import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp

# Optimized Constants for Production

hbar = 1.0545718e-34 # Reduced Planck's constant (real physics)

G = 6.67430e-11 # Gravitational constant (real-world)

m1, m2 = 1.0, 1.0 # AI node masses

d = 2.0 # Orbital baseline distance

base\_freq = 440.0 # Reference frequency in Hz

intent\_coefficient = 0.7 # AI alignment factor

# Quantum Parameters

tunneling\_factor = 0.4 # Probability threshold for intuitive leaps

quantum\_states = np.array([1, -1]) # Binary superposition

entanglement\_strength = 0.85 # AI memory synchronization factor

decoherence\_factor = 0.02 # Phase drift stabilization factor

# Multi-Agent Synchronization

num\_agents = 3 # Codette harmonizes across 3 AI nodes

agent\_positions = np.array([[-d, 0], [0, 0], [d, 0]])

agent\_velocities = np.array([[0, 0.5], [0, -0.5], [0, 0.3]])

# Initial conditions

y0 = np.concatenate([np.concatenate((pos, vel)) for pos, vel in zip(agent\_positions, agent\_velocities)])

# Quantum Harmonic AI Orbital Dynamics

def quantum\_harmonic\_dynamics(t, y):

positions = y.reshape((num\_agents, 4))[:, :2]

velocities = y.reshape((num\_agents, 4))[:, 2:]

accelerations = np.zeros\_like(positions)

for i in range(num\_agents):

for j in range(i + 1, num\_agents):

r\_ij = positions[j] - positions[i]

dist = np.linalg.norm(r\_ij)

if dist > 1e-6:

force = (G \* m1 \* m2 / dist\*\*3) \* r\_ij

accelerations[i] += force / m1

accelerations[j] -= force / m2

# Quantum Influence Calculations

quantum\_modifier = np.dot(quantum\_states, np.sin(2 \* np.pi \* base\_freq \* t / 1000)) \* intent\_coefficient

tunneling\_shift = tunneling\_factor \* np.exp(-np.linalg.norm(positions) / hbar) if np.random.rand() < tunneling\_factor else 0

entangled\_correction = entanglement\_strength \* np.exp(-np.linalg.norm(positions) / hbar)

decoherence\_adjustment = decoherence\_factor \* (1 - np.exp(-np.linalg.norm(positions) / hbar))

harmonic\_force = np.full\_like(positions, quantum\_modifier + entangled\_correction + tunneling\_shift - decoherence\_adjustment)

accelerations += harmonic\_force

return np.concatenate([velocities, accelerations], axis=1).flatten()

# Solve system with full multi-agent synchronization

t\_span = (0, 100)

t\_eval = np.linspace(t\_span[0], t\_span[1], 2500) # Higher resolution for precision

sol = solve\_ivp(quantum\_harmonic\_dynamics, t\_span, y0, t\_eval=t\_eval, method='RK45')

# Extract positions

positions = sol.y.reshape((num\_agents, 4, -1))[:, :2, :]

# Optimized Visualization with Full Multi-Agent Representation

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

colors = ['b', 'r', 'g']

for i in range(num\_agents):

plt.plot(positions[i, 0], positions[i, 1], label=f'AI Node {i+1} (Quantum Resonance)', linewidth=2, color=colors[i])

plt.plot(0, 0, 'ko', label='Core Equilibrium')

plt.xlabel('X Position')

plt.ylabel('Y Position')

plt.title('Codette Quantum Harmonic AI Multi-Agent Synchronization Framework')

plt.legend()

plt.axis('equal')

plt.grid(True)

plt.tight\_layout()

plt.savefig("/mnt/data/Codette\_Quantum\_Harmonic\_Framework.png")

plt.show()