...'

fig, (ax21, ax22, ax23) = pl.subplots(3,sharex=True)
fig.suptitle(r'$L_0={}$, $m={}$'.format(L0,m))
for U in Ulist:
        lvl = [(1+r) for r in range(len(kt(m,L0,U)))]
        A = SDx(m,L0,U)
        B = SDp(m,L0,U)
        C = np.array(A*B)
        ax21.plot(lvl,A,'--',alpha=0.45)
        ax21.scatter(lvl,A,label='U'+'='+str(U))
        ax22.plot(lvl,B,'--',alpha=0.45)
        ax22.scatter(lvl,B,label='U'+'='+str(U))
        ax23.plot(lvl,C,'--',alpha=0.45)
        ax23.scatter(lvl,C,label='U'+'='+str(U))
ax23.axhline(y=hbar*0.5, c='k', ls='-.', label=r'$\hbar/2$')
ax21.set_ylabel(r"$SD_x$")
ax21.legend(fontsize=8)
ax22.set_ylabel(r"$SD_p$")
ax22.legend(fontsize=8)
ax23.set_ylabel(r"$SD_x * SD_p$")
ax23.legend(fontsize=8)
ax23.set_xlabel(r"$n$")
pl.title(r'Standard Deviations',fontweight='bold')
pl.savefig('plots/well/wSD.png', dpi=600)
pl.clf()

# plot standard deviation barchart
print 'Plotting Standard Deviation Barchart...'

barWidth = 0.3
fig = pl.subplots(figsize =(12, 8))
lvl = [(1+r) for r in range(len(kt(m,L0,V0)))]
br1 = lvl
```

```
br2 = [br + barWidth for br in br1]
br3 = [br + barWidth for br in br2]
A2 = SDx(m,L0,V0)
B2 = SDp(m,L0,V0)
C2 = np.array(A2*B2)
p1 = pl.bar(br1,A2, color ='blue', width = barWidth, label = r'$SD_x$',alpha=0.7)
p2 = pl.bar(br2,B2, color ='green', width = barWidth, label = r'$SD_p$',alpha=0.7)
p3 = pl.bar(br3,C2, color ='red', width = barWidth, label = r'$HP$',alpha=0.7)
pl.xlabel(r"$n$", fontweight ='bold')
pl.ylabel(r"Standard␣Deviations", fontweight ='bold')
pl.suptitle(r'$L_0={}$,␣$m={}$,␣$V0={}$'.format(L0,m,V0))
pl.title(r'Standard␣Deviations',fontweight='bold')
pl.xticks([r + barWidth for r in lvl],lvl)
pl.axhline(y=hbar*0.5, c='k', ls='-.', label=r'$\hbar/2$')
pl.legend()
pl.savefig('plots/well/wSDbar.png', dpi=600)
pl.clf()
```

Listing 4.4 – This is a Python program for plotting the stationary densities and uncertainty measures of a particle in a finite barrier:

```
from matplotlib import pyplot as pl
from matplotlib import rc
rc('font',**{'family':'Latin␣Modern␣Roman␣Demi','sans-serif':['Helvetica']})
rc('text', usetex=True)
pl.rc('legend', fontsize=10)
pl.rcParams['figure.figsize']=[6,4]
import numpy as np
from cmath import exp, sin, cos, sinh, cosh
from operator import add
from scipy.integrate import simps
from master import INDEP,norm,normalize,barrier

##########################################################################
# specify constants

hbar = 1.0 # planck's constant
m = 1.0 # particle mass
```

```
# specify range in x coordinate

N = 2 ** 11
dx = 0.1
x = dx * (np.arange(N) - 0.5 * N)

# specify potential

L0 = 4.0
V0 = 1.0
a = 0.0
V_x = barrier(x,a,L0,V0)

# specify stationary state wavefunctions

def psi_g(m,L,U,E):
        k = np.sqrt(2*m*E)
        alpha = np.sqrt(k**2.-2.*m*U)
        def f1(x):
                return exp(1.j*k*L)*(-1.j*(k**2. + alpha**2.)*sin(alpha*L)+2.*k*alpha*cos(alpha*L))
                *exp(1.j*k*x)/(k*alpha) + exp(1.j*k*L)*(-1.j)*(k**2. - alpha**2.)*sin(alpha*L)
                *exp(-1.j*k*x)/(k*alpha)
        def f2(x):
                return exp(1.j*k*L)*(-1.j*((k/alpha) + 1.)*sin(alpha*L)+((k/alpha) + 1.)*cos(alpha*L))
                *exp(1.j*alpha*x) + exp(1.j*k*L)*(1.j*(1.-(k/alpha))*sin(alpha*L)+(1.-(k/alpha))
                *cos(alpha*L))*exp(-1.j*alpha*x)
        def f3(x):
                return 2.*exp(1.j*k*x)
        def f(x):
                        if x <= 0.:
                                return f1(x)
                        elif 0. < x < L:
                                return f2(x)
                        else:
                                return f3(x)
        return normalize([f(i) for i in x],x)

