import numpy as np

import matplotlib.pyplot as plt

# Grid setup

N = 32

timesteps = 100

memory = np.zeros((N,N))

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

# Insert braided vortex

y, x = np.ogrid[:N, :N]

center = (N//2, N//2)

r = np.sqrt((x-center[1])\*\*2 + (y-center[0])\*\*2)

vortex = np.angle((x-center[1]) + 1j\*(y-center[0]))

phase += vortex

# Evolve with memory, store a topological event at t=40 (braid halfway)

braid\_time = 40

braid\_phase = np.zeros\_like(phase)

order\_history = []

error\_sites = []

for t in range(timesteps):

# Evolve: MBT memory as running average of cos(phase)

memory = 0.95\*memory + 0.05\*np.cos(phase)

# Topological memory: overlap with braided region

mask = (r < 12) & (r > 8)

braid\_phase[mask] = phase[mask]

order = np.abs(np.mean(np.cos(braid\_phase[mask] - np.mean(braid\_phase[mask]))))

order\_history.append(order)

# Add random errors ("hits") in memory after braid halfway

if t == braid\_time+10:

error = (np.random.randint(0,N,8), np.random.randint(0,N,8))

memory[error] = np.random.uniform(-1,1,8)

error\_sites.append(error)

# Local “healing” - MBT topological memory corrects errors gradually

if t > braid\_time+10:

memory += 0.01\*np.cos(phase-braid\_phase)

# Phase diffusion: keep realistic decoherence

phase += 0.08\*np.random.randn(N,N)

# Plot

plt.figure(figsize=(14,4))

plt.subplot(1,3,1)

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

plt.title("Final MBT Phase Field (cos)")

plt.axis('off')

plt.subplot(1,3,2)

plt.imshow(memory, cmap='inferno')

if error\_sites:

ey, ex = error\_sites[-1]

plt.scatter(ex, ey, color='cyan', s=12, label='Error Hits')

plt.legend(loc='lower left')

plt.title("Final MBT Memory Field\n(errors/healing)")

plt.axis('off')

plt.subplot(1,3,3)

plt.plot(order\_history, label="Topological Memory")

plt.axvline(braid\_time, ls='--', color='gray', label='Braid Event')

plt.axvline(braid\_time+10, ls=':', color='red', label='Errors Added')

plt.xlabel("Timestep")

plt.ylabel("Order (memory)")

plt.legend()

plt.title("MBT Topological Quantum Memory\n(With Error Correction)")

plt.tight\_layout()

plt.show()