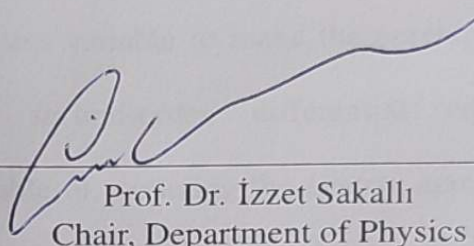One-dimensional quantum scattering from multiple
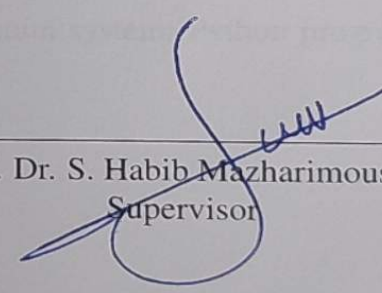Dirac delta potential: A Python-based solution

Erfan Keshavarz

Approval of the Department of Physics

I certify that this report satisfies all the requirements as a PHYS400: Graduation Project report for the degree of Bachelor of Science in Physics.
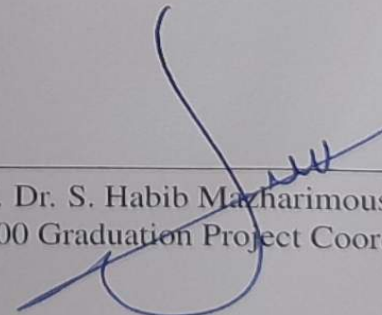
_____

Prof. Dr. İzzet Sakallı
Chair, Department of Physics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a PHYS400: Graduation Project report for the degree of Bachelor of Science in Physics.

_____

Prof. Dr. S. Habib Mazharimousavi
Supervisor

_____

Prof. Dr. S. Habib Mazharimousavi
PHYS400 Graduation Project Coordinator

_____

One-dimensional quantum scattering from multiple Dirac delta potential: A Python-based solution

Erfan Keshavarz

Submitted to the
Department of Physics
in partial fulfillment of the requirements for the degree of

Bachelor of Science
in
Physics

Eastern Mediterranean University
Spring 2023
Gazimağusa, North Cyprus

ABSTRACT

In this project, we propose a Python-based program for simulating a one-dimensional quantum system that incorporates multiple delta Dirac potentials. The primary aim of this study is to investigate the scattering phenomenon within such a system. To achieve this, we first introduce a dimensionless variable to make the corresponding Schrödinger equation a dimensionless second-order differential equation. Subsequently, we developed a program capable of modeling the system across four distinct categories. By utilizing this program, we obtain wave functions and calculate transmission and reflection coefficients for various combinations of potential strengths, distances, and the number of the Dirac delta potentials. Through comprehensive data analysis, we have the capability to create variations plots that simulate the system effectively.

Keywords: Multiple Dirac delta Potentials; quantum system; Python programming; Transmissions; and Reflection probability.

ÖZ

Bu projede, birden fazla delta Dirac potansiyelini içeren bir boyutlu kuantum sisteminin simülasyonu için Python tabanlı bir program öneriyoruz. Bu çalışmanın temel amacı, böyle bir sistemde saçılma olgusunu incelemektir. Bu amaca ulaşmak için öncelikl, ilgili Schrödinger denklemini boyutsuz ikinci dereceden diferansiyel denklem haline getirmek için boyutsuz bir değişken tanıtıyoruz. Ardından, sistemi dört ayrı kategoride modelleyebilen bir program geliştiriyoruz. Bu programı kullanarak, çeşitli potansiyel güçleri, mesafeleri ve Dirac delta potansiyellerinin sayısını içeren kombinasyonlar için dalga fonksiyonlarını elde ediyor, iletim ve yansıma katsayılarını hesaplıyoruz. Kapsamlı veri analizi sayesind sistemi etkili bir şekilde simüle eden değişken grafikler elde ediyoruz.

Anahtar Kelimeler: Çoklu Dirac delta potansiyelleri; kuantum sistemi; Python programlama; İletim; Yansıma olasılığı.

DEDICATION

I dedicate this thesis to my cherished family and Maryam. Their unyielding backing, uplifting encouragement, and solid confidence in my potential have been the driving force boosting my academic voyage.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Dirac delta potential has a profound impact on the field of science, with significant applications in various areas. For instance, the Kronig-Penny model stands out as a crucial example, as it effectively elucidates the formation of band gaps in crystal structures [1]. Additionally, notable studies have demonstrated the behavior of scattering and reflection phenomena for arbitrary potentials using delta potentials [2]. Moreover, numerous investigations have focused on the study of Dirac delta potentials through different approaches, such as the transfer matrix method, which has been employed to explore transmission resonance, threshold anomalies, and Bloch states [3–8]. Another significant line of inquiry has centered around the use of the Lippmann-Schwinger equation to investigate scattering theory for Dirac delta potentials [9, 10].

In this study, we initiate our exploration by representing the Schrödinger equation dimensionless in Section 2. Subsequently, we proceed to simulate a system consisting of multiple one-dimensional Dirac delta potentials using Python. The versatility of this program allows it to accommodate any number of potentials, as detailed in Section 3. In Section 4, our focus shifts to the modification of the code to generate regional wavefunctions. Through the application of boundary conditions at each potential point, we establish a system of equations that yields transmission and reflection coefficients in Section 5. To further enrich our analysis, we extend our code to render the system, facilitating the discussion of impurities within the system. This

leads to the plotting of various essential parameters, such as wavefunctions, transmission, and reflection probabilities. Furthermore, the significance of this research is underscored by its potential to inspire further investigations in the field of quantum mechanics.

The combination of theoretical analysis and computational simulations showcased in this study serves as a powerful model for approaching complex quantum systems, providing a stepping stone for researchers to tackle even more intricate problems.

Chapter 2

DIMENSIONLESS FORM OF THE SCHRÖDINGER

EQUATION

In this section, we consider the Schrödinger equation with a Dirac delta potential $V(x) = V_0 \delta(x - x_0)$. Furthermore, we initiate the process of making the Schrödinger equation dimensionless by introducing a new variable, $y = kx$, where k is the scaling parameter. By applying this transformation, we aim to express the equation in a dimensionless form. Subsequently, we discuss the scattering problem, in order to determine the transmission and reflection coefficients associated with the system. starting from Schrodinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} - V_0 \delta(x - x_0)\psi = E\psi \tag{2.1}$$

By substituting the new variable $y = kx$, $k^2 = \frac{2mE}{\hbar^2}$, $\tilde{V}_0 = \frac{2mV_0}{\hbar^2}$ into equation (2.1) it will simplify as follows:

$$-\frac{d^2\psi}{dy^2} - \xi\delta(y - y_0)\psi = \psi \tag{2.2}$$

where $\xi = \frac{\tilde{V}_0}{k}$. by integrating both sides of (2.2) from $y_0 - \varepsilon$ to $y_0 + \varepsilon$ then by taking the limit as ε approaches zero we obtain the following result:

$$\frac{d\psi_R}{dy}(y_o) - \frac{d\psi_L}{dy}(y_o) = -\xi\psi(y_0) \tag{2.3}$$

which equation (2.3) shows the behaviour or wavefunctions at the boundary. here

$\frac{d\psi_R}{dy}(y_o)$ represent the derivative of wavefunctions from right hand side and $\frac{d\psi_L}{dy}(y_o)$

is the derivative of wavefunction from the left-hand side. Now by considering the Schrödinger equation at the regions of $y < y_0$ and $y_0 < y$, we get the following result:

$$\psi_L = \exp(iy) + r\exp(-iy) \tag{2.4}$$

$$\psi_R = t\exp(iy) \tag{2.5}$$

In which r and t are the reflection and transmission coefficients. by applying the continuity boundary conditions presented in equations (2.4) and (2.5) at $y = y_0$, and considering equation (2.3) we can easily determine the transmission and reflection coefficients, which are expressed as follows:

$$t = \frac{2i}{\xi + 2i} \tag{2.6}$$

$$r = \frac{-\xi\exp(2iy_0)}{\xi + 2i} \tag{2.7}$$

The probability invariance is also satisfied as follows $|t|^2 + |r|^2 = 1$, where $|t|^2$ and $|r|^2$ are the probability of transmission and reflection of a particle undergoing the single Dirac delta potential

