

eigenstates for infinite potentialtopf

March 9, 2023

0.1 Elektron im unendlichen Potentialtopf (Programm von chatGPT (März 23))

Eingabe für bot: > Write a Python program to plot the probability density for the $n=1$ and $n=3$ eigenstates of the Hamiltonian for the infinite square potential well in the position and in the momentum representation. Use Fourier transformation to calculate the momentum representation.

```
[1]: import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

# Constants
hbar = 1.054571817e-34 # Planck's constant divided by 2*pi
m = 9.10938356e-31 # Mass of the electron
L = 1e-9 # Length of the well
N = 1000 # Number of points in the grid
x = np.linspace(0, L, N) # Position vector
dx = x[1] - x[0] # Step size in position space

# Eigenfunctions and eigenvalues
n1 = 1
n3 = 3
psi1 = np.sqrt(2/L) * np.sin(n1*np.pi*x/L)
psi3 = np.sqrt(2/L) * np.sin(n3*np.pi*x/L)
E1 = (n1**2 * hbar**2 * np.pi**2)/(2*m*L**2)
E3 = (n3**2 * hbar**2 * np.pi**2)/(2*m*L**2)

# Position representation
plt.figure()
plt.plot(x, np.abs(psi1)**2, label='n=1')
plt.plot(x, np.abs(psi3)**2, label='n=3')
plt.xlabel('Position (m)')
plt.ylabel('Probability density')
plt.title('Position representation')
plt.legend()

# Momentum representation
p = np.linspace(-np.pi*hbar/L, np.pi*hbar/L, N) # Momentum vector
```

