import numpy as np

import matplotlib.pyplot as plt

# --- Setup grid and parameters ---

N = 32

phase = np.random.uniform(-np.pi, np.pi, (N, N))

memory = np.zeros\_like(phase)

def place\_anyons(pos\_list, val=+1):

field = np.zeros\_like(phase)

for x, y in pos\_list:

field[x, y] = val

return field

def braid(phase, anyons, order):

"""Perform sequence of braids given by 'order' (list of pairs to swap)"""

new\_phase = phase.copy()

N = phase.shape[0]

for pair in order:

# Swap positions

a1, a2 = anyons[pair[0]], anyons[pair[1]]

# Apply a "vortex" between the pair

for i in range(N):

for j in range(N):

dist1 = np.hypot(i - a1[0], j - a1[1])

dist2 = np.hypot(i - a2[0], j - a2[1])

if dist1 < 6: new\_phase[i, j] += np.pi/2

if dist2 < 6: new\_phase[i, j] -= np.pi/2

# Swap anyon positions

anyons[pair[0]], anyons[pair[1]] = a2, a1

return new\_phase

# Define 4 anyon positions (corners)

anyons\_init = [(5,5), (5,26), (26,5), (26,26)]

# Two different braid orders

braid\_seq1 = [(0,1),(2,3)] # A then B

braid\_seq2 = [(2,3),(0,1)] # B then A

# Run both sequences

phase1 = braid(phase, anyons\_init.copy(), braid\_seq1)

phase2 = braid(phase, anyons\_init.copy(), braid\_seq2)

# "Read out" quantum memory: sum of cosines at core sites

def memory\_readout(phase, anyons):

return np.array([np.cos(phase[x,y]) for x,y in anyons]).sum()

mem1 = memory\_readout(phase1, anyons\_init)

mem2 = memory\_readout(phase2, anyons\_init)

# --- Plotting ---

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)

plt.imshow(np.cos(phase1), cmap='twilight')

plt.title(f'Braiding Order: A then B\nMemory: {mem1:.3f}')

plt.scatter(\*zip(\*anyons\_init), c='cyan', s=100, marker='o', edgecolors='w')

plt.subplot(1,2,2)

plt.imshow(np.cos(phase2), cmap='twilight')

plt.title(f'Braiding Order: B then A\nMemory: {mem2:.3f}')

plt.scatter(\*zip(\*anyons\_init), c='magenta', s=100, marker='o', edgecolors='w')

plt.suptitle('MBT Double Braid: Non-Abelian Logic Demo')

plt.tight\_layout()

plt.show()

print(f"Memory after A then B: {mem1:.4f}")

print(f"Memory after B then A: {mem2:.4f}")

print(f"Difference (non-Abelian logic): {abs(mem1-mem2):.4e}")