def psi_e(m,L,U,E):
        k = np.sqrt(2*m*E)
```

```
        def f1(x):
            return exp(1.j*k*L)*(2. - 1.j*k*L)*exp(1.j*k*x) + exp(1.j*k*L)*(-1.j*k*L)
            *exp(-1.j*k*x)
        def f2(x):
            return 2.*exp(1.j*k*L)*(1. - 1.j*k*L) + 2.j*k*exp(1.j*k*L)*x
        def f3(x):
            return 2.*exp(1.j*k*x)
        def f(x):
                if x <= 0.:
                    return f1(x)
                elif 0. < x < L:
                    return f2(x)
                else:
                    return f3(x)
        return normalize([f(i) for i in x],x)

def psi_l(m,L,U,E):
        k = np.sqrt(2*m*E)
        alpha = np.sqrt(2.*m*U-k**2.)
        def f1(x):
            return exp(1.j*k*L)*((k**2. - alpha**2.)*sinh(alpha*L)+2.j*k*alpha*cosh(alpha*L))
            *exp(1.j*k*x)/(1.j*k*alpha) + exp(1.j*k*L)*(k**2. + alpha**2.)*sinh(alpha*L)
            *exp(-1.j*k*x)/(1.j*k*alpha)
        def f2(x):
            return exp(1.j*k*L)*exp(-alpha*L)*(1.+1.j*k/alpha)*exp(alpha*x) + exp(1.j*k*L)
            *exp(alpha*L)*(1.-1.j*k/alpha)*exp(-alpha*x)
        def f3(x):
            return 2.*exp(1.j*k*x)
        def f(x):
                if x <= 0.:
                    return f1(x)
                elif 0. < x < L:
                    return f2(x)
                else:
                    return f3(x)
        return normalize([f(i) for i in x],x)

def psi(m,L,U,E):
        if E > U:
```

```
                return psi_g(m,L,U,E)
        elif E == U:
                return psi_e(m,L,U,E)
        else:
                return psi_l(m,L,U,E)

# compute momentum wavefunction with delta function normalizealization

def psi_k(m,L,U,E):
        psi_x = np.array(psi(m,L,U,E))
        k0 = np.sqrt(2*m*E)
        k = list(np.linspace(-k0-0.5,-k0+0.5,N//2))
        k1 = list(-np.array(k))[::-1]
        k.extend(k1)
        k=np.array(k)
        def F(w):
                g = (1/np.sqrt(2.*np.pi))*psi_x*np.exp(-1.j*w*x)
                return simps(g,x)
        psi_k0 = [F(i) for i in k]
        return normalize(psi_k0,k)

# compute momentum entropy

def Sk_barrier(m,L,U,E):
        psi = psi_k(m,L,U,E)
        return simps(-abs(psi)*abs(psi)*np.log(abs(psi)*abs(psi)),k)

# transmission through barrier

def T(m,L,U,E):
        k = np.sqrt(2.*m*E)
        if E > U:
                alpha = np.sqrt(k**2.-2.*m*U)
                return abs((4.*(k**2.)*(alpha**2.))/(4.*(k**2.)*(alpha**2.) + (((k**2.)-(alpha**2.))
                **2.)*(sin(alpha*L))**2.))
        elif E == U:
                return abs(4./(4. + (k*L)**2))
        else:
                alpha = np.sqrt(2.*m*U-k**2.)
```

```
                return abs((4.*(k**2.)*(alpha**2.))/(4.*(k**2.)*(alpha**2.) + (((k**2.)+(alpha**2.))
                **2.)*(sinh(alpha*L))**2.))

##########################################################################
# lists for plotting

I = [hbar*hbar*((n*np.pi/L0)**2.0+2.0*m*V0)/(2.*m) for n in range(1,4)]
Elist1 = [0.5,1.,I[0],1.8,I[1],3,I[2]]
Elist2 = list(set(np.linspace(0.1,4.,2**7))|set(I))
Elist2.sort()

# error analysis
print 'Computing Norms...'

for E in Elist1:
        S = INDEP(x=x,psi_x0=psi(m,L0,V0,E),m=m)
        k0 = np.sqrt(2*m*E)
        k = list(np.linspace(-k0-0.5,-k0+0.5,N//2))
        k1 = list(-np.array(k))[::-1]
        k.extend(k1)
        k=np.array(k)
        print norm(S.psi_x,S.x),norm(psi_k(m,L0,V0,E),k)

z0,z1,z2,y1,y2=[],[],[],[],[]
for E in Elist2:
        z0.append(T(m,L0,V0,E))
        S = INDEP(x=x,psi_x0=psi(m,L0,V0,E),m=m)
        z1.append(S.S_x)
        z2.append(Sk_barrier(m,L0,V0,E))
        y1.append(S.SD_x)
        y2.append(S.SD_p)
z00 = [(1 - t) for t in z0]
z3 = list(map(add,z1,z2))
y3 = np.array(y1) * np.array(y2)