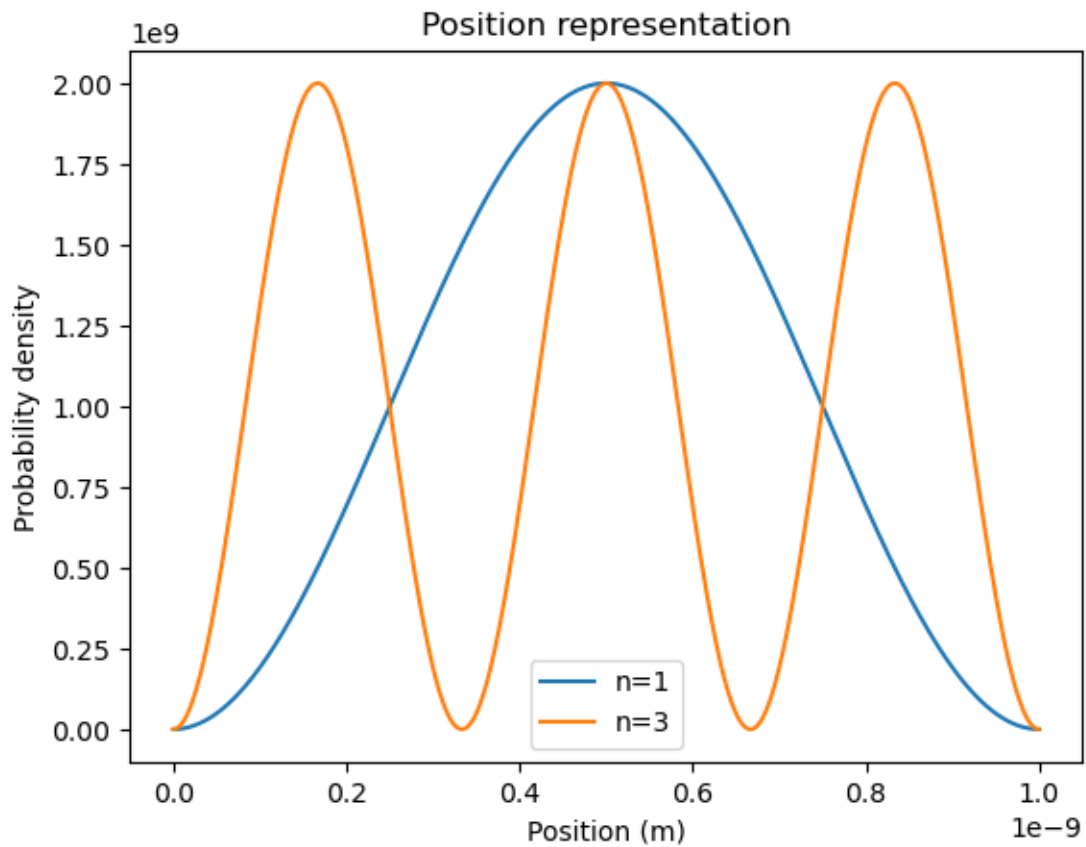
```

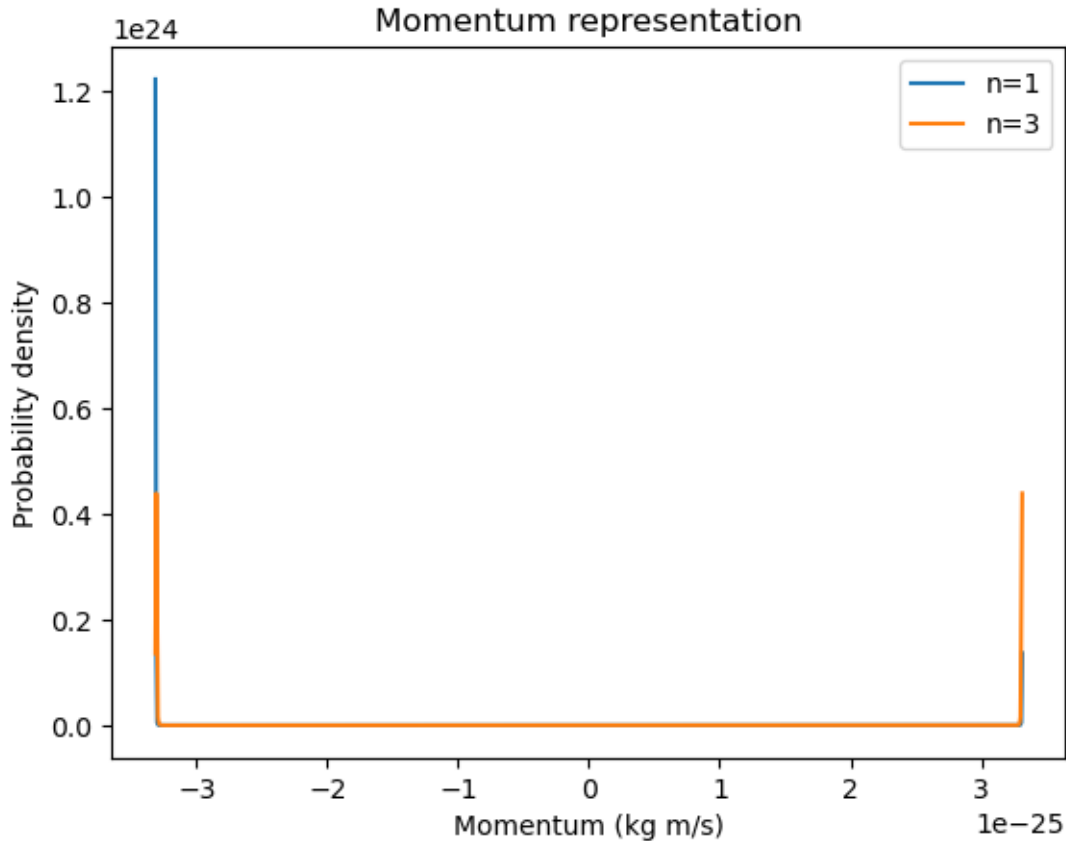
dp = p[1] - p[0]          # Step size in momentum space
phi1 = fft(psi1) * dx / np.sqrt(2*np.pi*hbar)
phi3 = fft(psi3) * dx / np.sqrt(2*np.pi*hbar)

plt.figure()
plt.plot(p, np.abs(phi1)**2, label='n=1')
plt.plot(p, np.abs(phi3)**2, label='n=3')
plt.xlabel('Momentum (kg m/s)')
plt.ylabel('Probability density')
plt.title('Momentum representation')
plt.legend()

plt.show()

```





Explanation (vom chatBot):

- We first import the necessary libraries: numpy for numerical operations, matplotlib for plotting, and scipy.fftpack for Fourier transforms.
- We then define the constants, including Planck's constant divided by 2π , the mass of the electron, the length of the well, the number of points in the grid, the position vector, and the step size in position space.
- Next, we calculate the eigenfunctions and eigenvalues for the $n=1$ and $n=3$ states using the analytical solutions for the infinite square potential well. We also normalize the eigenfunctions by dividing by the square root of the integral of the square of the absolute value of the function.
- We then plot the probability density for the $n=1$ and $n=3$ states in the position representation using `plt.plot` function.
- To obtain the momentum representation, we first create a momentum vector p using the `numpy.linspace` function. We also calculate the step size in momentum space.
- We then perform the Fourier transform of the eigenfunctions using the `scipy.fftpack.fft` function. We also normalize the Fourier transform by multiplying by the step size in position space and dividing by the square root of $2\pi\hbar$.
- Finally, we plot the probability density for the $n=1$