Chapter 3

PYTHON-BASED SOLUTION FOR MULTIPLE DIRAC

DELTA POTENTIALS

In this section, we propose a Python code designed to generate multiple Dirac delta potentials. The primary aim of this program is to calculate the wave functions within each region and create corresponding plots based on the associated distances. Furthermore, the program facilitates the calculation of transmission and reflection coefficients. Additionally, the program allows for the consideration of impurities within the system. At the program's outset, we import the essential libraries, such as numpy, sympy, scipy and matplotlib.pyplot for numerical calculation, symbolic and scientific computation as well as data visualization as shown in the code below:

```
import sympy as smp
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from sympy import solve,symbols,Eq,I,conjugate,expand,Abs,
    lambdify,simplify,factor
from mpl_toolkits.mplot3d
import Axes3D from scipy.interpolate
import griddata
```

To initiate the system, the first step involves establishing the initial values, namely the potential_list, and distance_list. The program itself is implemented as a user input

program, offering various options to the user. Initially, the user is presented with a choice between equal distances or non-equal distances, as well as equal potentials or non-equal potentials. Furthermore, the user is prompted to input the value of k. With these inputs at hand, we proceed to compute the ξ list. as shown in the below code: These options allow for flexibility in defining the characteristics of the generated non-linear multiple Dirac delta potentials.

```
potential_list = []
distance_list=[]
num_potential = int(input("Enter the number of potentials: "))
potential_types = input("choose between 'equal potentials' or '
    non equal potentials': ").lower()
distance_types=input("choose between 'equal distances' or 'non
    equal distances': ").lower()
if potential_types == 'non equal potentials':
    for i in range(num_potential):
        value_of_potential = float(input("Enter the value of
            potential {}: ".format(i+1)))
        potential_list.append(value_of_potential)
elif potential_types == 'equal potentials':
    value_of_potential = float(input("Enter the value of potential
        : "))
    potential_list = [value_of_potential] * num_potential
else:
    print("Invalid potential types!")
if distance_types=='non equal distances':
    for i in range(num_potential-1):
        value_of_distance=float(input("Enter the value of distance
            {}: ".format(i+1)))
        distance_list.append(value_of_distance)
```

```python
    elif distance_types=='equal distances':
        value_of_distance = float(input("Enter the value of distance:
            "))
        distance_list = [value_of_distance] * int((num_potential-1))
    else:
        print("Invalid distance types!")
def divide_list_elements(lst, divisor):
    new_lst = [num / divisor for num in lst]
    return new_lst
k = int(input("enter value of k: "))
kisi_f = divide_list_elements(potential_list,k)
```

Chapter 4

WAVE FUNCTIONS AND THE DERIVATIVES OF WAVE FUNCTIONS

Once you have selected your preferred option and set the initial values, such as ξ and distances, the code will initiate by providing you with a list of general wave functions and their corresponding derivatives as functions of y. You can observe this in the following code:

```
#Algebric approach for wave functions and the derivatives of wave
    functions
A = [smp.symbols("a{}".format(i)) for i in range(1, num_potential
    + 2)]
B = [smp.symbols("b{}".format(i)) for i in range(1, num_potential
    + 2)]
t, r,y=smp.symbols("t r y")
B[0] = r
A[0] = 1
B[num_potential] = 0
A[num_potential] = t
psi_y = [A[i] * smp.exp(1j*y) + B[i] * smp.exp(-1j* y) for i in
    range(num_potential+1)]
print("wave functions in our system is as followes:")
print(psi_y)
dpsi_y = []
for i in range(num_potential + 1):
    derivative = smp.diff(psi_y[i],y)
```

```
        dpsi_y.append(derivative)
print("derivatives of wave functions are as followes:")
print(dpsi_y)
```

After the wave functions and their corresponding derivatives have been compiled into a list, it becomes essential to apply the continuity boundary condition individually for each distinct region, taking into account whether the distances are equal or non-equal. The provided code demonstrates this particular procedure:

```
bc_eq1=[]
privious_psi_y=psi_y[0]
for element in psi_y[1:]:
    difference=element-privious_psi_y
    bc_eq1.append(difference)
    privious_psi_y=element
list_of_distances=[]
D_t=[smp.symbols("d_t{}".format(i)) for i in range(1,
    num_potential )]
if distance_types=='equal distances':
    for i in range(num_potential):
    y= y0 + n*d_t
    result= y.subs([(y0,y0_n),(n,i)])
    list_of_distances.append(result)
elif distance_types=='non equal distances':
    list_of_distances = [y0_n] + [y0_n+ smp.Add(*D_t[:i]) for i
        in range(1, num_potential)]
y = smp.symbols('y')
n = num_potential
y_symbols = smp.symbols('y1:{}'.format(n + 1))
bc_eq1_indexed = [eq.subs(y, y_sym) for eq, y_sym in zip(bc_eq1,
```

```
    y_symbols)]
bc_eq1_n=[]

for i in range(len(list_of_distances)):

    if distance_types=='equal distances':

        y_index1 = i + 1

        y_value1 = list_of_distances[i]

        bc_eq1_n.append(bc_eq1_indexed[i].subs(f'y{y_index1}',

            y_value1))

    elif distance_types=='non equal distances':

        y_index1 = i + 1

        y_value1 = list_of_distances[i]

        bc_eq1_n.append(bc_eq1_indexed[i].subs(f'y{y_index1}',

            y_value1))

print(bc_eq1_n)
```

We follow a similar procedure to design the boundary condition for the derivative of wave functions at each region, as shown in the code below:

```
bc_eq2 = []

previous_dpsi_y = dpsi_y[0]

previous_psi_y2 = psi_y[1]

for i in range(1, len(dpsi_y)):

    if potential_types == 'equal potentials':

        kisi=smp.symbols("kisi")

        current_dpsi_y = dpsi_y[i]

        current_psi_y = psi_y[i]

        difference = current_dpsi_y - previous_dpsi_y + (kisi) *
            current_psi_y

        bc_eq2.append(difference)

        previous_dpsi_y = current_dpsi_y
```

```python
                previous_psi_y2 = current_psi_y
        elif potential_types == 'non equal potentials':
            kisi_list = [smp.symbols("kisi{}".format(i)) for i in
                range(1, num_potential+1)]
            kisi=smp.symbols("kisi")
            current_dpsi_y = dpsi_y[i]
            current_psi_y = psi_y[i]
            difference = current_dpsi_y - previous_dpsi_y + kisi_list[
                i-1] * current_psi_y
            bc_eq2.append(difference)
            previous_dpsi_y = current_dpsi_y
            previous_psi_y2 = current_psi_y
print(bc_eq2)
y = smp.symbols('y')
n = num_potential
y_symbols = smp.symbols('y1:{}'.format(n + 1))
bc_eq2_indexed = [eq.subs(y, y_sym) for eq, y_sym in zip(bc_eq2,
    y_symbols)]
print(bc_eq2_indexed )
bc_eq2_n= []
for i in range(len(list_of_distances)):
    if distance_types=='equal distances':
        y_index = i + 1
        y_value = list_of_distances[i]
        bc_eq2_n.append(bc_eq2_indexed[i].subs(f'y{y_index}',
            y_value))
    elif distance_types=='non equal distances':
        y_index = i + 1
        y_value = list_of_distances[i]
        bc_eq2_n.append(bc_eq2_indexed[i].subs(f'y{y_index}',
            y_value))
```

```
print(bc_eq2_n)
```

4.1 Reflection and Transmission coefficient

Now that we have symbolically defined the boundary conditions for any variation our user may request, and we have obtained the initial values of ξ and distances, the next step is to import these values and construct a system of equations. Once the system is formed, we can proceed to solve it effectively as follows:

```
if distance_types=='non equal distances':
   new_list = [expr.subs(zip(D_t, distance_list)) for expr in
      bc_eq1_n]
elif distance_types=='equal distances':
   distance_list_sympy = [smp.sympify(element) for element in
      distance_list]
   substitutions = [(d_t, value) for value in distance_list_sympy
      ]
   new_list = [expr.subs(substitutions) for expr in bc_eq1_n]
if distance_types=='non equal distances' and  potential_types=='
   non equal potentials':
   new_list2 = [expr.subs(zip(kisi_list, kisi_f)) for expr in
      bc_eq2_n]
   new_list3 =[expr.subs(zip(D_t, distance_list)) for expr in
      new_list2]
elif distance_types=='equal distances' and  potential_types=='non
    equal potentials':
   distance_list_sympy = [smp.sympify(element) for element in
      distance_list]
   substitutions = [(d_t, value) for value in distance_list_sympy
      ]
   new_list2 = [expr.subs(substitutions) for expr in bc_eq2_n]
```

