import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import trapezoid

# --- PARAMETERS ---

grid\_size = 120

timesteps = 180

dt = 0.12

width = 6

r = np.linspace(0, grid\_size, grid\_size)

dr = r[1] - r[0]

# --- ψ₁: Inbound packet ---

x1 = 25

k1 = 1.4

ψ1 = np.exp(-((r - x1)\*\*2) / (2 \* width\*\*2)) \* np.exp(1j \* k1 \* r)

# --- ψ₃: Phase-mismatched decoy beyond the barrier ---

x3 = 80 # Position beyond barrier

phase\_offset = np.pi / 2

ψ3 = np.exp(-((r - x3)\*\*2) / (2 \* width\*\*2)) \* np.exp(1j \* (k1 \* r + phase\_offset))

# --- Combine wavefunctions ---

ψ = np.array([ψ1, ψ3])

n = len(ψ)

# --- Barrier Potential ---

barrier\_center = 60

barrier\_width = 6

barrier\_height = 0.06

V = np.zeros\_like(r)

V[(r > barrier\_center - barrier\_width/2) & (r < barrier\_center + barrier\_width/2)] = barrier\_height

# --- Evolution Storage ---

ψ\_total = []

snapshot\_times = np.arange(0, timesteps, 12)

ψ\_snapshots = []

# --- Time Evolution ---

for t in range(timesteps):

for i in range(n):

lap = np.zeros\_like(ψ[i], dtype=complex)

lap[1:-1] = (ψ[i][2:] - 2\*ψ[i][1:-1] + ψ[i][:-2]) / dr\*\*2

ψ[i] += dt \* (0.65 \* lap - 0.5 \* V \* ψ[i])

norm = np.sqrt(trapezoid(np.abs(ψ[i])\*\*2, r))

if norm != 0:

ψ[i] /= norm

total = np.sum(ψ, axis=0)

ψ\_total.append(total)

if t in snapshot\_times:

ψ\_snapshots.append(np.abs(total)\*\*2)

# --- Transmission & Reflection ---

final = np.abs(ψ\_total[-1])\*\*2

trans\_region = r > (barrier\_center + barrier\_width/2)

refl\_region = r < (x1 - width\*2)

P\_trans = trapezoid(final[trans\_region], r[trans\_region])

P\_refl = trapezoid(final[refl\_region ], r[refl\_region])

# --- Plot Results ---

fig, axs = plt.subplots(3, 1, figsize=(10,10))

# 1. Evolution

for i, ψs in enumerate(ψ\_snapshots):

axs[0].plot(r, ψs, alpha=0.55, label=f't={snapshot\_times[i]}' if i in [0,len(ψ\_snapshots)-1] else "")

axs[0].plot(r, V / barrier\_height \* np.max([np.max(s) for s in ψ\_snapshots]), 'k--', lw=2, label='Barrier')

axs[0].set\_title("Dispatch Tunneling with Phase-Mismatched Decoy")

axs[0].legend()

# 2. Transmission

axs[1].bar(['Transmission'], [P\_trans], color='lightgreen')

axs[1].set\_ylim(0, 1)

axs[1].set\_ylabel('Probability')

axs[1].set\_title('Transmission Probability')

# 3. Reflection

axs[2].bar(['Reflection'], [P\_refl], color='gold')

axs[2].set\_ylim(0, 1)

axs[2].set\_ylabel('Probability')

axs[2].set\_title('Reflection Probability')

plt.tight\_layout()

plt.show()