Nun die Frage etwas knapper, unpräziser in Deutsch formuliert: > Schreibe ein Pythonprogramm, das für ein Elektron im unendlich hohen Potentialtopf die Eigenzustände $n=1$, $n=2$ und $n=3$

beschreibt, sowohl in der Orts- als auch der Impulsdarstellung.

```
[1]: import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

# Konstanten
hbar = 1.054571817e-34 # Plancksches Wirkungsquantum / 2*pi
m = 9.10938356e-31     # Masse des Elektrons
L = 1e-9               # Länge des Potentials
N = 1000               # Anzahl der Punkte im Gitter
x = np.linspace(0, L, N) # Ortsvektor
dx = x[1] - x[0]        # Schrittweite im Ortsraum

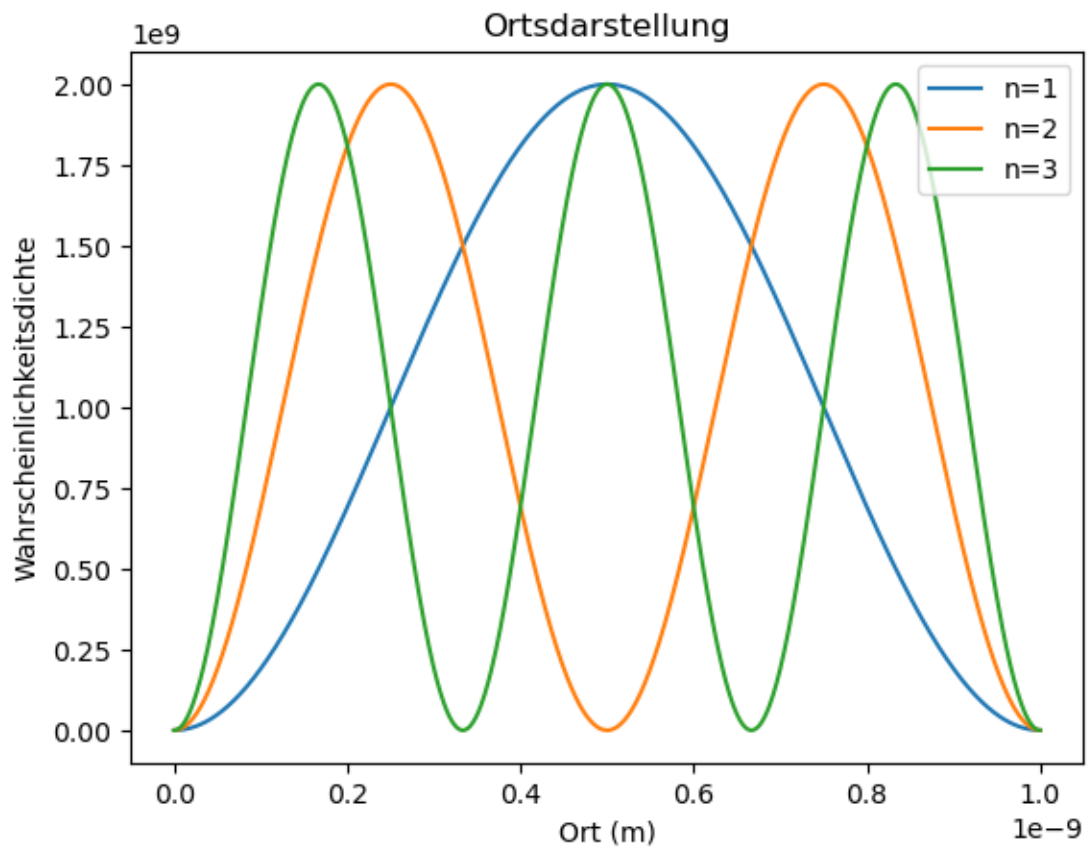
# Eigenfunktionen und Eigenwerte
n1 = 1
n2 = 2
n3 = 3
psi1 = np.sqrt(2/L) * np.sin(n1*np.pi*x/L)
psi2 = np.sqrt(2/L) * np.sin(n2*np.pi*x/L)
psi3 = np.sqrt(2/L) * np.sin(n3*np.pi*x/L)
E1 = (n1**2 * hbar**2 * np.pi**2)/(2*m*L**2)
E2 = (n2**2 * hbar**2 * np.pi**2)/(2*m*L**2)
E3 = (n3**2 * hbar**2 * np.pi**2)/(2*m*L**2)

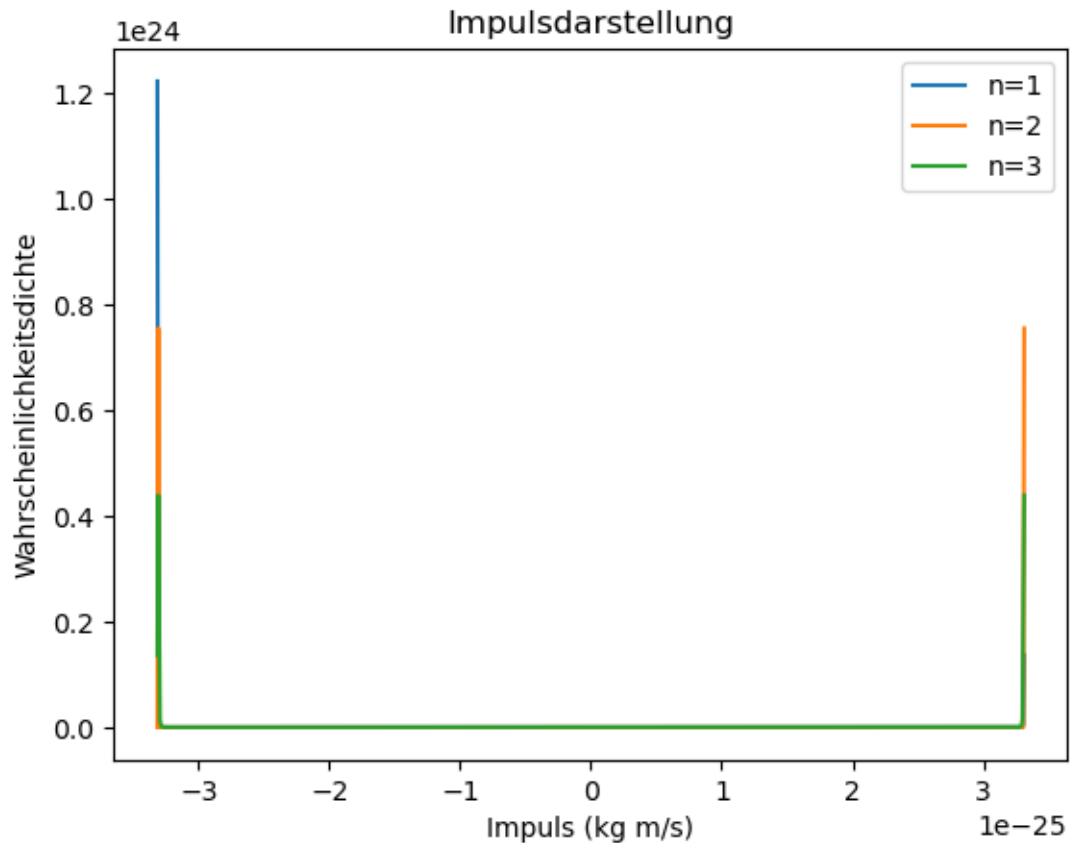
# Ortsdarstellung
plt.figure()
plt.plot(x, np.abs(psi1)**2, label='n=1')