12

```python
    new_list3 = [expr.subs(zip(kisi_list, kisi_f)) for expr in
        new_list2]
elif distance_types=='non equal distances' and  potential_types==
    'equal potentials':
    potential_list_sympy = [smp.sympify(element) for element in
        kisi_f]
    substitutions = [(kisi, value) for value in
        potential_list_sympy]
    new_list2 = [expr.subs(substitutions) for expr in bc_eq2_n]
    new_list3 =[expr.subs(zip(D_t, distance_list)) for expr in
        new_list2]
elif distance_types=='equal distances' and  potential_types=='
    equal potentials':
    distance_list_sympy = [smp.sympify(element) for element in
        distance_list]
    substitutions = [(d_t, value) for value in distance_list_sympy
        ]
    new_list2 = [expr.subs(substitutions) for expr in bc_eq2_n]
    potential_list_sympy = [smp.sympify(element) for element in
        kisi_f]
    substitutions = [(kisi, value) for value in
        potential_list_sympy]
    new_list3 = [expr.subs(substitutions) for expr in new_list2]
system_of_equations = [Eq(eq, 0) for eq in (new_list + new_list3)
    ]
print(system_of_equations)
symbols_list = list(set().union(*[eq.free_symbols for eq in
    system_of_equations]))
solution = solve(system_of_equations, symbols_list)
symbol_values = {symbol: value for symbol, value in solution.
    items()}
```

```python
print("Assigned Values:")

for symbol, value in symbol_values.items():

    print(f"{symbol}: {value}")

functions = {}

for symbol, value in symbol_values.items():

    func = smp.lambdify([], value)

    functions[symbol] = func

r_value = functions[r]()

t_value=functions[t]()

ai=[]

bi=[]

for i in range(2, num_potential+1):

    a_value = functions[A[i-1]]()

    ai.append(a_value)

    b_value = functions[B[i-1]]()

    bi.append(b_value)

    print(f"a{i} value: {a_value}")

    print(f"b{i} value: {b_value}")
```

Once the solutions for each coefficient in every region have been obtained, allowing

for user-defined variations in the wave functions, and the reflection and transmission

coefficients have been evaluated, the next step is to analyze the probability invariance.

Additionally, you can assign the coefficients to the wave functions to calculate their

conjugates as well as the absolute value of the wave functions. The code below

illustrates how to perform these tasks:

```python
probability_invariance=Abs(t_value)**2+Abs(r_value)**2

print(probability_invariance)

substituted_psi_y = []
```

14

```python
for expr in psi_y:
    substituted_expr = expr.subs({'r': r_value, 't': t_value})
    for i in range(0,num_potential-1):
        substituted_expr = substituted_expr.subs({A[i+1]: ai[i],B
            [i+1]: bi[i]})
    substituted_psi_y.append(substituted_expr)
print(substituted_psi_y)
conjugate_psi_y1 = [conjugate(expr) for expr in substituted_psi_y
    ]
conjugate_psi_y = [expr.subs(conjugate(y), y) for expr in
    conjugate_psi_y1]
print(conjugate_psi_y)
psi_y_square = [expand(expr1 * expr2) for expr1, expr2 in zip(
    substituted_psi_y, conjugate_psi_y)]
abs_psi_y_square = [Abs(expr).rewrite(Abs) for expr in
    psi_y_square]
print(abs_psi_y_square)
```

Chapter 5

DATA VISUALIZATION AND DATA ANALYSIS

In this section, we will divide our code into four parts to create different plots and visualize the behavior of the system with multiple Dirac delta potentials. Firstly, we will modify the code to generate plots for the wave functions and their absolute values with respect to their region distances. Next, we will create plots for the transmission and reflection coefficients, considering both their regional distances and their corresponding energies.

5.1 Equal distances and Equal potentials

Initially, we proceed with code modification to create a plot depicting the wave function concerning the distances, assuming both the distances and potentials are equally spaced:

```
if potential_types == 'equal potentials'and distance_types == '
    equal distances':
    lambdified_psi_y_functions = [lambdify(y, func) for func in
        substituted_psi_y]
    l = num_potential -1  # Define the value of n
    range_of_graph = []  # Initialize an empty list to store the
        ranges
    start = y0_n - (10 * value_of_distance)
    end = y0_n
    range_of_graph.append((start, end))  # Add the first range
    for i in range(l):
        start = y0_n + (i * value_of_distance)
```

```python
        end = y0_n + ((i+1) * value_of_distance)

        range_of_graph.append((start, end))

end = y0_n + ((l + 10) * value_of_distance)

range_of_graph.append((y0_n + (l * value_of_distance), end))

    # Add the last range

print(range_of_graph)

plt.figure(1)

for i, func in enumerate(lambdified_psi_y_functions):

    y_vals = np.linspace(range_of_graph[i][0], range_of_graph[
        i][1], 1000)

    f_vals = np.real(func(y_vals))

    plt.plot(y_vals, f_vals, label=str(substituted_psi_y [i]))

plt.xlabel('y')

plt.ylabel('psi_y')

plt.title('Plot of wave function')

lambdified_abs_psi_y_square = [lambdify(y, func) for func in
    abs_psi_y_square]

m = num_potential - 1

range_of_graph1 = []

start1 = y0_n - (10 * value_of_distance)

end1 = y0_n

range_of_graph1.append((start1, end1))

for i in range(m):

    start1 = y0_n + (i * value_of_distance)

    end1 = y0_n + ((i + 1) * value_of_distance)

    range_of_graph1.append((start1, end1))

end1 = y0_n + ((m + 10) * value_of_distance)

range_of_graph1.append((y0_n + (m * value_of_distance), end1))

plt.figure(2)

for i, func in enumerate(lambdified_abs_psi_y_square):
```

```
        y_vals1 = np.linspace(range_of_graph1[i][0],

            range_of_graph1[i][1], 1000)

        f_vals1 = np.vectorize(func)(y_vals1)

        plt.plot(y_vals1, np.real(f_vals1), label=str(

            abs_psi_y_square[i]))

    plt.xlabel('y')

    plt.ylabel('psi_y_square')

    plt.title('Plot of abs wave function square')

    plt.grid(True)

    plt.show()
```

Let's take multiple specific examples and examine various scenarios involving different values of the variable ξ and distances

5.2 Data visualization and Data analysis for $n = 4$, $\tilde{V}_0 = 1$, $d = 1$ and $k = 1$

Once the initial values are input into the given code, the following steps are carried out Generation of Wavefunction and First Derivatives: By using sympy library, the code generates the list of wavefunctions and its first derivatives as shown below:

$$\psi_1(y) = \exp(iy) + r\exp(-iy) \tag{5.1}$$

$$\frac{d\psi_1}{dy} = i\exp(iy) - ir\exp(-iy) \tag{5.2}$$

$$\psi_2(y) = a_2\exp(iy) + b_2\exp(-iy) \tag{5.3}$$

$$\frac{d\psi_2}{dy} = ia_2\exp(iy) - ib_2\exp(-iy) \tag{5.4}$$

$$\psi_3(y) = a_3\exp(iy) + b_3\exp(-iy) \tag{5.5}$$

$$\frac{d\psi_3}{dy} = ia_3\exp(iy) - ib_3\exp(-iy) \tag{5.6}$$

$$\psi_4(y) = a_4\exp(iy) + b_4\exp(-iy) \tag{5.7}$$

18

$$\frac{d\psi_4}{dy} = ia_4 \exp(iy) - ib_4 \exp(-iy) \tag{5.8}$$

$$\psi_5(y) = t \exp(iy) \tag{5.9}$$

$$\frac{d\psi_5}{dy} = it \exp(iy) \tag{5.10}$$

Solving the System of Equations: by assigning equations from in to the boundary conditions The code proceeds to solve the system of equations to determine the coefficients, as well as the reflection and transmission coefficients as shown below:

$$a_2 = 1.00493843116683 + 0.386972112079938i \tag{5.11}$$

$$a_3 = 0.623857433460731 + 0.757078206280336i \tag{5.12}$$

$$a_4 = 0.299276671335977 + 1.047107929606i \tag{5.13}$$

$$b_2 = -0.23099420700695 - 0.39684897441359i \tag{5.14}$$

$$b_3 = -0.0530433395616452 + 0.10368547650104i \tag{5.15}$$

$$b_4 = -0.484698702505869 + 0.0476180243087344i \tag{5.16}$$

$$r = -0.226055775840124 - 0.00987686233365196i \tag{5.17}$$

$$t = -0.179421834773618 + 0.957397012219189i \tag{5.18}$$

These coefficients play a crucial role in describing how the wavefunction behaves through the system .After finding the coefficients, the code applies these values to the wavefunction to obtain the complete solution that takes into account the interactions and behaviors of the wavefunction in each region. Data visualization: Using the solutions obtained, the code generates plots that showcase the behavior of the wavefunction within the various regions. These plots visually represent how the wavefunction evolves over the distances in each region, providing valuable insights

into the physical interpretation of the system under different scenarios. The code creates additional plots that display the transmission and reflection probabilities with respect to their respective distances. These plots provide a clear visualization of how these coefficients change throughout the system, offering a deeper understanding of the wave's behavior at boundaries and interfaces. Lastly, the code generates plots that show how the transmission and reflection probabilities vary concerning the variable ξ and distances. This analysis helps to explore how different values of ξ influence the transmission and reflection phenomena, shedding light on the system's sensitivity to changes in this variable.
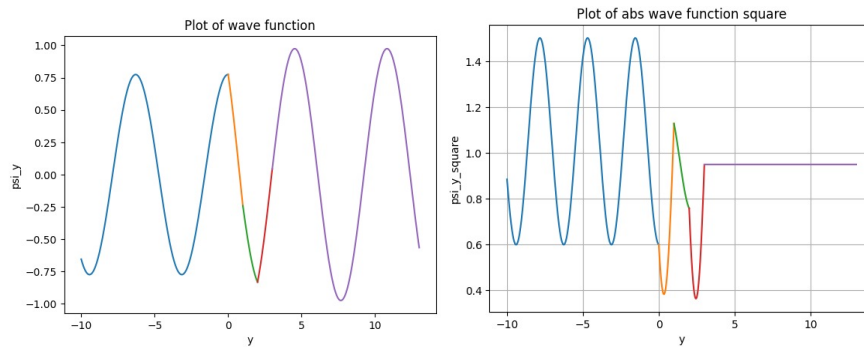
Figure 5.1: In the context of this system, the wave functions and the absolute value of wave functions that have not undergone normalization are examined concerning their variations concerning regional distances for $n = 4, \tilde{V}_0 = 1, d = 1$ and $k = 1$. In this plot, it is evident that the wave function at the boundary fulfills the expected continuity conditions also from the the absolute value of the wavefunction we can compare the probability distribution within the system.
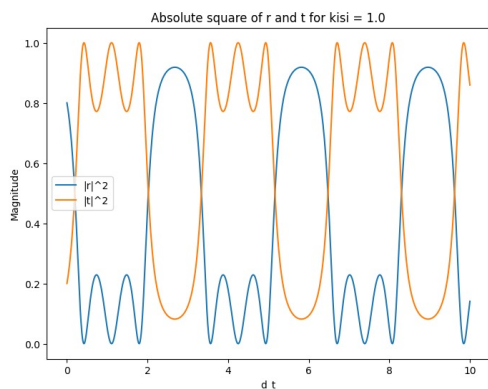
Figure 5.2: The transmission and reflection probability functions have been plotted as functions of distance within the system for $n = 4, \tilde{V}_0 = 1, d = 1$ and $k = 1$ and the plot demonstrates that the summation of probabilities always remains equal to 1.
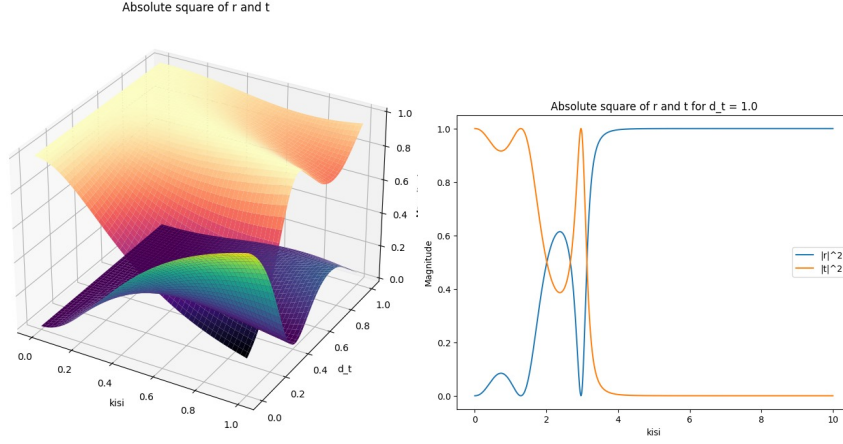
Figure 5.3: The plot of transmission and reflection probabilities as a function of ξ and distance reveals interesting observations. Specifically, for $d = 1$ and $\xi = 0$, the reflection coefficient is 0, and the transmission coefficient is 1, as expected. Furthermore, as ξ approaches 4, both the reflection and transmission probabilities tend to be zero.

Transmission and reflection coefficient: Moreover, the code has the capability to symbolically generate the solutions for the transmission and reflection coefficients, which are represented as follows:

$$t = \frac{-16}{\lambda + \eta} \tag{5.19}$$

$$r = \frac{\alpha + \beta}{\lambda + \eta} \tag{5.20}$$

$$\lambda = \xi^4(3e^{2id} - 3e^{4id} + e^{6id} - 1) + 4i\xi^3(3e^{2id} - e^{6id} - 2) \tag{5.21}$$

$$\eta = 4\xi^2(-3e^{2id} - 2e^{4id} - e^{6id} + 6) + 32i\xi - 16 \tag{5.22}$$

$$\alpha = \xi^4(-3e^{2id} + 3e^{4id} - e^{6id} + 1) + 6i\xi^3(-e^{2id} - e^{4id} + e^{6id} + 1) \tag{5.23}$$

$$\beta = 4\xi^2(-e^{2id} + e^{4id} + 3e^{6id} - 3) - 8i\xi(e^{2id} + e^{4id} + e^{6id} + 1) \tag{5.24}$$

21

5.2.1 Data visualization and Data analysis for $n = 4$, $\tilde{V}_0 = 1$, $d = 2$ and $k = 1$

Equations 5.25 and 5.26 represent reflection and transmission coefficient:

$$r = 0.471896745237825 + 0.473958619723303i \qquad (5.25)$$

$$t = -0.652402684313879 + 0.356437127869662i \qquad (5.26)$$
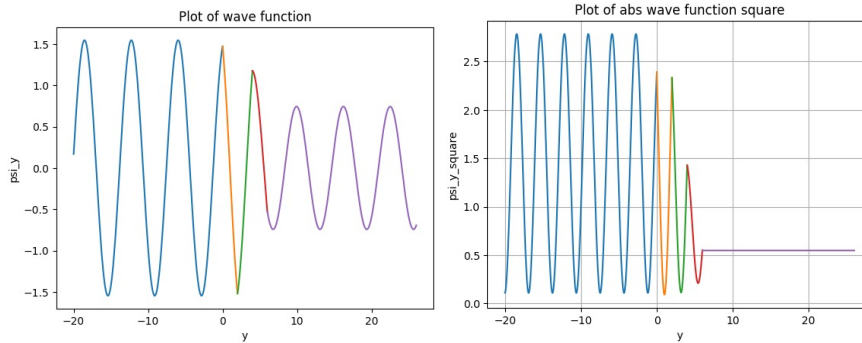
Figure 5.4: This figure illustrates the wave function and its squared absolute value. By comparing it to Figure 5.1, we observe that increasing the distance of each potential by a factor of two leads to a greater oscillation, as anticipated.
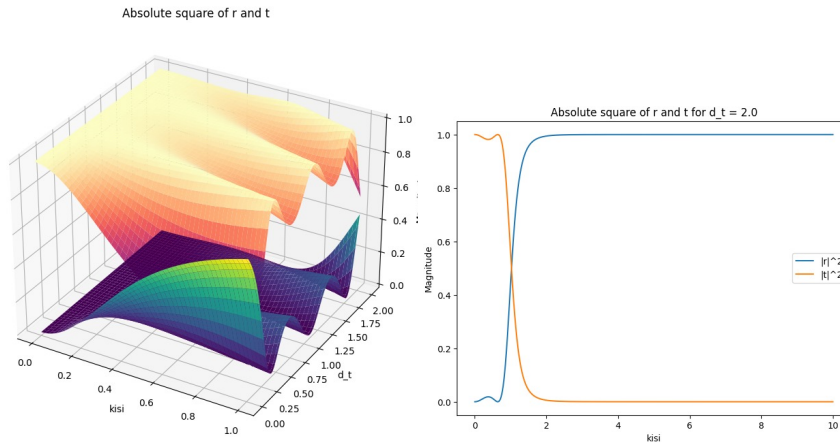
Figure 5.5: The figure displays the transmission and reflection probability as functions of distances and ξ. A comparison with Figure 5.3 reveals that when the distance is doubled, both the transmission and reflection probability decay more rapidly with an increase in ξ.

5.2.2 Data visualization and Data analysis for $n = 4$, $\tilde{V}_0 = 2$, $d = 1$ and $k = 1$

Equations 5.27 and 5.28 represent reflection and transmission coefficient :

$$r = -0.224960026676688 + 0.665143015537305i \qquad (5.27)$$

$$t = -0.635548605273626 - 0.321022936274516i \qquad (5.28)$$
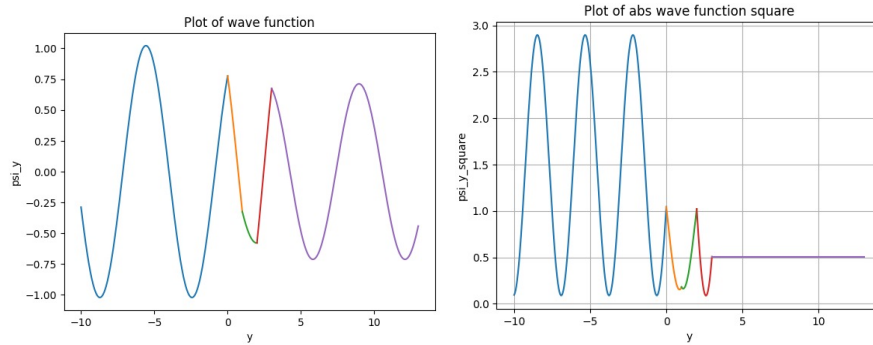
Figure 5.6: This figure shows the wave function and the absolute value of the wave function squared. By comparing it with Figure 5.1, we see that the amplitude of the absolute wave function will decrease.
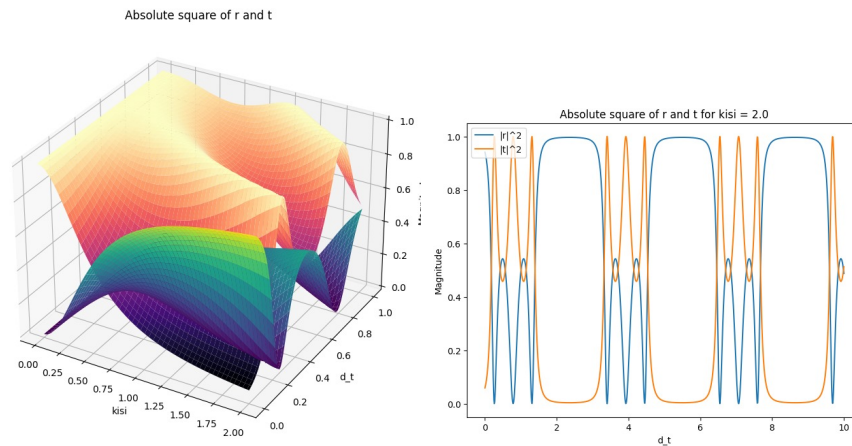
Figure 5.7: The figure depicts the transmission and reflection probabilities as functions of distances and ξ. When compared to Figure 5.2, we observe a significant increase in amplitude, such that at certain distances, the probability of reflection can either reach 1 or 0, similar to the transmission probability.

5.2.3 Data visualization and Data analysis for $n = 4$, $\tilde{V}_0 = 1$, $d = 1$, and $k = 2$

Equations 5.29 and 5.30 represent some equations:

$$r = -0.177415895619637 + 0.176012004235319i \qquad (5.29)$$

$$t = 0.578115913901493 + 0.776740216829533i \qquad (5.30)$$
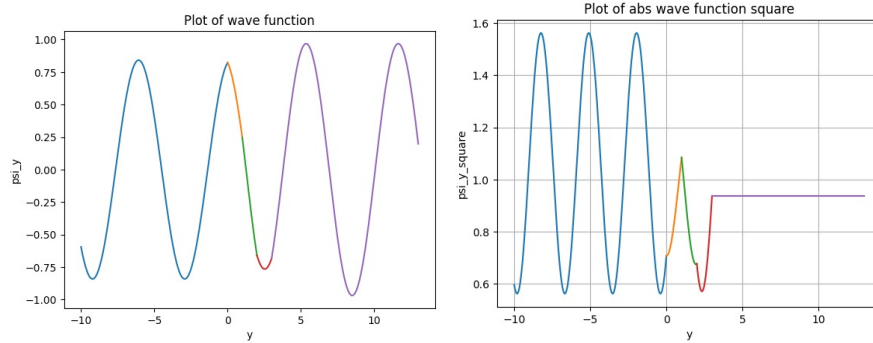
Figure 5.8: Figure of wave function and their absolute value for $n = 4$, $\tilde{V}_0 = 1$, $d = 1$, and $k = 2$.
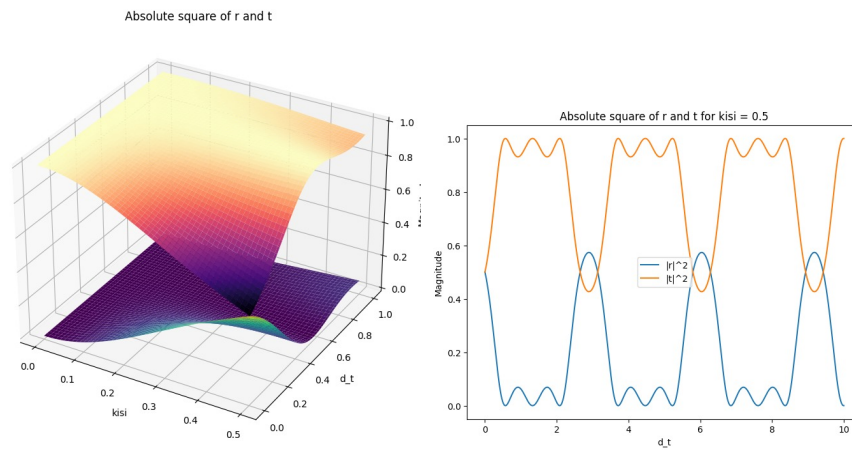
Figure 5.9: The figure depicts the transmission and reflection probabilities as functions of distances and ξ. When compared to Figure 5.7, we can observe decreasing in the amplitude of transmission and reflection probability.

5.2.4 Data visualization and Data analysis for $n = 4$, $\tilde{V}_0 = -2$, $d = 1$ and $k = 1$

Equations 5.31 and 5.32 represent coefficient of reflection and transmission :

$$r = -0.63886367234195 - 0.7666483547201i \qquad (5.31)$$

$$t = -0.0545062190644411 + 0.0336538326200937i \qquad (5.32)$$
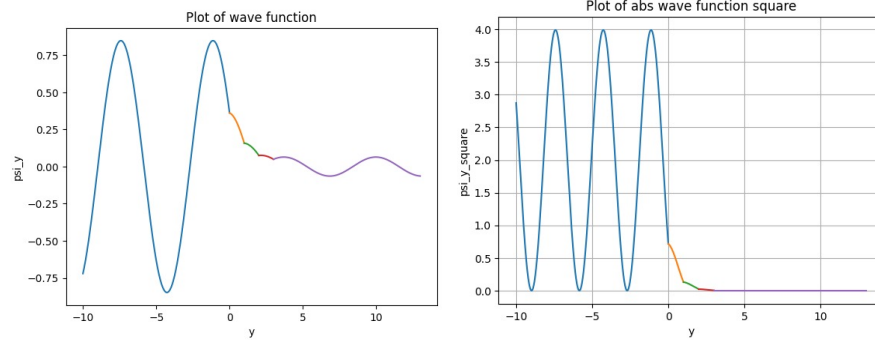
Figure 5.10: Figure of regional wave function and their absolute value for $n = 4$, $\tilde{V}_0 = -2$, $d = 1$, and $k = 1$.
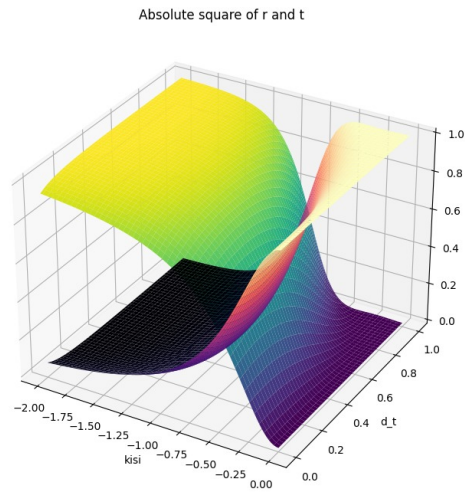
Figure 5.11: As ξ is reduced, the likelihood of transmission decreases, and simultaneously, the probability of reflection increases.

5.2.5 Data visualization and Data analysis for $n = 4$, $\tilde{V}_0 = 1$, $d = 1$ and $k = -2$

Equations 5.33 and 5.34 represent coefficient of reflection and transmission:

$$r = 0.0906937616349323 + 0.0279841779448918i \qquad (5.33)$$

$$t = 0.424810032390065 - 0.900293265422999i \qquad (5.34)$$
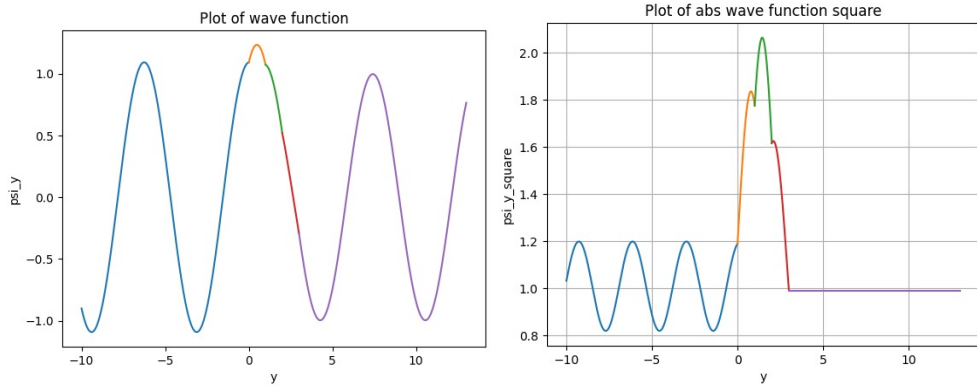
Figure 5.12: Figure of regional wavefunctions and the absolute value of wave function square for k=-2.
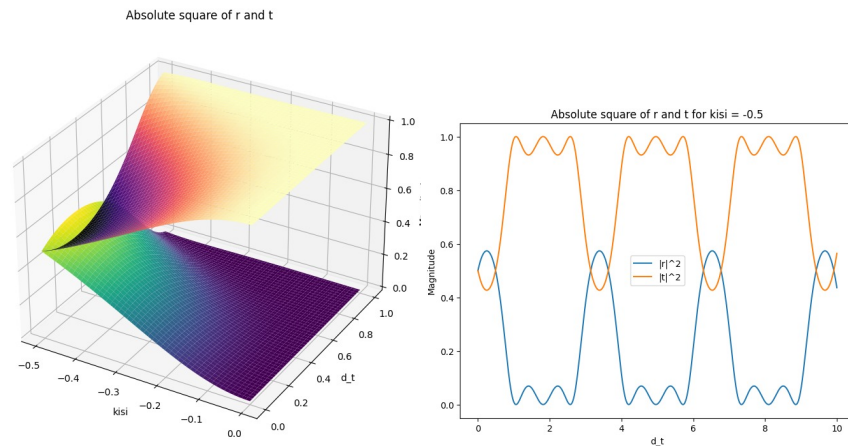
Figure 5.13: From this figure we can see as the ξ increase from -0.5 to 0 transmission probability will decrease and reflection will increase rapidly

5.3 Equal distances and non equal potentials

Through code modification as shown below, we can introduce impurities in the potential strengths, enabling the exploration of a system. By considering these impurities, we have the opportunity to plot the wavefunctions, their absolute values squared, as well as the transmission and reflection probabilities. This approach allows for a comprehensive analysis of the system's behavior under the existence of impurities.

26

```
elif potential_types == 'non equal potentials'and distance_types
    == 'equal distances':
    #graph of psi_y and psi_y_square
    lambdified_psi_y_functions = [lambdify(y, func) for func in
        substituted_psi_y]
    l = num_potential -1  # Define the value of n
    range_of_graph = []  # Initialize an empty list to store the
        ranges
    start = y0_n - (10 * value_of_distance)
    end = y0_n
    range_of_graph.append((start, end))  # Add the first range
    for i in range(l):
        start = y0_n + (i * value_of_distance)
        end = y0_n + ((i+1) * value_of_distance)
        range_of_graph.append((start, end))
    end = y0_n + ((l + 10) * value_of_distance)
    range_of_graph.append((y0_n + (l * value_of_distance), end))
        # Add the last range
    print(range_of_graph)
    plt.figure(1)
    for i, func in enumerate(lambdified_psi_y_functions):
        y_vals = np.linspace(range_of_graph[i][0], range_of_graph
            [i][1], 1000)
        f_vals = np.real(func(y_vals))
        plt.plot(y_vals, f_vals, label=str(substituted_psi_y [i])
            )
    plt.xlabel('y')
    plt.ylabel('psi_y')
    plt.title('Plot of wave function')
    lambdified_abs_psi_y_square = [lambdify(y, func) for func in
        abs_psi_y_square]
    m = num_potential - 1
    range_of_graph1 = []
    start1 = y0_n - (10 * value_of_distance)
    end1 = y0_n
```

```python
    range_of_graph1.append((start1, end1))
    for i in range(m):
        start1 = y0_n + (i * value_of_distance)
        end1 = y0_n + ((i + 1) * value_of_distance)
        range_of_graph1.append((start1, end1))
    end1 = y0_n + ((m + 10) * value_of_distance)
    range_of_graph1.append((y0_n + (m * value_of_distance), end1)
        )
    plt.figure(2)
    for i, func in enumerate(lambdified_abs_psi_y_square):
        y_vals1 = np.linspace(range_of_graph1[i][0],
            range_of_graph1[i][1], 1000)
        f_vals1 = np.vectorize(func)(y_vals1)
        plt.plot(y_vals1, np.real(f_vals1), label=str(
            abs_psi_y_square[i]))
    plt.xlabel('y')
    plt.ylabel('psi_y_square')
    plt.title('Plot of abs wave function square')
    plt.grid(True)
    plt.show()
    # Define the equations
    system_of_equations_algebric = [Eq(eq, 0) for eq in (bc_eq1_n
        + bc_eq2_n)]
    system_of_equations_algebric1 = [eq.subs({symbol: value for
        symbol,
value in zip(kisi_list, kisi_f)}) for eq in
    system_of_equations_algebric]
    symbols_list_algebric = list(set().union(*[eq.free_symbols
        for eq in
system_of_equations_algebric1]))
    # Solve the system of equations
    solution_algebric = smp.solve(system_of_equations_algebric1,
        symbols_list)
    symbol_values_algebric = {symbol: value for symbol, value in
        solution_algebric.items()}
```

```
print(symbol_values_algebric)

r_value_algebric = symbol_values_algebric[r]

t_value_algebric = symbol_values_algebric[t]

r_func = lambdify((d_t), r_value_algebric, modules='numpy')

t_func = lambdify((d_t), t_value_algebric, modules='numpy')

d_t_values2=np.linspace(0,10,1000)

# Calculate the absolute square of r and t for the fixed kisi

abs_r_product_kisi_list = np.abs(r_func(d_t_values2) * np.
    conj(r_func(d_t_values2)))

abs_t_square_kisi_list = np.abs(t_func(d_t_values2) * np.conj
    (t_func(d_t_values2)))

# Plot the 2D graph for the fixed kisi

fig, ax = plt.subplots(figsize=(8, 6))

ax.plot(d_t_values2, abs_r_product_kisi_list, label='|r|^2')

ax.plot(d_t_values2, abs_t_square_kisi_list, label='|t|^2')

ax.set_xlabel('d_t')

ax.set_ylabel('Magnitude')

ax.set_title('Absolute square of r and t for kisi = {}'.
    format(kisi_f))

ax.legend()

plt.show()
```

In the subsequent instances, we will examine specific examples of these impurities, considering various scenarios to explore the effects on the system. These examples will help us gain a deeper understanding of the implications of different impurity configurations on the wavefunctions, absolute values squared, and transmission and reflection probabilities within the non-linear system.

5.3.1 Data visualization and Data analysis for $n = 8$, $\tilde{V}_1 = 1$, $\tilde{V}_2 = 1$, $\tilde{V}_3 = 1$, $\tilde{V}_4 = 1$, $\tilde{V}_5 = 2$, $\tilde{V}_6 = 1$, $\tilde{V}_7 = 1$, $\tilde{V}_8 = 1$, $d = 1$ and $k = 1$

Utilizing the above code and considering the initial values provided earlier, we obtained the transmission and reflection coefficients for the system with impurities.

The results for these coefficients are presented below.

$$r = -0.202133102733224 + 0.383354052275485i \qquad (5.35)$$

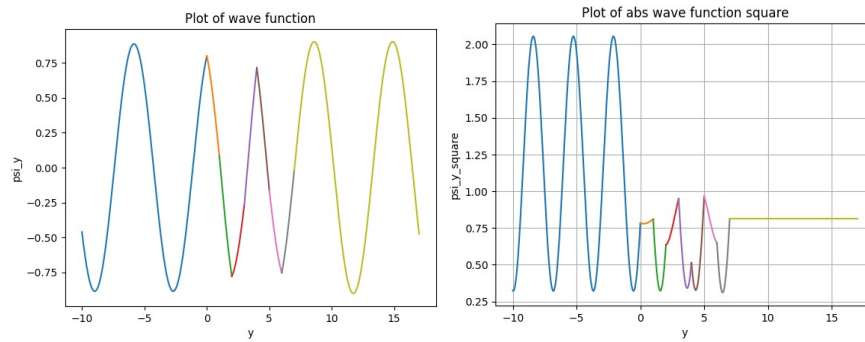$$t = -0.605754596686556 - 0.66726550036437i \qquad (5.36)$$

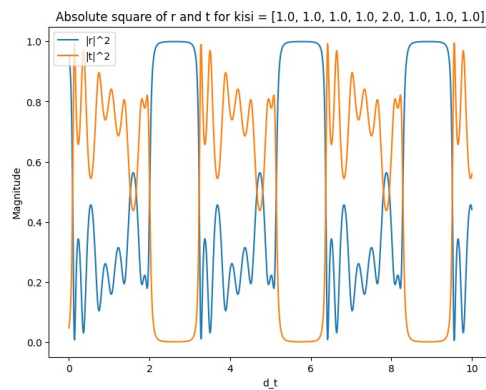Figure 5.14: Regional wave function and the absolute value of wave function square under impurity case

Figure 5.15: The observation made in this context is that a minor impurity on the fifth potential has a significant impact on the overall system probabilities

5.3.2 Data visualization and Data analysis for $n = 8$, $\tilde{V}_1 = 1$, $\tilde{V}_2 = 1$, $\tilde{V}_3 = 1$, $\tilde{V}_4 = 1$, $\tilde{V}_5 = -2$, $\tilde{V}_6 = 1$, $\tilde{V}_7 = 1$, $\tilde{V}_8 = 1$, $d = 1$ and $k = 1$

for such cases, the code provides the following value for transmission and reflection coefficients

$$r = -0.800336145380984 + 0.270977654995426i \qquad (5.37)$$

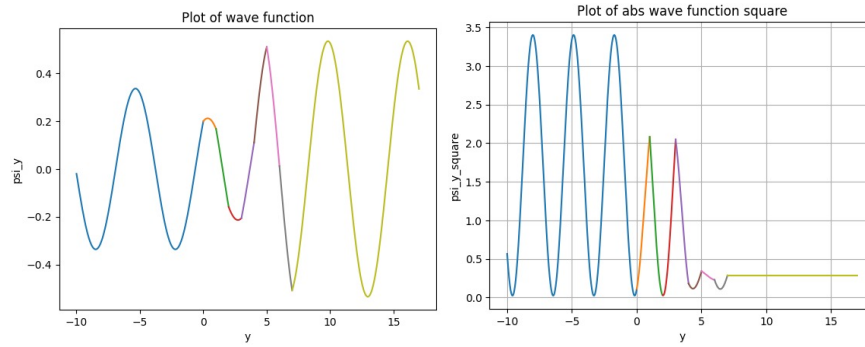$$t = -0.493024386207782 + 0.207268230788822i \qquad (5.38)$$

Figure 5.16: Regional wave function and the absolute value of wave function square under impurity case
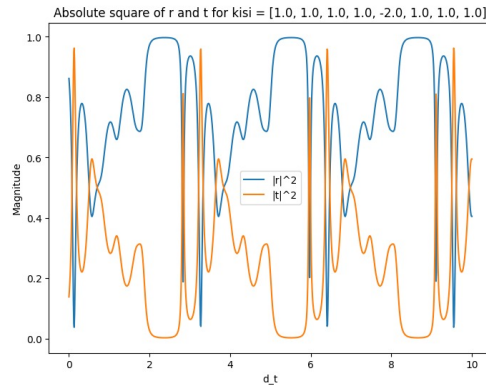
Figure 5.17: By comparing the graph in Figure 5.15, it becomes apparent that only by changing the sign of k, we observe tremendous change in the transmission and reflection probabilities.

31

5.3.3 Data visualization and Data analysis for $n = 8$, $\tilde{V}_1 = 1$, $\tilde{V}_2 = 1$, $\tilde{V}_3 = 1$, $\tilde{V}_4 = 1$, $\tilde{V}_5 = 2$, $\tilde{V}_6 = 1$, $\tilde{V}_7 = 1$, $\tilde{V}_8 = 1$, $d = 1$ and $k = 2$

for such cases, the code provides the following value for transmission and reflection coefficients

$$r = 0.223159115204081 + 0.0196202694529155i \tag{5.39}$$

$$t = -0.490044752768376 + 0.842419844621497i \tag{5.40}$$

Figure 5.18: Regional wave function and the absolute value of wave function square under impurity case

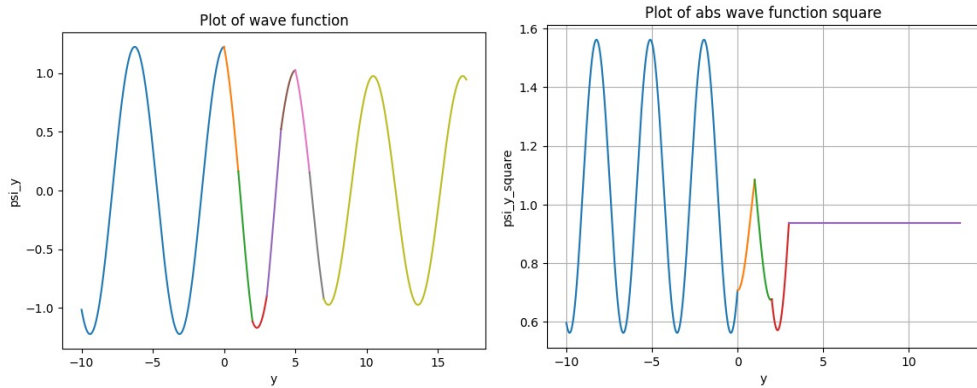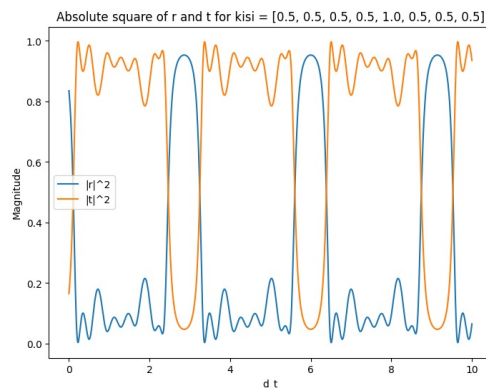Figure 5.19: Upon comparing with Figure 5.15, we observe that increasing k results in a reduction of the amplitude of transmission and reflection probabilities.

