Quantum Mechanics Homework № 2

Juan Daniel Vasconez, Yachay Tech University October 15, 2024

1 Libraries for this Homework:

Listing 1: Python Code – Enviroment Preparation

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
import scipy.optimize as opt
```

Problem 1: Probability densities

Consider the probability density function (PDF) of the Beta distribution is defined as:

$$\rho(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \tag{1}$$

over the domain $x \in [0,1]$ when $\alpha > 0$ and $\beta > 0$ are shape parameters $B(\alpha,\beta)$ is the Beta function, which serves as the normalisation constant and is defined as:

$$B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{2}$$

where Γ refers to Gamma functions.

1.1 Find the expectation value $\langle x \rangle$

We need to find the expectation value $\langle x \rangle$ which is calculated as:

$$\langle x \rangle = \int_0^1 x \cdot \rho(x;\alpha,\beta)dx \tag{3}$$

First let's substitute the PDF into the formula:

$$\langle x \rangle = \int_0^1 x \cdot \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}dx \tag{4}$$

This can be simplified to:

$$\langle x \rangle = \frac{1}{B(\alpha,\beta)} \int_0^1 x^{\alpha}(1-x)^{\beta-1}dx \tag{5}$$

Since the Beta function is defined as:

$$B(\alpha,\beta) = \int_0^1 x^{m-1}(1-x)^{n-1}dx \tag{6}$$

We can recognize the previous equation as:

$$\langle x \rangle = \frac{B(\alpha+1,\beta}{B(\alpha,\beta)} \tag{7}$$

By the relation between Beta and Gamma functions:

$$B(\alpha,\beta) = \frac{\gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha,\beta)} \tag{8}$$

We can translate the expectation value to:

$$\langle x \rangle = \frac{\Gamma(\alpha+1)\Gamma(\beta)}{\Gamma(\alpha+\beta+1)} \cdot \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \tag{9}$$

Now, for simplification, lets consider that:

$$\begin{aligned}
\Gamma(x+1) &= \int_0^\infty t^x e^{-t}\,dt \\
&= \left[-t^x e^{-t}\right]_0^\infty + x\int_0^\infty t^{x-1}e^{-t}\,dt \\
&= 0 + x\int_0^\infty t^{x-1}e^{-t}\,dt \\
&= x\Gamma(x)
\end{aligned}$$

Therefore, by having that $\Gamma(\alpha+1)$, the expression reduces to:

$$\langle x \rangle = \frac{\alpha}{\alpha+\beta} \tag{10}$$

1.2 Find the expectation value $\langle x^2 \rangle$

The expectation value, is calculated as:

$$\langle x^2 \rangle = \int_0^1 x^2 \cdot \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} dx \tag{11}$$

This again, as previously is the Beta function, so we can rewrite the expression as:

$$\langle x^2 \rangle = \frac{B(\alpha+2,\beta)}{B(\alpha,\beta)} \tag{12}$$

Using the relation between Beta and Gamma functions, we have:

$$\begin{aligned}
B(\alpha+2,\beta) &= \frac{\Gamma(\alpha+2)\Gamma(\beta)}{\Gamma(\alpha+\beta+2)} \\
B(\alpha,\beta) &= \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}
\end{aligned} \tag{13}$$

So, we rewrite as:

$$\langle x^2 \rangle = \frac{\Gamma(\alpha+2)}{\Gamma(\alpha)} \cdot \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha+\beta+2)} \tag{14}$$

Using that $\Gamma(\alpha+2) = (\alpha+1)\alpha\Gamma(\alpha)$

$$\begin{aligned}
\langle x^2 \rangle &= \frac{(\alpha+1)\alpha\Gamma(\alpha)}{\Gamma(\alpha)} \cdot \frac{\Gamma(\alpha+\beta)}{(\alpha+\beta+1)(\alpha+\beta)\Gamma(\alpha+\beta)} \\
\langle x^2 \rangle &= \frac{(\alpha+1)\alpha}{(\alpha+\beta+1)(\alpha+\beta)}
\end{aligned} \tag{15}$$

Therefore:

$$\langle x^2 \rangle = \frac{(\alpha+1)\alpha}{(\alpha+\beta+1)(\alpha+\beta)} \tag{16}$$

1.3 Find σ_x

We can define the standard deviation as:

$$\sigma_x = \sqrt{\langle x^2 \rangle - \langle x \rangle^2} \tag{17}$$

Computing $\langle x \rangle^2$ we have:

$$\langle x \rangle^2 = (\frac{\alpha}{\alpha + \beta})^2 = \frac{\alpha^2}{(\alpha + \beta)^2} \tag{18}$$

For convenience, we will rewrite the Variance as:

$$\begin{aligned} Var(x) &= \frac{(\alpha + 1)\alpha}{(\alpha + \beta + 1)(\alpha + \beta)} - \frac{\alpha^2(\alpha + \beta + 1)}{(\alpha + \beta)^2(\alpha + \beta + 1)} \\ Var(x) &= \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} \end{aligned} \tag{19}$$

Finally, the Standard deviation would be:

$$\sigma_x = \sqrt{\frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}} \tag{20}$$

1.4 If $\rho(x; \alpha, \beta) \equiv |\psi(x)|^2$ for a particle in an infinite box over $x \in [0, 1]$, find the conditions (for α and β) for which $\psi(0) = \psi(1) = 0$. What do the α and β constants represent?

Since,

$$\psi(0) = \psi(1) = 0 \equiv |\psi(0)|^2 = |\psi(1)|^2 = 0 \tag{21}$$

This implies that, $\rho(0; \alpha, \beta) = 0$ and $\rho(1; \alpha, \beta) = 0$, therefore,

- At $x = 0$, for $\rho(0; \alpha, \beta)$, we need $\alpha > 1$ so that $x^{\alpha-1} \to 0$
- At $x = 1$, for $\rho(1; \alpha, \beta)$, we need $\beta > 1$ so that $(1 - x)^{\beta-1} \to 0$

For what we can describe a physical meaning for αandβ, so as, this constants can be thought as shape parameters for the behaviour of the wave function in relation to this boundaries, where α controls the behaviour of the PDF near x=0 making it more narrower, and β controls the PDF function around x=1, making it to increase or decrease the probability that describes near the x=1 point

1.5 Plug some fiducial numbers for these constants, and sketch the graph of $\rho(x; \alpha, \beta)$ using your favourite programming language.

Lets, use python, for this, the code is written as:

Listing 2: Python Code – Normalized Beta PDF

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
```

```
# Define the x values and parameters alpha and beta
x_values = np.linspace(0, 1, 100)
alpha = 4
beta = 5

# Define the normalization constant B(alpha, beta)
def beta_normalization(alpha, beta):
    return (gamma(alpha) * gamma(beta)) / gamma(alpha + beta)

# Define the normalized Beta PDF
def beta_pdf_normalized(x, alpha, beta):
    B = beta_normalization(alpha, beta)
    return (x**(alpha - 1) * (1 - x)**(beta - 1)) / B

# Compute the normalized y values for the Beta PDF
y_values_normalized = beta_pdf_normalized(x_values, alpha, beta)

# Plot the normalized Beta distribution
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values_normalized, label=f'Normalized Beta PDF ($\\↩
   alpha={alpha}, \\beta={beta}$)', color='green')
plt.xlabel('x')
plt.ylabel('$\\rho(x)$')
plt.grid(True)
plt.legend()
plt.title('Normalized Beta Distribution PDF')
plt.show()
```

By running the code, we end up with a plot like this:

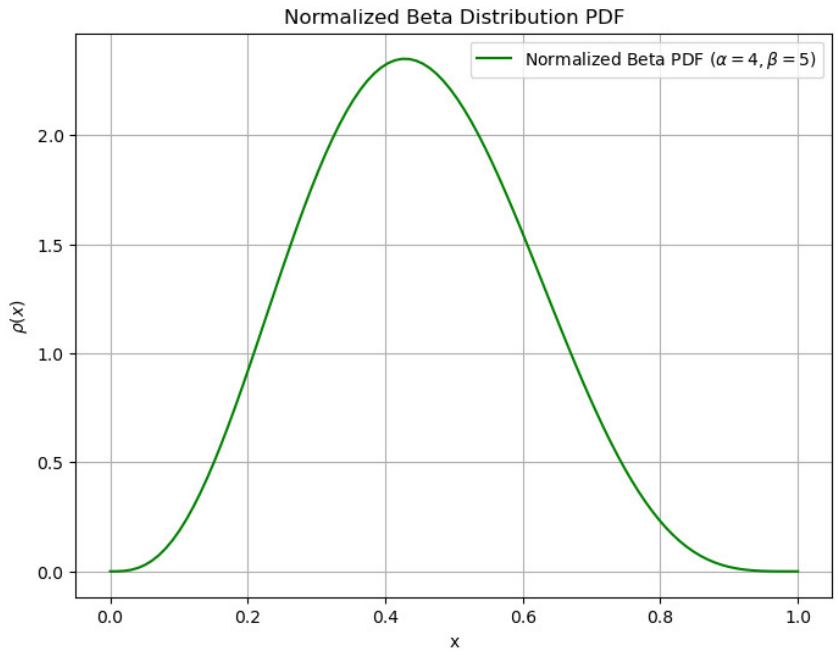

Figure 1: Beta PDF Plot

Where in this figure 1 we can see the form of a PDF characterized by:

$$\rho(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \tag{22}$$

and notice the stated values as how the PDF starts at x=0 and goes to 0 in x=1. With, $\alpha = 4$ and $\beta = 5$ for a more similar to the Gaussian distribution.

Problem 2: Infinite square well potential and expectation values

In class we solved the Schrodinger squation for an infinite square well potential of width L. Such potential allows for bound solutions only as the particle cannot scape from the well. The solutions we found had the following functional form:

$$\psi_n(x) = \sqrt{\frac{2}{L}} \sin \frac{n\pi}{L} x \tag{23}$$

For the n-th state by the function above, calculate:

1.6 The expectation values associated with the position x: $\langle x \rangle$, $\langle x^2 \rangle$

The expectation value of the position x, is defined as:

$$\langle x \rangle = \int_0^L x|\psi_n(x)|^2 dx \tag{24}$$

Since, the wave function $\psi_n(x)$ is defined, the pdf is:

$$|\psi_n(x)|^2 = \frac{2}{L} \sin^2 (\frac{2\pi x}{L}) \tag{25}$$

Thus, the integral becomes:

$$\begin{aligned}
\langle x\rangle &= \int_o^L x\frac{2}{L}\sin^2\left(\frac{n\pi x}{L}\right) \\
&= \frac{2}{L}\int_0^L x\frac{1-\cos\left(\frac{2n\pi x}{L}\right)}{2}dx \\
&= \frac{1}{L}\int_0^L xdx - \int_0^L x\cos\left(\frac{2n\pi x}{L}\right)dx \\
&= \frac{L}{2} - \frac{1}{L}\int_0^L x\cos\left(\frac{2n\pi x}{2}\right)dx \\
&= \frac{L}{2} - x\cdot\frac{L}{2n\pi}\sin\left(\frac{2n\pi x}{L}\right)\Big|_0^L - \int_0^L \frac{L}{2n\pi}\sin\left(\frac{2n\pi x}{L}\right)dx \\
\langle x\rangle &= \frac{L}{2}
\end{aligned} \tag{26}$$

Now for the calculation of $\langle x^2\rangle$

$$\langle x^2\rangle = \int_0^L x^2|\psi_n(x)|^2\,dx \tag{27}$$

Given that $\psi_n(x) = \sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right)$, the probability density becomes:

$$|\psi_n(x)|^2 = \frac{2}{L}\sin^2\left(\frac{n\pi x}{L}\right) \tag{28}$$

Thus, the integral for $\langle x^2\rangle$ is:

$$\begin{aligned}
\langle x^2\rangle &= \frac{2}{L}\int_0^L x^2\sin^2\left(\frac{n\pi x}{L}\right)\,dx \\
&= \frac{1}{L}\int_0^L x^2\,dx - \frac{1}{L}\int_0^L x^2\cos\left(\frac{2n\pi x}{L}\right)\,dx \\
&= \frac{L^2}{3} - [2x(\frac{L^2}{2n^2\pi^2})\cos\frac{2n\pi x}{L}]\Big|_0^L \\
\langle c|\,ketx^2 &= \frac{L^2}{3} - \frac{L^2}{2\pi^2n^2}
\end{aligned} \tag{29}$$

1.7 The expectation values associated with the momentum p: $\langle p\rangle\,,\langle p^2\rangle$

Calculation of $\langle p\rangle$ and $\langle p^2\rangle$

Expectation value of p

The expectation value of the momentum p is given by:

$$\langle p\rangle = \int_0^L \psi_n^*(x)\hat{p}\psi_n(x)\,dx = -i\hbar\int_0^L \psi_n(x)\frac{d}{dx}\psi_n(x)\,dx \tag{30}$$

Substituting $\psi_n(x) = \sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right)$, we get:

$$
\begin{aligned}
\langle p \rangle &= -i\hbar\sqrt{\frac{2}{L}}\int_0^L \sin\left(\frac{n\pi x}{L}\right)\frac{d}{dx}\sin\left(\frac{n\pi x}{L}\right)\,dx \\
&= -i\hbar\sqrt{\frac{2}{L}}\cdot\frac{n\pi}{L}\int_0^L \sin\left(\frac{n\pi x}{L}\right)\cos\left(\frac{n\pi x}{L}\right)\,dx \\
&= -i\hbar\frac{n\pi}{L}\frac{1}{2}\sqrt{\frac{2}{L}}\int_0^L \sin\left(\frac{2n\pi x}{L}\right)\,dx \\
\langle p \rangle &= 0
\end{aligned}
\tag{31}
$$

Next, we calculate $\langle p^2 \rangle$:

$$
\langle p^2 \rangle = \int_0^L \psi_n^*(x)\hat{p}^2\psi_n(x)\,dx = -\hbar^2\int_0^L \psi_n(x)\frac{d^2}{dx^2}\psi_n(x)\,dx \tag{32}
$$

The second derivative of $\psi_n(x) = \sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right)$ is:

$$
\frac{d^2}{dx^2}\sin\left(\frac{n\pi x}{L}\right) = -\left(\frac{n\pi}{L}\right)^2\sin\left(\frac{n\pi x}{L}\right) \tag{33}
$$

Substitute this into the integral:

$$
\begin{aligned}
\langle p^2 \rangle &= -\hbar^2\int_0^L \sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right)\left(-\left(\frac{n\pi}{L}\right)^2\sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right)\right)\,dx \\
&= \hbar^2\left(\frac{n\pi}{L}\right)^2\frac{2}{L}\int_0^L \sin^2\left(\frac{n\pi x}{L}\right)\,dx \\
&= \hbar^2\left(\frac{n\pi}{L}\right)^2\int_0^L \frac{1-\cos\left(\frac{2n\pi x}{L}\right)}{2}\,dx \\
&= \hbar^2\left(\frac{n\pi}{L}\right)^2\frac{2}{L}\frac{L}{2} \\
\langle p^2 \rangle &= \hbar^2\frac{n^2\pi^2}{L^2}
\end{aligned}
\tag{34}
$$

1.8 The dispersion σ_x and σ_p and their product $\sigma_x\sigma_p$

Firstly, to calculate the dispertion for the position x (σ_x), the formula states that:

$$
\begin{aligned}
\sigma_x &= \sqrt{\langle x^2 \rangle - \langle x \rangle^2} = \sqrt{\frac{L^2}{3} - \frac{L^2}{2\pi^2 n^2} - \frac{L^2}{4}} \\
\sigma_p &= \sqrt{\langle p^2 \rangle - \langle p \rangle^2} = \sqrt{\hbar^2\frac{n^2\pi^2}{L^2}}
\end{aligned}
\tag{35}
$$

And now, the multiplication between them, stated as:

$$
\sigma_x\sigma_p = \sqrt{\frac{L^2}{3} - \frac{L^2}{2\pi^2 n^2} - \frac{L^2}{4}}\cdot\sqrt{\hbar^2\frac{n^2\pi^2}{L^2}} = \frac{\hbar n\pi}{\sqrt{12}}\sqrt{1 - \frac{6}{\pi^2 n^2}} \tag{36}
$$

1.9 Use programming tools to make a plot of $(\sigma_x\sigma_p)$ vs n.

For the purpose we will use python coding as:

Listing 3: Python Code – Sigmas Scattered Plot.

```python
import scipy as sp

# Define constants
hbar = sp.hbar # Reduced Planck's constant in J s
pi = np.pi

# Function to calculate sigma_x * sigma_p
def sigma_x_sigma_p(n):
    return (hbar * n * pi / np.sqrt(12)) * np.sqrt(1 - 6 / (pi**2 * n**2))

# Range of n values
n_values = np.linspace(1, 50, 500)

# Calculate sigma_x * sigma_p for each n
sigma_values = sigma_x_sigma_p(n_values)

# Scatter plot for each integer n till n = 10
n_values_int_till_10 = n_values_int[:10]
sigma_values_int_till_10 = sigma_values_int[:10]
plt.figure(figsize=(8, 6))
plt.scatter(n_values_int_till_10, sigma_values_int_till_10, color='r', label='Scatter plot for n till 10')
for i, txt in enumerate(n_values_int_till_10):
    plt.annotate(f'n={txt}', (n_values_int_till_10[i], sigma_values_int_till_10[i]), textcoords="offset points", xytext=(0,10), ha='center')
plt.title('Scatter plot of $\sigma_x \sigma_p$ vs n')
plt.xlabel('n')
plt.ylabel(r'$\sigma_x \sigma_p$')
plt.grid(True)
plt.show()
```

Therefore, we can have a plot like this:

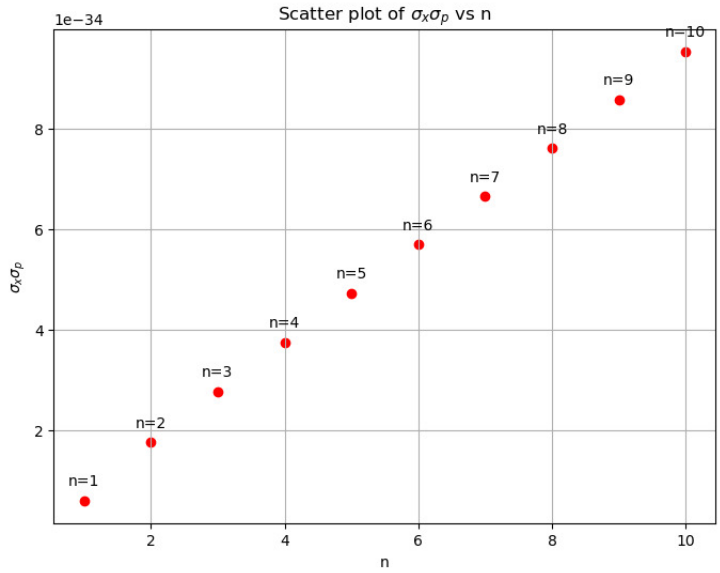

Figure 2: Scattered Plot of Sigmas vs N

1.10 Is the uncertainty principle satisfied? Which of the $\psi_n(x)$ states comes closes to the uncertainty limit?

To achieve this calculation, we will iterate the calculation,

Listing 4: Python Code – Uncertainty Limit.

```
# Calculate the uncertainty limit
uncertainty_limit = hbar / 2

# Calculate the ratio of sigma_x * sigma_p to the uncertainty limit
ratios = sigma_values / uncertainty_limit

# Find the state that comes closest to the uncertainty limit
closest_state_index = np.argmin(np.abs(ratios - 1))
closest_state_n = n_values_int[closest_state_index]
closest_state_ratio = ratios[closest_state_index]

# Check if the uncertainty principle is satisfied
is_satisfied = np.all(sigma_values >= uncertainty_limit)

print(f"Is the uncertainty principle satisfied? {'Yes' if is_satisfied else 'No'}")
print(f"The state \psi_{closest_state_n}(x) comes closest to the uncertainty limit with a ratio of {closest_state_ratio:.2f}")
print(f"The value of n that gives the closest sigma multiplication to the uncertainty principle is {closest_state_n}")
```

Where we can identify 3 things:

- The uncertainty principle IS satisfied
- The ratio in how $\psi_n(x)$ comes closes to the uncertainty principle is 1.14
- And the n that gives the closest sigma multiplication to the Heisenberg uncertainty limit is 1

And this makes sense, since the relation is linear, the dispersion for each expected value will grow for n>1

Problem 3: Plane waves for matter particles

Consider that eh plane wave for a matter particle moving in the x-direction with momentum $p = \hbar k$ is:

$$\psi(x,t) = \cos(kx - \omega t) + \gamma \sin(kx - \omega t) \tag{37}$$

where γ is a constant. A physical requirement is that an arbitrary displacement of x or an arbitrary shift of t should not alter the character of the wave. We therefore demand that after a phase shift by some constant ϵ, we have the following relationship:

$$\cos(kx - \omega t + \epsilon) + \gamma \sin(kx - \omega t + \epsilon) = a[\cos(kx - \omega t) + \gamma \sin(kx - \omega t)] \tag{38}$$

where a is some constant that may depend on ϵ

1.11 Use trigonometric identities to write the equation for γ, ϵ and a that follow the above requirement.

Using the trigonometric identities for sinusoidal functions with an argument of the sum of angles, we can rewrite the expression as:

$$\begin{aligned} \cos(kx - \omega t + \epsilon) &= \cos(kx - \omega t)\cos(\epsilon) - \sin(kx - \omega t)\sin(\epsilon) \\ \sin(kx - \omega t + \epsilon) &= \sin(kx - \omega t)\cos(\epsilon) + \cos(kx - \omega t)\sin(\epsilon) \end{aligned} \tag{39}$$

Thus:

$$a[\cos(kx - \omega t) + \gamma \sin(kx - \omega t)] = [\cos(\epsilon) + \gamma \sin(\epsilon)]\cos(kx - \omega t) + [\gamma \cos(\epsilon) - \sin(\epsilon)]\sin(kx - \omega t) \tag{40}$$

We end up with a linear sistem stated as:

$$\begin{aligned} \cos(\epsilon) + \gamma \sin(\epsilon) &= a \\ \gamma \cos(\epsilon) - \sin(\epsilon) &= a\gamma \end{aligned} \tag{41}$$

1.12 Find the possible solutions for γ and the associated a.

For this system solution, we will use python and the library sympy. The code will be stated as:

Listing 5: Python Code – a and gamma system.

```python
import sympy as sp

# Declare the symbols
epsilon, gamma, a = sp.symbols('epsilon gamma a')

# Define the system of equations
eq1 = sp.Eq(sp.cos(epsilon) + gamma * sp.sin(epsilon), a)
eq2 = sp.Eq(gamma * sp.cos(epsilon) - sp.sin(epsilon), a * gamma)
display(eq1, eq2)
# Solve the system for gamma and a
solutions = sp.solve([eq1, eq2], (gamma, a))
display(solutions)
```

Where, the solutions for the system that the script encountered are:

$$\begin{aligned} a_1, \gamma_1 &= e^{i\epsilon}, i \\ a_2, \gamma_2 &= e^{-i\epsilon}, -i \end{aligned} \tag{42}$$

1.13 Which one is the solution that corresponds to our conventional description of a matter wave?

Since we know, that for unbound particles, matter waves have negative energies, therefore, the solution we should select is $\gamma = -i$, thus, we would have $a = e^{-i\epsilon}$

1.14 Take the matter wave solution. Plug some fiducial numbers (with realistic physical units) for the constants k and ω, and use your favorite programming took to sketch $|\psi(x,t)|^2$ versus x for $t = 0$ and two later times

For matter waves, we have stablished that the solution and squared wavefunction are:

$$\begin{aligned} \psi(x) &= e^{i(kx-\omega t)} \\ |\psi(x)|^2 &= 1 \end{aligned} \tag{43}$$

Thus, by using python, we can make a plot for this functions, using the next code:

Listing 6: a and gamma system plot.

```python
# Define constants
k = 2 * np.pi  # wave number
omega = 2 * np.pi  # angular frequency

# Define the wave function
def psi(x, t):
    return np.exp(1j * (k * x - omega * t))

# Define |Psi(x, t)|^2
```

```
10 def psi_squared(x, t):
11     return np.abs(psi(x, t)) ** 2
12
13 # Define x range
14 x = np.linspace(-10, 10, 1000)
15
16 # Plot |Psi(x, t)|^2 for t = 0 and two later times
17 t_values = [0, 0.5, 1.0]  # Three different times
18
19 plt.figure(figsize=(8, 6))
20
21 for t in t_values:
22     plt.plot(x, psi_squared(x, t), label=f't = {t}')
23
24 plt.xlabel('x')
25 plt.ylabel(r'$|\Psi(x, t)|^2$')
26 plt.title(r'Squared Matter Wave Function $|\Psi(x, t)|^2$')
27 plt.legend()
28 plt.grid(True)
29 plt.show()
```

And end up with the next graphic:

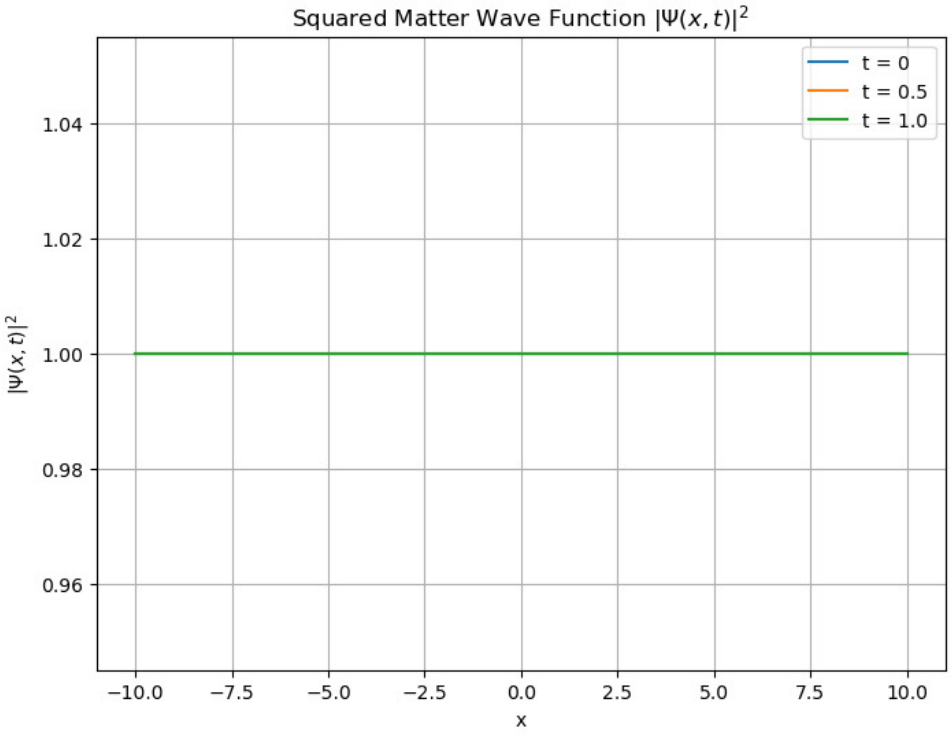

Figure 3: Matter Waves for 3 different Times

1.15 Use several mater wave solutions localised around k_0 to construct a wave packet, report the phase and group velocities, and use your favorite programming tool to sketch your wave packet.

By setting $k_0 = 2\pi$ we can make a code for the wave packet on python:

Listing 7: Python Code – Wave Packet Plot

```
x = np.linspace(-10, 10, 1000)

# Define a wave packet function
def wave_packet_t(x, t, A, s, k, w):
  """
  Function to create a wave packet.
  Inputs: A, k, s -> parameters, x -> vector
  Output: psi_x -> wave packet
  """
  psi_xt = A*np.exp(-(x - t)**2/(2*s**2))*np.sin(k*x - w*t)

  return psi_xt

A = 1
s = 2
k = 2*np.pi

# Define the new parameters
w = 3*np.pi

# Time vector
t = np.linspace(0., 10., 50)

# Evaluate the packet:
y0 = wave_packet_t(x, t[2], A, s, k, w)
y1 = wave_packet_t(x, t[10], A, s, k, w)
y2 = wave_packet_t(x, t[20], A, s, k, w)

# Combined plot for all times in one figure with subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

```

```
# Plot for t = 1
axs[0, 0].plot(x, y0, color='blue', label='t = 1')
axs[0, 0].set_title('Wave Packet Function at t = 1')
axs[0, 0].set_xlabel('t')
axs[0, 0].set_ylabel(r'$\psi(x, t)$')
axs[0, 0].legend()
axs[0, 0].grid(True)

# Plot for t = 5
axs[0, 1].plot(x, y1, color='cyan', label='t = 5')
axs[0, 1].set_title('Wave Packet Function at t = 5')
axs[0, 1].set_xlabel('t')
axs[0, 1].set_ylabel(r'$\psi(x, t)$')
axs[0, 1].legend()
axs[0, 1].grid(True)

# Plot for t = 10
axs[1, 0].plot(x, y2, color='black', label='t = 10')
axs[1, 0].set_title('Wave Packet Function at t = 10')
axs[1, 0].set_xlabel('t')
axs[1, 0].set_ylabel(r'$\psi(x, t)$')
axs[1, 0].legend()
axs[1, 0].grid(True)

# Combined plot for all times
axs[1, 1].plot(x, y0, color='blue', label='t = 1')
axs[1, 1].plot(x, y1, color='cyan', label='t = 5')
axs[1, 1].plot(x, y2, color='black', label='t = 10')
axs[1, 1].set_title('Wave Packet Function at Different Times')
axs[1, 1].set_xlabel('t')
axs[1, 1].set_ylabel(r'$\psi(x, t)$')
axs[1, 1].legend()
axs[1, 1].grid(True)

plt.tight_layout()
plt.show()
```

Where, our phase and group velocity are stated as:

$$\begin{aligned} V_p &= \frac{\omega}{k} = \frac{3}{2} \\ V_g &= \frac{V_p}{2} = \frac{3}{4} \end{aligned} \tag{44}$$

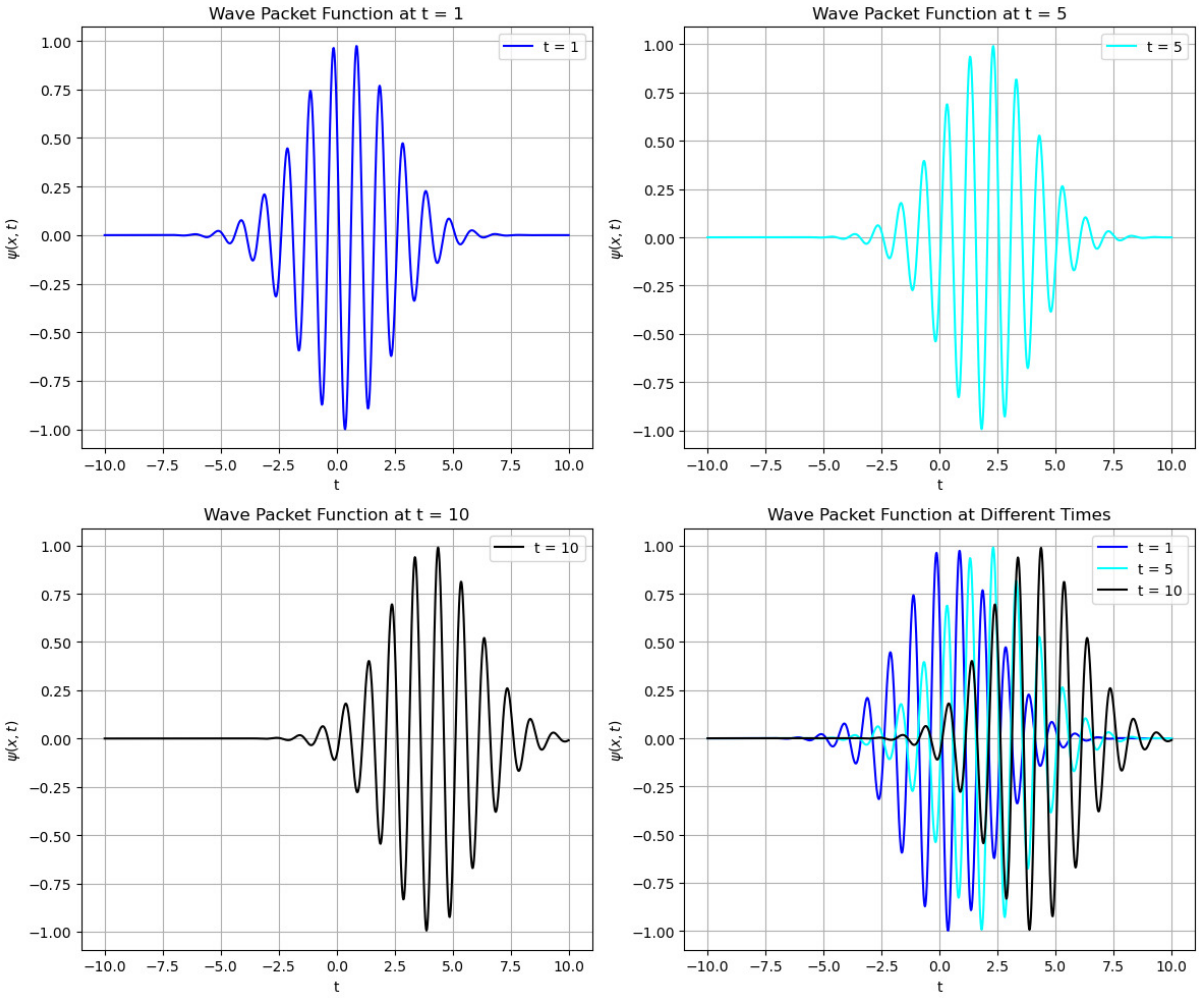

Figure 4: Matter Waves for 3 different Times

Problem 4: Finite square well potential

In class we studied the finite square well potential and found that this potential admits both scattering states (when $E > 0$) and bound states (when $E < 0$). For the latter, we derived the even solutions and numerically solved a transcendental equation for the allowed energies.

1.16 Normalise the even states found in class.

Since, the even states can be described as:

$$\psi(x)_{even} = \begin{cases} Fe^{kx} & x < -a \\ D\cos(lx) & -a \leq x \leq a \\ F^{-kx} & x > a \end{cases} \tag{45}$$

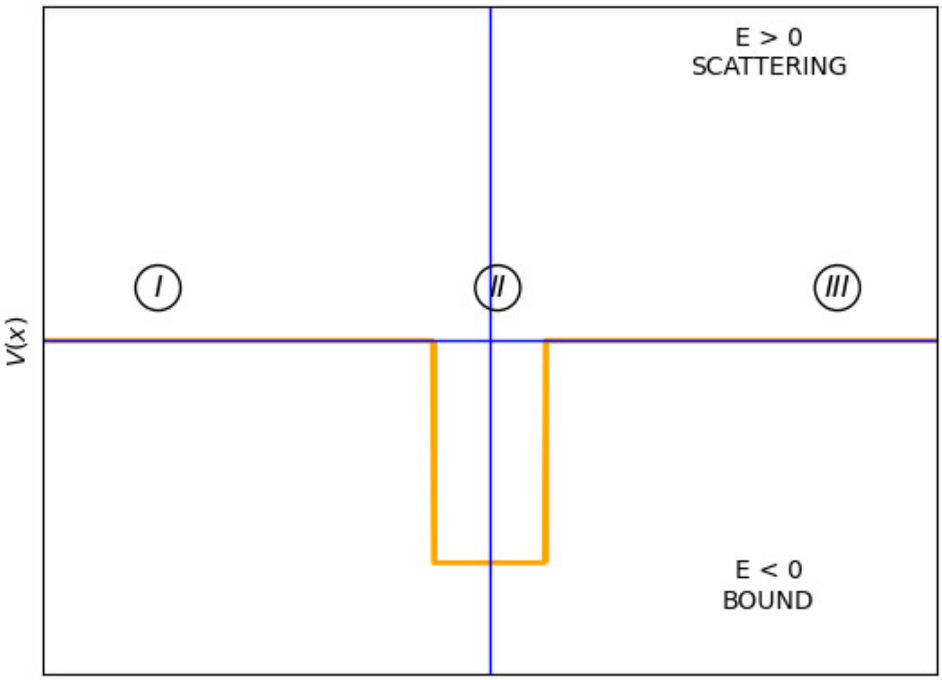

Figure 5: Potential

Where:

$$\begin{aligned} k &= \frac{\sqrt{2mE}}{\hbar} \\ l &= \frac{\sqrt{2m(E+V_0)}}{\hbar} \end{aligned} \tag{46}$$

First we must ensure our 2 boundary conditions, which are that $\psi(x)$ must be continuous and $\frac{d}{dx}\psi(x)$ also has to be continuous, therefore for this statements, we will proceed as:

1. BC1: $\psi(x)$ has to be continious:

$$\begin{aligned} x &= +a \\ \psi(I) &= \psi(II) \\ Fe^{-ka} &= D\cos(la) \end{aligned} \tag{47}$$

2. BC2: $\frac{d}{dx}\psi$ must be continious:

$$\begin{aligned} \frac{d\psi}{dx} &= \begin{cases} -Fke^{-\kappa x} \\ -D\sin(kx) \\ F(-x) \end{cases} \\ -Fke^{-ka} &= -Dl\sin(la) \end{aligned} \tag{48}$$

So, by dividing equation 48, and equation 47, we have that:

$$\begin{aligned} -k &= -l\tan(la) \\ k &= l\tan(la) \end{aligned} \tag{49}$$

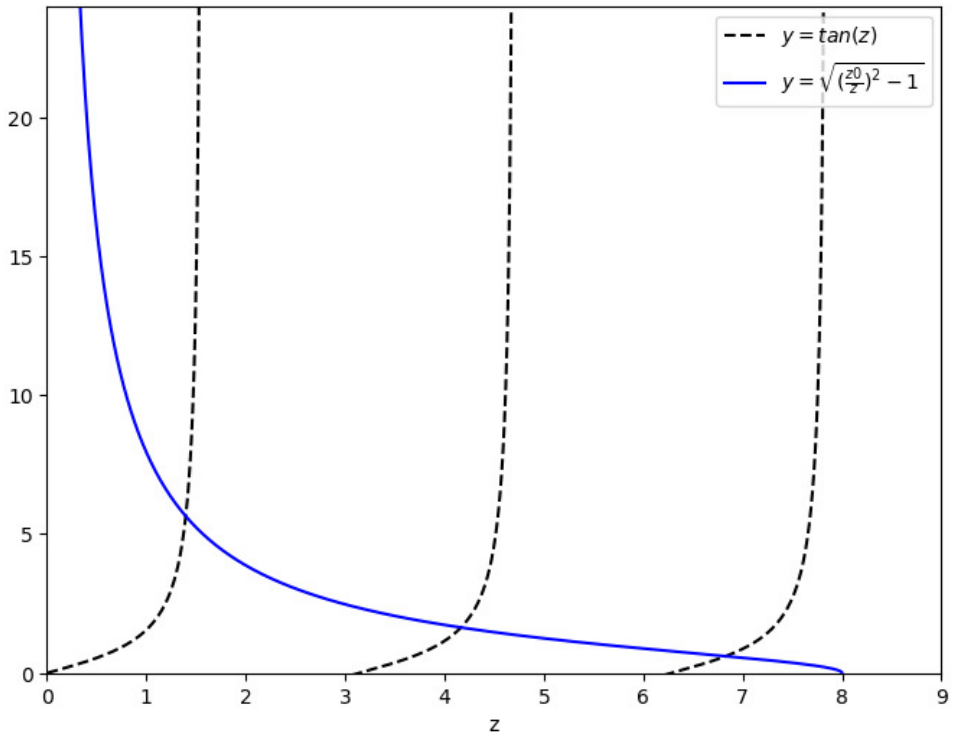

Figure 6: Trascendental Equation

Therefore, to complete the analysis, we must normalize the wave function's PDF ($|\psi(x)|^2$). And since it's an even function, we can state that:

$$\begin{aligned} F^2 \int_0^a \cos^2(lx)dx + B^2 \int_a^\infty e^{-2kx} &= \frac{1}{2} \\ \frac{1}{2l}F^2(\sin(la)\cos(la) + \frac{1}{2l}B^2 e^{-2ka} &= \frac{1}{2} \end{aligned} \tag{50}$$

From the first boundary condition, we can get:

$$F^2 e^{-2ka} = D^2 \cos^2(la) \tag{51}$$

Andm by solving the last equations,and remembering equation (49), we can get:

$$\begin{aligned} F^2 &= \frac{kl}{l\cos^2(la) + k\sin(la)\sin(la) + kla} \\ D^2 &= \frac{k}{1+ka} \\ B^2 &= \frac{kl\cos^2(la)e^{2ka}}{lcos^2(la) + k\sin(la)\cos(la) + kla} \\ B^2 &= \frac{k}{1+ka} cos(la)e^{ka} \end{aligned} \tag{52}$$

1.17 Following the same approach we followed on class, find from scratch the odd bound state wave functions, $\psi(x)$, for the finite square well

Since our complete wave function is:

$$\psi(x) = \begin{cases} Be^{kx} & x < -a \\ C\sin(lx) + D\cos(lx) & -a \leq x \leq a \\ Fe^{-kx} & x > a \end{cases} \tag{53}$$

For odd solutions,which can be stated when: $f(-x) = -f(x)$we have:

$$\psi(x)_{even} = \begin{cases} Fe^{kx} & x < -a \\ C\sin(lx) & -a \leq x \leq a \\ -Fe^{-kx} & x > a \end{cases} \tag{54}$$

1.18 Derive the transcendental equation for the allowed energies of these odd bound states.

Our boundary conditions, state that:

$$\begin{aligned} Be^{-ka} &= -C\sin(la) \\ kBe^{-ka} &= lC\cos(la) \end{aligned} \tag{55}$$

Which leads to a transcendental equation as previously like:

$$\begin{aligned} \frac{1}{k} &= -\frac{1}{l}\tan(la) \\ \frac{2ma^2V_0}{\hbar^2} &= 1 + \frac{\cos^2(z)}{\sin^2(z)} \\ \sqrt{(\frac{Z_0}{Z})^2 - 1} &= \frac{\cos(z)}{\sin(z)} \end{aligned} \tag{56}$$

To normalize, we integrate the PDF ($|\psi(x)|^2$).

$$\begin{aligned} C^2\int_0^a \sin^2(lx)dx + F^2\int_a^\infty e^{-2kx}dx &= \frac{1}{2} \\ -\frac{1}{2l}C^2[\sin(la)\cos(la) - la] + \frac{1}{2k}F^2e^{-2ka} &= \frac{1}{2} \end{aligned} \tag{57}$$

By solving the equations,and remembering the transcendental equation (55), we can get:

$$\begin{aligned} C^2 &= \frac{kl}{l\sin^2(la) - k\sin(la)\cos(la) + kla} \\ C^2 &= \frac{k}{1+ka} \\ F^2 &= \frac{kl\sin^2(la)e^{2ka}}{l\sin^2(la) - k\sin(la)\cos(la) + kla} \\ F^2 &= \frac{k}{1+ka}\sin^2(la)e^{2ka} \end{aligned} \tag{58}$$

Now, since the boundary equations have to be satisfied, we must take the negative root of F, getting finally the normalization constants as:

$$\begin{aligned} C &= \sqrt{\frac{k}{1+ka}} \\ F &= -\sqrt{\frac{k}{1+ka}}\sin(la)e^{ka} \end{aligned} \tag{59}$$

1.19 Solve it graphically and numerically (using your favourite programming tool). Sketch the solutions for different well depths, well widths, and particle masses. Briefly discuss the results.

For our odd transcendental equation, we have:

$$\sqrt{(\frac{Z_0}{Z})^2 - 1} = \frac{\cos(z)}{\sin(z)} \tag{60}$$

Where, to solve it, we will use the next code:

Listing 8: Python Code – Transcendental Function

```
#Plot of odd Transcendental functions
z0 = 8
z = np.arange(0., 10*np.pi, 0.001)

y_iz_odd = 1/np.tan(z)
y_de_odd = np.sqrt((z0/z)**2 -1.)
y_iz_odd[y_iz_odd > +4*z0] = np.nan
y_de_odd[y_de_odd < -4*z0] = np.nan

plt.figure(figsize=(8, 6))
plt.plot(z, y_iz_odd, c='black', linestyle='--', label='y = cot(z)}')
plt.plot(z, y_de_odd, c='blue', label=r'$y = \sqrt{(\frac{z0}{z})^2 - 1}$'
    )
plt.xlim(0, z0 + 1)
plt.ylim(0, 3 * z0)
plt.xlabel('z')
plt.legend()
plt.show()

#Numerical Solve
def f(z, z0):
    opt_f = (np.sqrt((z0/z)**2 -1.) * np.tan(z)) - 1
    return opt_f

r = opt.root(f, [1.,3.,7.], args=z0)
print(r.x)

y1 = 1/np.tan(r.x[1])
```

```
28 y2 = 1/np.tan(r.x[2])
29
30 plt.figure(figsize=(8, 6))
31 plt.plot(z, y_iz_odd, c='black', linestyle='--', label=r'$y = cot(z)$')
32 plt.plot(z, y_de_odd , c='blue', label=r'$y = \sqrt{(\frac{z0}{z})^2 - 1}$'
    )
33 plt.plot(r.x[1], y1, marker = "o", c = 'green')
34 plt.plot(r.x[2], y2, marker = "o", c = 'green')
35 plt.xlim(0, z0 + 1)
36 plt.ylim(0, 3 * z0)
37 plt.xlabel('z')
38 plt.legend()
39 plt.show()
```

Where, the plots we get back are: And the numerical solved roots for this transcendental equation are:

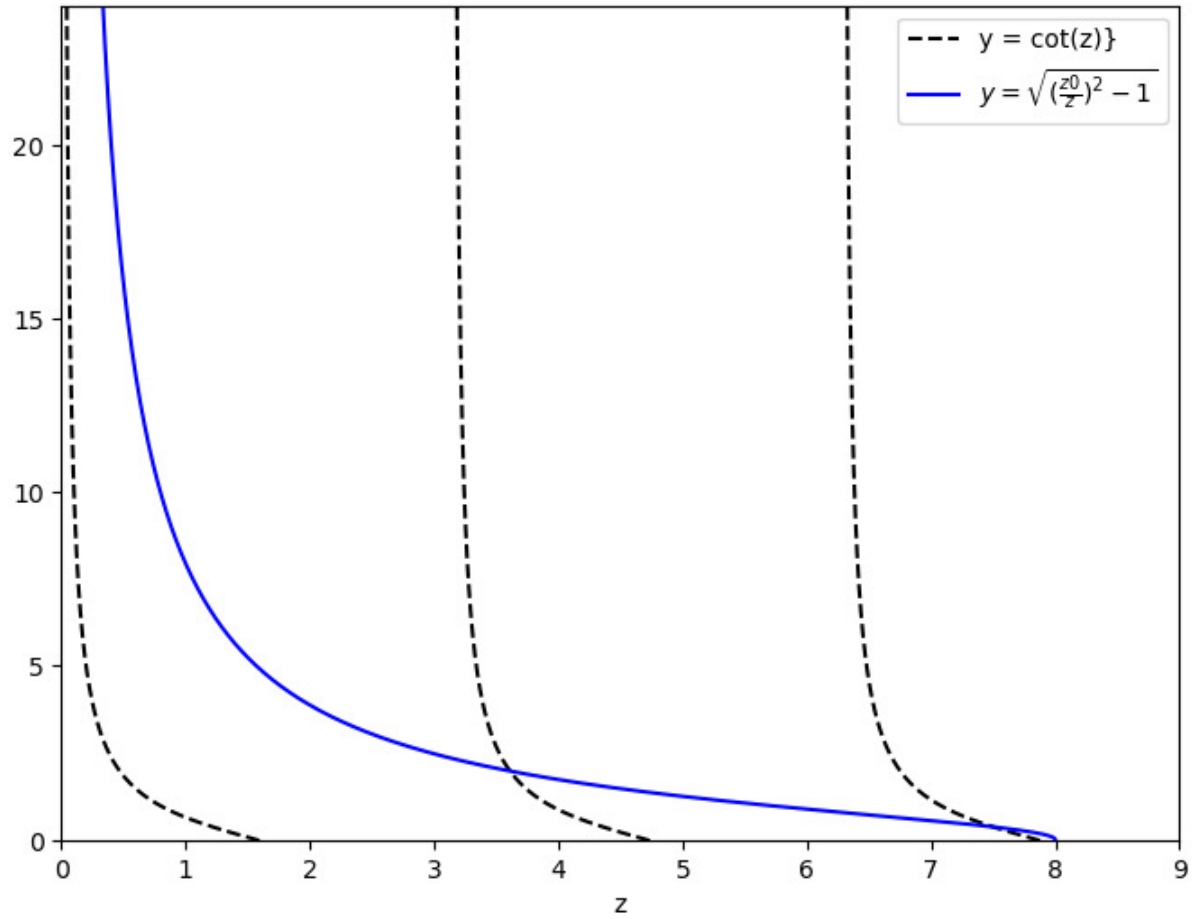

Figure 7: Odd Transcendental Equation Plot

1. 3.60792429
2. 7.58863341

And the plot with our solutions, are:

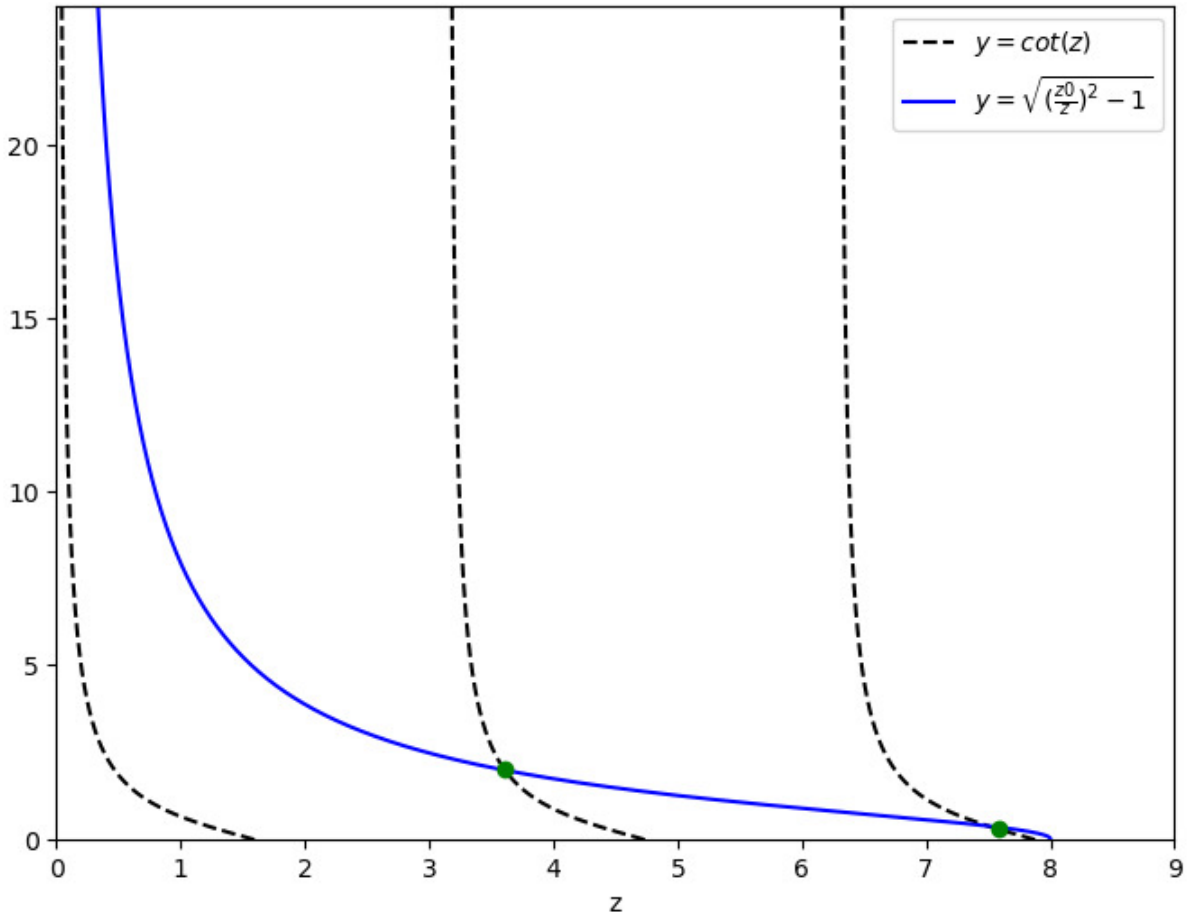

Figure 8: Odd Transcendental Equation Solution Plot

1.20 Study and discuss the two limiting cases and how the energy levels compare to those found for the even bound state wave functions studied in class. Is there always an odd bound state?

1.20.1 Wide, Deep Well Limit ($z_0 \gg 0$)

In this case, we have:

$$z_n = \frac{n\pi}{2} \quad \text{for odd } n$$

The energy levels are given by:

$$E_n + V_0 \approx \frac{n^2\pi^2\hbar^2}{2m(2a)^2} \quad \text{for odd } n = 1, 3, 5, ...$$

Key observations:

- This represents half of the infinite square well energies
- As $V_0 \to \infty$, the finite well approaches the infinite square well
- For any finite V_0, there is a finite number of bound states

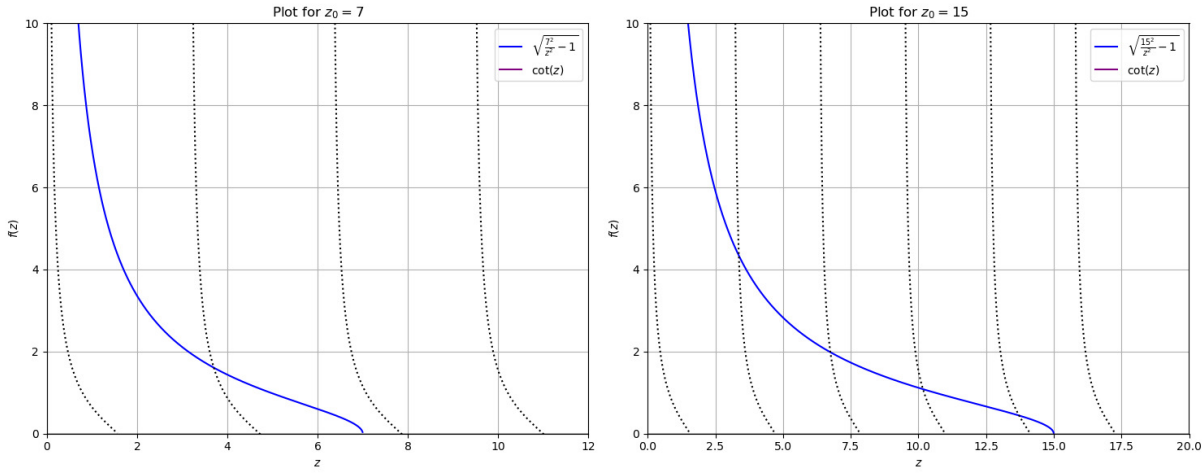

Figure 9: Wide, Deep Well Solutions Plot

1.20.2 Shallow, Narrow Well Limit ($z_0 < \frac{\pi}{2}$)

In this regime:

- Only one bound state exists
- The well is too "weak" to support multiple bound states

Comparison with Even Bound States

The odd bound states complement the even bound states studied in class:

- Even states have symmetric wavefunctions: $\psi(-x) = \psi(x)$
- Odd states have antisymmetric wavefunctions: $\psi(-x) = -\psi(x)$
- Energy levels alternate between even and odd states

1.21 Existence of Odd Bound States

To address whether there is always an odd bound state:

1. For sufficiently deep wells (V_0 large enough), there is always at least one odd bound state.
2. However, there exists a critical well depth below which no odd bound states exist. This occurs when:

$$z_0 < \frac{\pi}{2}$$

Therefore, the answer is no - there is not always an odd bound state. The existence depends on the well depth V_0 and width a.

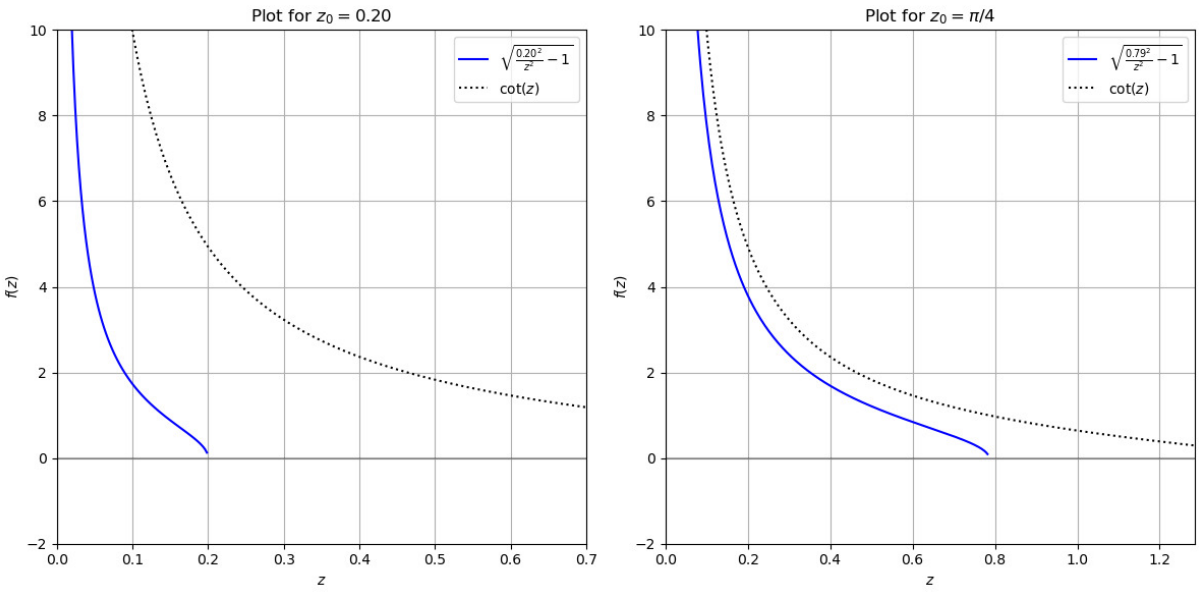

Figure 10: Shallow, Narrow Well Solutions Plot

Listing 9: Python Code – Different Wells Solutions

```
def create_z_arrays(z0):
    # Create arrays suitable for shallow wells
    z_main = np.linspace(0.01, z0 + 1, 500)
    z_cot = np.linspace(0.01, np.pi/2 - 0.01, 500)
    return z_main, z_cot

# Define z0 values for shallow potentials
z0_values = [0.2, np.pi/4]

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

for i, z0 in enumerate(z0_values):
    z_main, z_cot = create_z_arrays(z0)

    # Calculate square root term
    # Only plot where the expression under the square root is positive
    mask = z_main < z0
    y_rhs = np.sqrt((z0**2 / z_main[mask]**2) - 1)

    # Calculate cotangent
    y_cot = np.cos(z_cot) / np.sin(z_cot)

    # Plotting
    axs[i].plot(z_main[mask], y_rhs, c='blue',
                label=rf'$\sqrt{{\frac{{{z0:.2f}^2}}{{z^2}} - 1}}$', ↩
                    linestyle='-')
    axs[i].plot(z_cot, y_cot, c='black', linestyle=':', label=r'$\cot(z)$'↩
```

```
        )

    # Set plot limits and labels
    axs[i].set_xlim(0, z0 + 0.5)
    axs[i].set_ylim(-2, 10)
    axs[i].set_xlabel(r"$z$")
    axs[i].set_ylabel(r"$f(z)$")
    if z0 == np.pi/4:
        title_z0 = r'\pi/4'
    else:
        title_z0 = f'{z0:.2f}'
    axs[i].set_title(f'Plot for $z_0 = {title_z0}$')
    axs[i].legend()
    axs[i].grid(True)

    # Add horizontal and vertical lines at 0
    axs[i].axhline(y=0, color='k', linestyle='-', alpha=0.3)
    axs[i].axvline(x=0, color='k', linestyle='-', alpha=0.3)

plt.tight_layout()
plt.show()


def create_z_arrays(z0):
    # Create separate arrays to handle discontinuities
    z_points = np.linspace(0.01, z0 + 5, 1000)
    z_segments = []
    for n in range(int(z0/np.pi) + 2):
        if n == 0:
            segment = np.linspace(0.01, np.pi/2 - 0.01, 200)
        else:
            segment = np.linspace(n*np.pi + 0.01, (n+1)*np.pi - 0.01, 200)
        z_segments.append(segment)
    return z_points, z_segments

# Define z0 values for different plots
z0_values = [7, 15]

# Create subplots
fig, axs = plt.subplots(1, 2, figsize=(15, 6))

for i, z0 in enumerate(z0_values):
    z_points, z_segments = create_z_arrays(z0)

    # Plot right-hand side (square root term)
    y_rhs = np.sqrt((z0**2 / z_points**2) - 1)
    axs[i].plot(z_points, y_rhs, linestyle='-', color='blue',
```

```
                label=rf'$\sqrt{{\frac{{{z0}^2}}{{z^2}} - 1}}$')

    # Plot left-hand side (cotangent) in segments to handle asymptotes
    for segment in z_segments:
        y_lhs = np.cos(segment) / np.sin(segment)  # cot(z)
        axs[i].plot(segment, y_lhs, c='black', linestyle=':')

    # Add line for cot(z) in legend
    axs[i].plot([], [], c='purple', label=r'$\cot(z)$', linestyle='-')

    # Set plot limits and labels
    axs[i].set_xlim(0, z0 + 5)
    axs[i].set_ylim(0, 10)
    axs[i].set_xlabel(r"$z$")
    axs[i].set_ylabel(r"$f(z)$")
    axs[i].set_title(f'Plot for $z_0 = {z0}$')
    axs[i].legend()
    axs[i].grid(True)

    # Add horizontal and vertical lines at 0
    axs[i].axhline(y=0, color='k', linestyle='-', alpha=0.3)
    axs[i].axvline(x=0, color='k', linestyle='-', alpha=0.3)

plt.tight_layout()
plt.show()
```