plt.plot(x, np.abs(psi2)**2, label='n=2')
plt.plot(x, np.abs(psi3)**2, label='n=3')
plt.xlabel('Ort (m)')
plt.ylabel('Wahrscheinlichkeitsdichte')
plt.title('Ortsdarstellung')
plt.legend()

# Impulsdarstellung
p = np.linspace(-np.pi*hbar/L, np.pi*hbar/L, N) # Impulsvektor
dp = p[1] - p[0] # Schrittweite im Impulsraum
phi1 = fft(psi1) * dx / np.sqrt(2*np.pi*hbar)
phi2 = fft(psi2) * dx / np.sqrt(2*np.pi*hbar)
phi3 = fft(psi3) * dx / np.sqrt(2*np.pi*hbar)

plt.figure()
plt.plot(p, np.abs(phi1)**2, label='n=1')
plt.plot(p, np.abs(phi2)**2, label='n=2')
plt.plot(p, np.abs(phi3)**2, label='n=3')
plt.xlabel('Impuls (kg m/s)')
```

```
plt.ylabel('Wahrscheinlichkeitsdichte')  
plt.title('Impulsdarstellung')  
plt.legend()  
  
plt.show()
```





Erklärung:

- Wir importieren zunächst die erforderlichen Bibliotheken: numpy für numerische Operationen, matplotlib für die Darstellung von Diagrammen und scipy.fftpack für Fourier-Transformationen.
- Wir definieren dann die Konstanten, einschließlich des reduzierten Planckschen Wirkungsquantums, der Masse des Elektrons, der Länge des Potentials, der Anzahl der Punkte

[]: