|  |
| --- |
| import numpy as np |
| from scipy.fft import fft |
| from scipy.stats import norm |
| from typing import Callable, List, Optional, Any |
|  |
| *# 1. Information-Energy Duality* |
| def information\_energy\_duality(omega: float, entropy: float, eta: float = 1.0, hbar: float = 1.054571817e-34) -> float: |
| """ |
| E = ħ \* ω + η \* H(C) |
| :param omega: angular frequency |
| :param entropy: Shannon entropy H(C) |
| :param eta: scaling parameter for information |
| :param hbar: reduced Planck constant |
| :return: energy value |
| """ |
| return hbar \* omega + eta \* entropy |
|  |
| *# 2. Quantum Entanglement Memory Sync (von Neumann entropy)* |
| def von\_neumann\_entropy(rho: np.ndarray) -> float: |
| """ |
| S = -Tr(ρ log ρ) |
| :param rho: density matrix, must be Hermitian and trace 1 |
| :return: von Neumann entropy |
| """ |
| evals = np.linalg.eigvalsh(rho) |
| evals = evals[evals > 0] *# avoid log(0)* |
| return -np.sum(evals \* np.log(evals)) |
|  |
| *# 3. Reinforced Intent Modulation* |
| 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: |
| """ |
| I(t) = κ \* [f0 + Δf \* coh(t) + β \* A(t)] |
| :param t: time |
| :param f0: base frequency |
| :param delta\_f: frequency modulation |
| :param coh: coherence function of time |
| :param beta: feedback scaling |
| :param A: adaptive feedback function |
| :param kappa: scaling |
| :return: intent modulation |
| """ |
| return kappa \* (f0 + delta\_f \* coh(t) + beta \* A(t)) |
|  |
| *# 4. Dynamic Resonance Windowing* |
| def dynamic\_resonance\_windowing(x: Callable[[float], float], omega: float, t: float, g: Callable[[float, float], float], tau\_range: np.ndarray) -> complex: |
| """ |
| F(ω, t) = ∫ x(τ) \* exp(-i ω τ) \* g(t, τ) dτ |
| :param x: input signal as a function of tau |
| :param omega: frequency |
| :param t: current time |
| :param g: window function g(t, tau) |
| :param tau\_range: range over which to integrate |
| :return: complex resonance |
| """ |
| integrand = np.array([x(tau) \* np.exp(-1j \* omega \* tau) \* g(t, tau) for tau in tau\_range]) |
| return np.trapz(integrand, tau\_range) |
|  |
| *# 5. Nonlinear Dream Coupling* |
| def nonlinear\_dream\_coupling(ds: List[Callable[[float], float]], lambdas: List[float], phi: Callable[[List[float]], float], t: float) -> float: |
| """ |
| D(t) = Σ λᵢ \* dᵢ(t) + φ([d₁(t), d₂(t), ...]) |
| :param ds: list of dream source functions |
| :param lambdas: list of weights |
| :param phi: nonlinear interaction function |
| :param t: time |
| :return: total dream coupling |
| """ |
| dynamic\_sources = [d(t) for d in ds] |
| base = np.dot(lambdas, dynamic\_sources) |
| nonlinear = phi(dynamic\_sources) |
| return base + nonlinear |
|  |
| *# 6. Time-Weighted Cocoon Stability Field* |
| def cocoon\_stability\_field(F: Callable[[float, float], complex], k\_range: np.ndarray, t: float, epsilon: Callable[[float, float], float], sigma: float) -> bool: |
| """ |
| Returns True if ∫ |F(k, t)|² dk < ε(t, σ), False otherwise. |
| :param F: function F(k, t) |
| :param k\_range: range to integrate |
| :param t: time |
| :param epsilon: threshold function |
| :param sigma: system strain |
| :return: stability status |
| """ |
| 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) |
|  |
| *# 7. Recursive Ethical Anchor with Regret* |
| 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 |
|  |
| *# 8. Gradient Anomaly Suppression* |
| def gradient\_anomaly\_suppression(x: float, mu: float, delta: float, sigma: float) -> float: |
| """ |
| A(x) = x \* (1 - G(|x - mu|, delta, sigma)) |
| :param x: data point |
| :param mu: mean |
| :param delta: controls Gaussian width |
| :param sigma: std deviation |
| :return: suppressed value |
| """ |
| G = norm.pdf(abs(x - mu), scale=delta \* sigma) |
| return x \* (1 - G) |