I_t,I_x,I_k,I_x1,I_k1=[],[],[],[],[]
for E in I:
        I_t.append(T(m,L0,V0,E))
        S = INDEP(x=x,psi_x0=psi(m,L0,V0,E),m=m)
```

```
        I_x.append(S.S_x)
        I_k.append(Sk_barrier(m,L0,V0,E))
        I_x1.append(S.SD_x)
        I_k1.append(S.SD_p)
I_tot = list(map(add,I_x,I_k))
I_tot1 = np.array(I_x1) * np.array(I_k1)

# plot transmittance
print 'Plotting Transmittance...'

pl.plot(Elist2,z0,'--',label = 'Transmission',alpha=0.7)
pl.plot(Elist2,z00,'--',label = 'Reflection',alpha=0.7)
pl.plot(I[0],I_t[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_t[0],3)) + ')')
pl.plot(I[1],I_t[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_t[1],3)) + ')')
pl.plot(I[2],I_t[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_t[2],3)) + ')')
pl.xlabel(r"$E$", fontweight ='bold')
pl.ylabel(r"Probability", fontweight ='bold')
pl.suptitle(r'$V_0={}$, $L_0={}$, $m={}$'.format(V0,L0,m))
pl.title(r'Transmission and Reflection Probabilitites',fontweight='bold')
pl.legend()
pl.savefig('plots/barrier/bT.png', dpi=600)
pl.clf()

pl.rcParams['figure.figsize']=[8,8]

# plot densities
print 'Plotting Densities...'

fig, (ax01, ax02) = pl.subplots(2)
fig.suptitle(r'$V_0={}$, $L_0={}$, $m={}$'.format(V0,L0,m))
ax01.set_xlim(-1.,L0+1.)
ax01.set_ylim(0.,0.023)
ax02.set_xlim(-np.sqrt(2*m*max(Elist1))-.5,np.sqrt(2*m*max(Elist1))+.5)
ax02.set_ylim(0,10.)
for E in Elist1:
        S = INDEP(x=x,psi_x0=psi(m,L0,V0,E),m=m)
        ax01.plot(np.array(x),np.array(S.D_x_line),label='E'+'='+str(np.round(E,3)),alpha=0.7)
        ax02.plot(np.array(S.k),np.array(S.D_k_line),label='E'+'='+str(np.round(E,3)),alpha=0.7)
        ax02.axvline(-np.sqrt(2*m*E), c='k', ls=':')
```

```
            ax02.axvline(np.sqrt(2*m*E),c='k', ls=':')
ax01.plot(np.array(x),np.array(V_x),color='black',label=r'$V_x$')
ax01.set_xlabel(r"$x$")
ax01.set_ylabel(r"$|\psi(x)|^2$")
ax02.set_xlabel(r"$k$")
ax02.set_ylabel(r"$|\psi(k)|^2$")
pl.title(r'Position and Momentum Densities',fontweight='bold')
pl.savefig('plots/barrier/bD.png', dpi=600)
pl.clf()

# plot entropies
print 'Plotting Entropies...'

fig, (ax11, ax12, ax13) = pl.subplots(3,sharex=True)
fig.suptitle(r'$V_0={}$, $L_0={}$, $m={}$'.format(V0,L0,m))
ax11.scatter(Elist2,z1,alpha=0.3)
ax11.plot(I[0],I_x[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_x[0],3)) + ')')
ax11.plot(I[1],I_x[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_x[1],3)) + ')')
ax11.plot(I[2],I_x[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_x[2],3)) + ')')
ax11.set_ylabel(r"$S_x$")
ax11.legend(fontsize=10)
ax12.scatter(Elist2,z2,alpha=0.3)
ax12.plot(I[0],I_k[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_k[0],3)) + ')')
ax12.plot(I[1],I_k[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_k[1],3)) + ')')
ax12.plot(I[2],I_k[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_k[2],3)) + ')')
ax12.set_ylabel(r"$S_k$")
ax12.legend(fontsize=10)
ax13.scatter(Elist2,z3,alpha=0.3)
ax13.plot(I[0],I_tot[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_tot[0],3)) + ')')
ax13.plot(I[1],I_tot[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_tot[1],3)) + ')')
ax13.plot(I[2],I_tot[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_tot[2],3)) + ')')
ax13.axhline(y=1+np.log(np.pi), c='k', ls='-.', label=r'$1+\log(\pi)$')
ax13.set_ylabel(r"$S_x + S_k$")
ax13.legend(fontsize=10)
ax13.set_xlabel(r"$E$")
pl.title(r'Entropies',fontweight='bold')
pl.savefig('plots/barrier/bE.png', dpi=600)
pl.clf()
```

```
# plot standard deviations
print 'Plotting Standard Deviations...'

fig, (ax21, ax22, ax23) = pl.subplots(3,sharex=True)
fig.suptitle(r'$V_0={}$, $L_0={}$, $m={}$'.format(V0,L0,m))
ax21.scatter(Elist2,y1,alpha=0.3)
ax21.plot(I[0],I_x1[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_x1[0],3)) + ')')
ax21.plot(I[1],I_x1[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_x1[1],3)) + ')')
ax21.plot(I[2],I_x1[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_x1[2],3)) + ')')
ax21.set_ylabel(r"$SD_x$")
ax21.legend(fontsize=10)
ax22.scatter(Elist2,y2,alpha=0.3)
ax22.plot(I[0],I_k1[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_k1[0],3)) + ')')
ax22.plot(I[1],I_k1[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_k1[1],3)) + ')')
ax22.plot(I[2],I_k1[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_k1[2],3)) + ')')
ax22.set_ylabel(r"$SD_p$")
ax22.legend(fontsize=10)
ax23.scatter(Elist2,y3,alpha=0.3)
ax23.plot(I[0],I_tot1[0],'ro', label = '(' + str(np.round(I[0],3)) + ',' + str(np.round(I_tot1[0],3)) + ')')
ax23.plot(I[1],I_tot1[1],'yo', label = '(' + str(np.round(I[1],3)) + ',' + str(np.round(I_tot1[1],3)) + ')')
ax23.plot(I[2],I_tot1[2],'go', label = '(' + str(np.round(I[2],3)) + ',' + str(np.round(I_tot1[2],3)) + ')')
ax23.axhline(y=hbar*0.5, c='k', ls='-.', label=r'$\hbar/2$')
ax23.set_ylabel(r"$SD_x * SD_p$")
ax23.legend(fontsize=10)
ax23.set_xlabel(r"$E$")
pl.title(r'Standard Deviations',fontweight='bold')
pl.savefig('plots/barrier/bSD.png', dpi=600)
pl.clf()
```

Listing 4.5 – This is a Python program for plotting the time-dependent densities and uncertainty measures of a Gaussian wavelet through a finite barrier:

```
from matplotlib import pyplot as pl
from matplotlib import animation, rc
rc('font',**{'family':'Latin Modern Roman Demi','sans-serif':['Helvetica']})
rc('text', usetex=True)
pl.rc('legend', fontsize=6)
pl.rcParams['figure.figsize']=[12,4]
import numpy as np
```

```
from scipy.integrate import simps
from master import DEP,gauss_x,barrier

##########################################################################
# create animation

# specify time steps and duration

dt = 0.01
N_steps = 50

# specify constants

hbar = 1.0 # planck's constant
m = 1.0 # particle mass

# specify range in x coordinate

N = 2 ** 11
dx = 0.1
x = dx * (np.arange(N) - 0.5 * N)

# specify potential

L0 = 1.0
V0 = 1.0
x0 = -40.
t0 = 0.0
a = 0.0
V_x = barrier(x,a,L0,V0)

# specify initial momentum and derived quantities

n=1.1
nameno='1'
E=3.0
if n%1==0:
        p0=hbar*np.sqrt((n*np.pi/L0)**2.0+2.0*m*V0)
else:
```

```
        p0=hbar*np.sqrt(2*m*E)
d = 5.
k0 = p0 / hbar
v0 = p0 / m
psi_x0 = gauss_x(x, d, x0, k0)
t_max = 2*(abs(x0)+d)/v0
steps=int(t_max/dt)
frames = int(t_max / float(N_steps * dt))

S = DEP(x=x,psi_x0=psi_x0,V_x=V_x,v0=v0,m=m)

tlist=[(-x0+a)/(2*v0),(-x0+a)/v0,(-x0+a+L0)/v0,3*(-x0+a+L0)/(2*v0)]
Ntl=len(tlist)
j=0
Dxlist=[]
Dklist=[]
snapshots=[]

#Time evolution
print 'Computing evolution...'

Ste=[[S.t,S.x,S.k,S.D_x_line,S.D_k_line]]
for i in range(steps):
    if j<Ntl:
        tj=tlist[j]
    S.time_step(dt,1)
    Ste.append([ S.t,S.x,S.k,S.D_x_line, S.D_k_line])
    if tj>=S.t-dt and tj<S.t+dt and j<Ntl:
        j+=1
        snapshots.append([S.t,S.x,S.k,S.D_x_line, S.D_k_line])

##############################################################
# set up plot
# print 'Plotting animation...'
fig = pl.figure()


title = pl.title("")
```

```
# plotting limits

xlim = (-v0*t_max/2-d, v0*t_max/2+d)
klim = (-k0-5, k0+5)

# top axes show the x-space data

ymin = 0
ymax = (Ste[0][3]).max()
ax1 = fig.add_subplot(121, xlim=xlim,
                      ylim=(ymin - 0.2 * (ymax - ymin),
                            ymax + 0.2 * (ymax - ymin)))
psi_x_line, = ax1.plot([], [], c='r', label=r'$|\psi(x)|^2$')
V_x_line, = ax1.plot([], [], c='k', label=r'$V(x)$')
center_line = ax1.axvline(0, c='k', ls=':',
                          label = r"$x_0 + v_0t$")
ax1.legend(prop=dict(size=10))
ax1.set_xlabel(r'$x$')
ax1.set_ylabel(r'$|\psi(x)|^2$')

# bottom axes show the k-space data

ymin = 0
ymax = (Ste[0][4]).max()
ax2 = fig.add_subplot(122, xlim=klim,
                      ylim=(ymin - 0.2 * (ymax - ymin),
                            ymax + 0.2 * (ymax - ymin)))
psi_k_line, = ax2.plot([], [], c='r', label=r'$|\psi(k)|^2$')

p0_line1 = ax2.axvline(-p0 / hbar, c='k', ls=':', label=r'$\pm p_0$')
p0_line2 = ax2.axvline(p0 / hbar, c='k', ls=':')
ax2.legend(prop=dict(size=10))
ax2.set_xlabel(r'$k$')
ax2.set_ylabel(r'$|\psi(k)|^2$')
ax2.yaxis.tick_right()
ax2.yaxis.set_label_position("right")

V_x_line.set_data(S.x, S.V_x)
if n%1==0:
```

```
        Etitle=r'Resonant Energy = '+str(np.round((p0**2)/(2*m),3))
else:
        Etitle=r'Non-resonant Energy = '+str(np.round((p0**2)/(2*m),3))
pl.suptitle(r'$V_0={}$, $L_0={}$, $m={}$'.format(V0,L0,m))

##########################################################################
# animate plot

def init():
    psi_x_line.set_data([], [])
    V_x_line.set_data([], [])
    center_line.set_data([], [])

    psi_k_line.set_data([], [])
    title.set_text("")
    return (psi_x_line, V_x_line, center_line, psi_k_line, title)

def animate(i):
    if i*N_steps<steps:
        t,x,k,D_x,D_k=[Ste[i*N_steps][j] for j in range(len(Ste[0]))]
    psi_x_line.set_data(x, D_x)
    V_x_line.set_data(x, S.V_x)
    center_line.set_data(2 * [x0 + t * p0 / m])
    psi_k_line.set_data(k, D_k)
    title.set_text(Etitle+r" | t = %.2f" % t)
    return (psi_x_line, V_x_line, center_line, psi_k_line, title)
# anim = animation.FuncAnimation(fig, animate, init_func=init,frames=frames, interval=30/v0,
blit=True)
# anim.save('plots/barrier/b_'+nameno+'.gif', writer='imagemagick', fps=15/v0)
pl.clf()

# error analysis
print 'Computing Norms...'

print simps(snapshots[-1][3],snapshots[-1][1]), simps(snapshots[-1][4],snapshots[-1][2])

pl.rcParams['figure.figsize']=[8,9]

# plot snapshots
```

```
print 'Plotting snapshots...'

fig=pl.figure()
fig,ax = pl.subplots(Ntl,2)
fig.suptitle(Etitle)
for i in range(len(snapshots)):
    ti,xi,ki,Dxi,Dki=snapshots[i][0],snapshots[i][1],snapshots[i][2],snapshots[i][3],snapshots[i][4]

    # plotting limits

    xlim = (-v0*t_max/2-d, v0*t_max/2+d)
    klim = (-k0-5, k0+5)

    # top axes show the x-space data

    ymin = 0
    ymax = (Dxi).max()
    ax[i,0].set_xlim(-v0*t_max/2-d, v0*t_max/2+d)
    ax[i,0].set_ylim(ymin - 0.2 * (ymax - ymin), ymax + 0.2 * (ymax - ymin))
    psi_x_line, = ax[i,0].plot(xi, Dxi, c='r', label=r'$|\psi(x)|^2$')
    V_x_line, = ax[i,0].plot([], [], c='k', label=r'$V(x)$')
    center_line = ax[i,0].axvline(0, c='k', ls=':',
                                  label = r"$x_0 + v_0t$")
    ax[i,0].set_xlabel(r'$x$')
    ax[i,0].set_ylabel(r'$|\psi(x)|^2$')
    ax[i,0].set_title(r"t=%.2f" % ti)

    # bottom axes show the k-space data

    ymin = 0
    ymax = (Dki).max()
    ax[i,1].set_xlim(-k0-5, k0+5)
    ax[i,1].set_ylim(ymin - 0.2 * (ymax - ymin),
                                    ymax + 0.2 * (ymax - ymin))
    psi_k_line, = ax[i,1].plot(ki, Dki, c='r', label=r'$|\psi(k)|^2$')
    p0_line1 = ax[i,1].axvline(-p0 / hbar, c='k', ls=':', label=r'$\pm p_0$')
    p0_line2 = ax[i,1].axvline(p0 / hbar, c='k', ls=':')
    # ax[i,1].legend(prop=dict(size=10))
    ax[i,1].set_xlabel('$k$')
```

```
        ax[i,1].set_ylabel(r'$|\psi(k)|^2$')
        ax[i,1].yaxis.tick_right()
        ax[i,1].yaxis.set_label_position("right")

        V_x_line.set_data(S.x, S.V_x)
        center_line.set_data(2 * [x0 + ti * p0 / m], [0, 1])

fig.tight_layout()
pl.savefig('plots/barrier/bD_'+nameno+'.png', dpi=600)
pl.clf()


pl.rcParams['figure.figsize']=[6,4]


# plot potential
print 'Plotting potential...'

fig = pl.figure()
xaxis=np.linspace(-N*dx/2,N*dx/2,8*N)
pl.xlim(-1.,L0+1.)
pl.plot(xaxis,barrier(xaxis,a,L0,V0),color='black')
pl.xlabel(r'$x$', fontweight ='bold')
pl.ylabel(r'$V_x$', fontweight ='bold')
pl.suptitle(r'$V_0={}$, $L_0={}$'.format(V0,L0))
pl.title(r'Finite Barrier Potential',fontweight='bold')
pl.legend()
pl.savefig('plots/barrier/bV.png', dpi=600)
pl.clf()

# plot uncertainties
print 'Plotting uncertainties...'

fig, (ax11, ax12) = pl.subplots(2, sharex=True)
fig.suptitle(r'$V_0={}$, $L_0={}$, $m={}$'.format(V0,L0,m))
ax11.plot(t0+np.array(S.t_arr),np.array(S.SD_x_line),label=r'$SD_x$')
ax11.plot(t0+np.array(S.t_arr),S.SD_p_line,label=r'$SD_p$')
ax11.plot(t0+np.array(S.t_arr),S.HP_line,label=r'$HP$')
ax11.axvline((-x0+a)/v0, c='k', ls=':', label=r'$x_0$ enters $V_0$')
```

```
ax11.axvline((-x0+a+L0)/v0, c='k', ls='--', label=r'$x_0$ leaves $V_0$')
ax11.axhline(y=hbar*0.5, c='k', ls='-.', label=r'$\hbar/2$')
ax11.set_ylabel(r'Standard Deviations')
ax11.legend(fontsize=6)
ax12.plot(t0+np.array(S.t_arr),np.array(S.S_x_line),label=r'$S_x$')
ax12.plot(t0+np.array(S.t_arr),S.S_k_line,label=r'$S_p$')
ax12.plot(t0+np.array(S.t_arr),S.S_line,label=r'$S$')
ax12.axvline((-x0+a)/v0, c='k', ls=':', label=r'$x_0$ enters $V_0$')
ax12.axvline((-x0+a+L0)/v0, c='k', ls='--', label=r'$x_0$ leaves $V_0$')
ax12.axhline(y=1+np.log(np.pi), c='k', ls='-.', label=r'$1+\log(\pi)$')
ax12.set_xlabel(r'$t$')
ax12.set_ylabel(r'Shannon Entropies')
ax12.legend(fontsize=6)
pl.title(Etitle, fontweight ='bold')
pl.savefig('plots/barrier/bU_'+nameno+'.png', dpi=600)
pl.clf()
```

Listing 4.6 – This is a Python program for plotting the time-dependent densities and uncertainty measures of a Gaussian wavelet through a symmetric double barrier:

```
from matplotlib import pyplot as pl
from matplotlib import animation, rc
rc('font',**{'family':'Latin Modern Roman Demi','sans-serif':['Helvetica']})
rc('text', usetex=True)
pl.rc('legend', fontsize=6)
pl.rcParams['figure.figsize']=[12,4]
import numpy as np
from scipy.integrate import simps
from master import DEP,gauss_x,double_barrier

##########################################################################
# create animation

# specify time steps and duration

dt = 0.01
N_steps = 50
```

```
# specify constants

hbar = 1.0 # planck's constant
m = 1.0 # particle mass

# specify range in x coordinate

N = 2 ** 11
dx = 0.1
x = dx * (np.arange(N) - 0.5 * N)

# specify potential

L0 = 1.0
V0 = 1.0
V1 = V0
V2 = V0
L1 = L0
L2 = L0
sep = 4*L0
x0 = -40.
t0 = 0.0
a = -L1-sep/2.
V_x = double_barrier(x,a,sep, L1, L2, V1, V2)

# specify initial momentum and derived quantities

n=1.1
nameno='1'
E=3.0
if n%1==0:
        p0=hbar*np.sqrt((n*np.pi/L0)**2.0+2.0*m*V0)
else:
        p0=hbar*np.sqrt(2*m*E)
d = 5.
k0 = p0 / hbar
v0 = p0 / m
psi_x0 = gauss_x(x, d, x0, k0)
t_max = 2*(abs(x0)+d)/v0
```

```
steps=int(t_max/dt)
frames = int(t_max / float(N_steps * dt))

S = DEP(x=x,psi_x0=psi_x0,V_x=V_x,v0=v0,m=m)

tlist=[(-x0+a)/(2*v0),(-x0+a)/v0,(-x0+a+L1+sep/2.)/v0,(-x0+a+L1+sep+L2)/v0,
3*(-x0+a+L1+sep+L2)/(2*v0)]
Ntl=len(tlist)
j=0
Dxlist=[]
Dklist=[]
snapshots=[]

#Time evolution
print 'Computing evolution...'

Ste=[[S.t,S.x,S.k,S.D_x_line,S.D_k_line]]
for i in range(steps):
    if j<Ntl:
        tj=tlist[j]
    S.time_step(dt,1)
    Ste.append([ S.t,S.x,S.k,S.D_x_line, S.D_k_line])
    if tj>=S.t-dt and tj<S.t+dt and j<Ntl:
        j+=1
        snapshots.append([ S.t,S.x,S.k,S.D_x_line, S.D_k_line])

##########################################################################
# set up plot
# print 'Plotting animation...'
fig = pl.figure()

# plotting limits

xlim = (-v0*t_max/2-d, v0*t_max/2+d)
klim = (-k0-5, k0+5)

# top axes show the x-space data

ymin = 0
```

```
ymax = (S.D_x_line).max()
ax1 = fig.add_subplot(211, xlim=xlim,
                      ylim=(ymin - 0.2 * (ymax - ymin),
                            ymax + 0.2 * (ymax - ymin)))
psi_x_line, = ax1.plot([], [], c='r', label=r'$|\psi(x)|^2$')
V_x_line, = ax1.plot([], [], c='k', label=r'$V(x)$')
center_line = ax1.axvline(0, c='k', ls=':',
                          label = r"$x_0 + v_0t$")

title = ax1.set_title("")
ax1.legend(prop=dict(size=12))
ax1.set_xlabel(r'$x$')
ax1.set_ylabel(r'$|\psi(x)|^2$')

# bottom axes show the k-space data

ymin = 0
ymax = (S.D_k_line).max()
ax2 = fig.add_subplot(212, xlim=klim,
                      ylim=(ymin - 0.2 * (ymax - ymin),
                            ymax + 0.2 * (ymax - ymin)))
psi_k_line, = ax2.plot([], [], c='r', label=r'$|\psi(k)|^2$')

p0_line1 = ax2.axvline(-p0 / hbar, c='k', ls=':', label=r'$\pm p_0$')
p0_line2 = ax2.axvline(p0 / hbar, c='k', ls=':')
ax2.legend(prop=dict(size=10))
ax2.set_xlabel(r'$k$')
ax2.set_ylabel(r'$|\psi(k)|^2$')

V_x_line.set_data(S.x, S.V_x)
if n%1==0:
        Etitle=r'Resonant Energy = '+str(np.round((p0**2)/(2*m),3))
else:
        Etitle=r'Non-resonant Energy = '+str(np.round((p0**2)/(2*m),3))
pl.suptitle(r'$V_1={}$, $V_2={}$, $L_1={}$, $L_2={}$, $d={}$, $m={}$'
.format(V1,V2,L1,L2,sep,m))

##########################################################################
# animate plot
```

```
def init():
    psi_x_line.set_data([], [])
    V_x_line.set_data([], [])
    center_line.set_data([], [])

    psi_k_line.set_data([], [])
    title.set_text("")
    return (psi_x_line, V_x_line, center_line, psi_k_line, title)

def animate(i):
    if S.t<t_max:
        S.time_step(dt, N_steps)
    psi_x_line.set_data(S.x, S.D_x_line)
    V_x_line.set_data(S.x, S.V_x)
    center_line.set_data(2 * [x0 + S.t * p0 / m], [0, 1])
    psi_k_line.set_data(S.k, S.D_k_line)
    title.set_text(Etitle+" | t = %.2f" % S.t)
    return (psi_x_line, V_x_line, center_line, psi_k_line, title)
# anim = animation.FuncAnimation(fig, animate, init_func=init,frames=frames, interval=30/v0,
blit=True)
# anim.save('plots/double_barrier/db_'+nameno+'.gif', writer='imagemagick', fps=15/v0)
pl.clf()

# error analysis
print 'Computing Norms...'

print simps(snapshots[-1][3],snapshots[-1][1]), simps(snapshots[-1][4],snapshots[-1][2])

pl.rcParams['figure.figsize']=[8,9]

# plot snapshots
print 'Plotting snapshots...'

fig=pl.figure()
fig,ax = pl.subplots(Ntl,2)
fig.suptitle(Etitle)
for i in range(len(snapshots)):
    ti,xi,ki,Dxi,Dki=snapshots[i][0],snapshots[i][1],snapshots[i][2],snapshots[i][3],snapshots[i][4]
```

```
# plotting limits

xlim = (−v0∗t_max/2−d, v0∗t_max/2+d)
klim = (−k0−5, k0+5)

# top axes show the x−space data

ymin = 0
ymax = (Dxi).max()
ax[i,0].set_xlim(−v0∗t_max/2−d, v0∗t_max/2+d)
ax[i,0].set_ylim(ymin − 0.2 ∗ (ymax − ymin), ymax + 0.2 ∗ (ymax − ymin))
psi_x_line, = ax[i,0].plot(xi, Dxi, c='r', label=r'$|\psi(x)|^2$')
V_x_line, = ax[i,0].plot([], [], c='k', label=r'$V(x)$')
center_line = ax[i,0].axvline(0, c='k', ls=':',
                              label = r"$x_0 + v_0t$")
ax[i,0].set_xlabel('$x$')
ax[i,0].set_ylabel(r'$|\psi(x)|^2$')
ax[i,0].set_title(r"t=%.2f" % ti)

# bottom axes show the k−space data

ymin = 0
ymax = (Dki).max()
ax[i,1].set_xlim(−k0−5, k0+5)
ax[i,1].set_ylim(ymin − 0.2 ∗ (ymax − ymin),
                                 ymax + 0.2 ∗ (ymax − ymin))
psi_k_line, = ax[i,1].plot(ki, Dki, c='r', label=r'$|\psi(k)|^2$')
p0_line1 = ax[i,1].axvline(−p0 / hbar, c='k', ls=':', label=r'$\pm p_0$')
p0_line2 = ax[i,1].axvline(p0 / hbar, c='k', ls=':')
# ax[i,1].legend(prop=dict(size=10))
ax[i,1].set_xlabel('$k$')
ax[i,1].set_ylabel(r'$|\psi(k)|^2$')
ax[i,1].yaxis.tick_right()
ax[i,1].yaxis.set_label_position("right")

V_x_line.set_data(S.x, S.V_x)
center_line.set_data(2 ∗ [x0 + ti ∗ p0 / m], [0, 1])
```

```
fig.tight_layout()
pl.savefig('plots/double_barrier/dbD_'+nameno+'.png', dpi=600)
pl.clf()


pl.rcParams['figure.figsize']=[6,4]

# plot potential
print 'Plotting␣potential...'

fig = pl.figure()
xaxis=np.linspace(-N*dx/2,N*dx/2,8*N)
pl.xlim(-L1-sep/2.-1.,L1+sep/2.+1.)
pl.plot(xaxis,double_barrier(xaxis,a,sep,L1,L2,V1,V2),color='black')
pl.xlabel(r'$x$', fontweight ='bold')
pl.ylabel(r'$V_x$', fontweight ='bold')
pl.suptitle(r'$a={}$,␣$d={}$,␣$L_1={}$,␣$L_2={}$,␣$V_1={}$,␣$V_2={}$'
.format(a,sep,L1,L2,V1,V2))
pl.title(r'Symmetric␣Double␣Barrier␣Potential',fontweight='bold')
pl.legend()
pl.savefig('plots/double_barrier/dbV.png', dpi=600)
pl.clf()

# plot uncertainties
print 'Plotting␣uncertainties...'

fig, (ax11, ax12) = pl.subplots(2, sharex=True)
fig.suptitle(r'$V_1={}$,␣$V_2={}$,␣$L_1={}$,␣$L_2={}$,␣$d={}$,␣$m={}$'
.format(V1,V2,L1,L2,sep,m))
ax11.plot(t0+np.array(S.t_arr),np.array(S.SD_x_line),label=r'$SD_x$')
ax11.plot(t0+np.array(S.t_arr),S.SD_p_line,label=r'$SD_p$')
ax11.plot(t0+np.array(S.t_arr),S.HP_line,label=r'$HP$')
ax11.axvline((-x0+a)/v0, c='k', ls=':', label=r'$x_0$␣enters␣$V_1$')
ax11.axvline((-x0+a+L1+L2+sep)/v0, c='k', ls='--', label=r'$x_0$␣leaves␣$V_2$')
ax11.axhline(y=hbar*0.5, c='k', ls='-.', label=r'$\hbar/2$')
ax11.set_ylabel(r'Standard␣Deviations')
ax11.legend(fontsize=6)
ax12.plot(t0+np.array(S.t_arr),np.array(S.S_x_line),label=r'$S_x$')
ax12.plot(t0+np.array(S.t_arr),S.S_k_line,label=r'$S_p$')
```

```
ax12.plot(t0+np.array(S.t_arr),S.S_line,label=r'$S$')
ax12.axvline((-x0+a)/v0, c='k', ls=':', label=r'$x_0$ enters $V_1$')
ax12.axvline((-x0+a+L1+L2+sep)/v0, c='k', ls='--', label=r'$x_0$ leaves $V_2$')
ax12.axhline(y=1+np.log(np.pi), c='k', ls='-.', label=r'$1+\log(\pi)$')
ax12.set_xlabel(r'$t$')
ax12.set_ylabel(r'Shannon Entropies')
ax12.legend(fontsize=6)
pl.title(Etitle, fontweight ='bold')
pl.savefig('plots/double_barrier/dbU_'+nameno+'.png', dpi=600)
```