5.3.4 Data visualization and Data analysis for $n = 8$, $\tilde{V}_1 = 1$, $\tilde{V}_2 = 1$, $\tilde{V}_3 = 1$, $\tilde{V}_4 = 1$, $\tilde{V}_5 = -2$, $\tilde{V}_6 = 1$, $\tilde{V}_7 = 1$, $\tilde{V}_8 = 1$, $d = 1$ and $k = -0.5$

In our final example of the impurities case, we examined a combination in which the value of ξ became greater than 1. The code generated the reflection and transmission coefficients as follows:

$$r = 0.185344845895648 + 0.960348833223816i \tag{5.41}$$

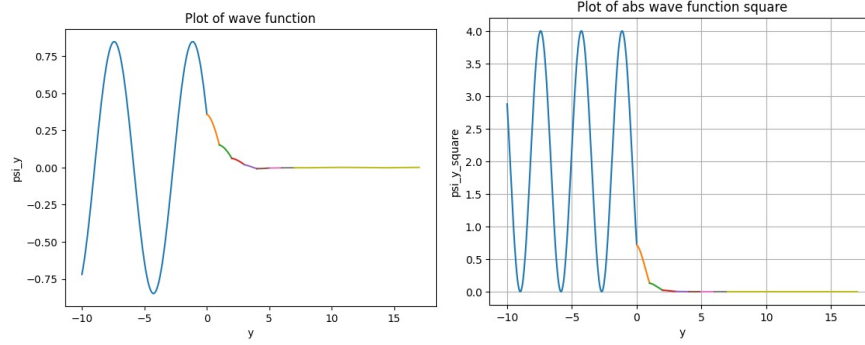$$t = 0.196354344598363 - 0.069443343690528i \tag{5.42}$$

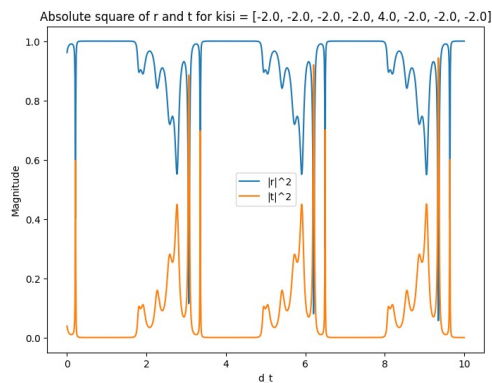Figure 5.20: Regional wave function and the absolute value of wave function square under impurity case

Figure 5.21: Upon comparing this plot with the rest of the graphs, it is evident that the amplitude decreases rapidly as ξ increases. As ξ approaches infinity, the transmission probability tends to 0, while the reflection probability becomes 1.

5.4 Non-equal distances and Non-equal potentials

In this section, we extend the generality of our system by further modifying the code. We consider scenarios where distances and potentials are not constrained to be equal. By relaxing these constraints, we explore a more diverse range of possibilities within the system. Despite these variations, we are still able to calculate the transmission and reflection coefficients, allowing us to gain a comprehensive understanding of the system's behavior under more general conditions.

```
else:
    # Define the equations
    system_of_equations_algebric = [Eq(eq, 0) for eq in (bc_eq1_n
        + bc_eq2_n)]
    symbols_list_algebric = list(set().union(*[eq.free_symbols
        for eq in system_of_equations_algebric]))
    # Solve the system of equations
    solution_algebric = smp.solve(system_of_equations_algebric,
        symbols_list)
    symbol_values_algebric = {symbol: value for symbol, value in
        solution_algebric.items()}
    print(symbol_values_algebric)
    r_value_algebric = symbol_values_algebric[r]
    t_value_algebric = symbol_values_algebric[t]
```

Now let's consider some examples for this specific case imagine we consider a system of 3 potentials with the strength of $\tilde{V}_1 = 1, \tilde{V}_2 = 2, \tilde{V}_3 = 3$ and $d_1 = 1, d_2 = 2$ and $k = 1$ the coefficient of transmission and reflection are as followed:

$$r = 0.44765413320866 - 0.799041236786126i \qquad (5.43)$$

$$t = -0.305274111412524 + 0.260665678288869i \qquad (5.44)$$

on the other hand the symbolic answer for transmission coefficient is as follows:

$$t_3 = \frac{-8i}{\gamma + \omega} \tag{5.45}$$

$$\gamma = \xi_1\xi_2\xi_3(e^{2i(d_{1-2}+d_{2-3})} - e^{2id_{1-2}} - e^{2id_{2-3}} + 1) + 2i\xi_1\xi_2(1 - e^{2id_{1-2}}) \tag{5.46}$$

$$\omega = 2i\xi_1\xi_3(1 - e^{2i(d_{1-2}+d_{2-3})}) + 2i\xi_2\xi_3(1 - e^{2id_{2-3}}) - 4(\xi_1 + \xi_2 + \xi_3) - 8i \tag{5.47}$$

also we can see for $\tilde{V}_1 = 1, \tilde{V}_2 = -2$, and $d_1 = 1$ and $k = 2$

$$r = 0.288273992003145 + 0.45235738801617i \tag{5.48}$$

$$t = 0.288273992003145 + 0.45235738801617i \tag{5.49}$$

$$t = \frac{4}{\xi_1\zeta_2(e^{2id_1} - 1) - 2i(\xi_1 + \xi_2) + 4} \tag{5.50}$$

The code allows users to input a wide range of variational values, enabling the simulation of diverse scenarios with any desired number of potentials.

Chapter 6

CONCLUSION

In this project, we embarked on an exploration of the one-dimensional form of multiple Dirac delta potentials, employing simulations to unravel the intricate behavior of the system. The Schrödinger equation governing the dynamics of the particles was transformed into a dimensionless representation, allowing us to delve deep into the study of scattering problems. A focal point of our investigation was the renowned Kronig-Penney model, an exemplary system widely employed in studying scattering phenomena. By analyzing and simulating this system, we aimed to gain invaluable insights into the broader implications and applications of multiple Dirac delta potentials in quantum mechanics. To facilitate our research, we design an implementation of a user interface program using the versatile Python programming language. The program was crafted to design a system governed by multiple delta Dirac potentials. With this user-friendly tool, we seamlessly generated and analyzed wave functions arising from incident and outgoing waves. In order to ensure the accuracy of our system, we imposed boundary conditions at every point along the potential. These boundary conditions played a crucial role in capturing the true essence of the scattering process. Through rigorous computations, we obtained essential physical parameters, including transmission and reflection coefficients, regional wave functions and etc. The implications of our findings were profound. Not only did we achieve a better understanding of the scattering problem within the context of multiple Dirac delta potentials, but we also advanced our knowledge of

how these potentials impact the scattering behavior of particles. In conclusion, our thesis represents a significant contribution to the study of scattering phenomena in quantum mechanics, particularly in the context of impurities in one-dimensional systems with multiple delta Dirac potentials. The development of the Python-based user interface program further enhances the accessibility and accuracy of analyzing and visualizing such systems. As we progress in our understanding of quantum mechanics, the knowledge gained from this research paves the way for future advancements and applications in various scientific and technological domains.

REFERENCES

[1] R. d. L. Kronig and W. G. Penney, "Quantum mechanics of electrons in crystal lattices," *Proceedings of the royal society of London. series A, containing papers of a mathematical and physical character*, vol. 130, no. 814, pp. 499–513, 1931.

[2] B. Sahu and B. Sahu, "Accurate delta potential approximation for a coordinate-dependent potential and its analytical solution," *Physics Letters A*, vol. 373, no. 44, pp. 4033–4037, 2009.

[3] I. R. Lapidus, "Resonance scattering from a double δ-function potential," *American Journal of Physics*, vol. 50, no. 7, pp. 663–664, 1982.

[4] P. Senn, "Threshold anomalies in one-dimensional scattering," *American Journal of Physics*, vol. 56, no. 10, pp. 916–921, 1988.

[5] P. Berman, "Transmission resonances and bloch states for a periodic array of delta function potentials," *American Journal of Physics*, vol. 81, no. 3, pp. 190–201, 2013.

[6] G. Cordourier-Maruri, R. De Coss, and V. Gupta, "Transmission properties of the one-dimensional array of delta potentials," *Modern Physics Letters B*, vol. 25, no. 16, pp. 1349–1358, 2011.

[7] Z. Ahmed, S. Kumar, M. Sharma, and V. Sharma, "Revisiting double dirac delta potential," *European Journal of Physics*, vol. 37, no. 4, p. 045406, 2016.

[8] S. Patil, "Quadrupolar, triple δ-function potential in one dimension," *European journal of physics*, vol. 30, no. 3, p. 629, 2009.

[9] V. E. Barlette, M. M. Leite, and S. K. Adhikari, "Integral equations of scattering in one dimension," *American Journal of Physics*, vol. 69, no. 9, pp. 1010–1013, 2001.

[10] D. Lessie and J. Spadaro, "One-dimensional multiple scattering in quantum mechanics," *American Journal of Physics*, vol. 54, no. 10, pp. 909–913, 1986.