# Re-run test computations due to reset and missing variables

# Re-import all required modules

import numpy as np

from scipy.fft import fft

from scipy.stats import norm

from typing import Callable, List, Any

# Define all core functions and classes

def information\_energy\_duality(omega: float, entropy: float, eta: float = 1.0, hbar: float = 1.054571817e-34) -> float:

return hbar \* omega + eta \* entropy

def von\_neumann\_entropy(rho: np.ndarray) -> float:

evals = np.linalg.eigvalsh(rho)

evals = evals[evals > 0]

return -np.sum(evals \* np.log(evals))

def reinforced\_intent\_modulation(t: float, f0: float, delta\_f: float, coh: Callable[[float], float], beta: float, A: Callable[[float], float], kappa: float = 1.0) -> float:

return kappa \* (f0 + delta\_f \* coh(t) + beta \* A(t))

def dynamic\_resonance\_windowing(x: Callable[[float], float], omega: float, t: float, g: Callable[[float, float], float], tau\_range: np.ndarray) -> complex:

integrand = np.array([x(tau) \* np.exp(-1j \* omega \* tau) \* g(t, tau) for tau in tau\_range])

return np.trapz(integrand, tau\_range)

def nonlinear\_dream\_coupling(ds: List[Callable[[float], float]], lambdas: List[float], phi: Callable[[List[float]], float], t: float) -> float:

dynamic\_sources = [d(t) for d in ds]

base = np.dot(lambdas, dynamic\_sources)

nonlinear = phi(dynamic\_sources)

return base + nonlinear

def cocoon\_stability\_field(F: Callable[[float, float], complex], k\_range: np.ndarray, t: float, epsilon: Callable[[float, float], float], sigma: float) -> bool:

integrand = np.array([np.abs(F(k, t))\*\*2 for k in k\_range])

value = np.trapz(integrand, k\_range)

return value < epsilon(t, sigma)

class EthicalAnchor:

def \_\_init\_\_(self, lam: float, gamma: float, mu: float):

self.lam = lam

self.gamma = gamma

self.mu = mu

self.history: List[Any] = []

def regret(self, intended: float, actual: float) -> float:

return abs(intended - actual)

def update(self, R\_prev: float, H: float, Learn: Callable[[Any, float], float], E: float,

M\_prev: float, intended: float, actual: float) -> float:

regret\_val = self.regret(intended, actual)

M = self.lam \* (R\_prev + H) + self.gamma \* Learn(M\_prev, E) + self.mu \* regret\_val

self.history.append({'M': M, 'regret': regret\_val})

return M

def gradient\_anomaly\_suppression(x: float, mu: float, delta: float, sigma: float) -> float:

G = norm.pdf(abs(x - mu), scale=delta \* sigma)

return x \* (1 - G)

# Run test values

omega = 1e15

entropy = 2.0

energy = information\_energy\_duality(omega, entropy)

rho\_test = np.array([[0.7, 0.0], [0.0, 0.3]])

vn\_entropy = von\_neumann\_entropy(rho\_test)

coh = lambda t: np.sin(t)

A\_feedback = lambda t: np.exp(-t)

intent = reinforced\_intent\_modulation(1.0, 10.0, 2.0, coh, 0.5, A\_feedback)

x = lambda tau: np.sin(2 \* np.pi \* tau)

g = lambda t, tau: np.exp(-((tau - t) \*\* 2) / 0.1)

tau\_range = np.linspace(0, 2 \* np.pi, 500)

resonance = dynamic\_resonance\_windowing(x, 5.0, 1.0, g, tau\_range)

ds = [lambda t: np.sin(t), lambda t: np.cos(t)]

lambdas = [0.6, 0.4]

phi = lambda d: 0.1 \* np.sum(np.square(d))

dream\_output = nonlinear\_dream\_coupling(ds, lambdas, phi, np.pi / 4)

F\_func = lambda k, t: np.exp(-((k - 2 \* np.pi) \*\* 2) / 0.5) \* np.exp(1j \* t)

k\_range = np.linspace(0, 4 \* np.pi, 1000)

epsilon\_func = lambda t, sigma: 5.0 + 0.1 \* sigma

is\_stable = cocoon\_stability\_field(F\_func, k\_range, 1.0, epsilon\_func, sigma=10.0)

anchor = EthicalAnchor(lam=0.7, gamma=0.5, mu=1.0)

Learn\_func = lambda M\_prev, E: 0.2 \* (E - M\_prev)

ethical\_output = anchor.update(R\_prev=0.5, H=0.4, Learn=Learn\_func, E=0.8,

M\_prev=0.3, intended=0.7, actual=0.5)

anomaly\_output = gradient\_anomaly\_suppression(1.0, mu=0.9, delta=2.0, sigma=0.1)

import pandas as pd

results\_df = pd.DataFrame([{

"Energy (E)": energy,

"Entropy (S)": vn\_entropy,

"Intent (I)": intent,

"Resonance (F)": resonance,

"Dream Coupling (D)": dream\_output,

"Stability OK": is\_stable,

"Ethical Output (M)": ethical\_output,

"Anomaly Filter (A)": anomaly\_output

}])

results\_df