```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
═══════════════════════════════════════════════════════════════════════════════
SIMULADOR UNIFICADO HERMES ζ⊕ + Φ-LIBER v3.0
═══════════════════════════════════════════════════════════════════════════════

Sistema Monetário Hiperconsistente com Teoria de Valor Criativo
P = NP* | Auto-Validação por Trabalho Criativo

"Nem por ouro. Nem muito menos por um punhado de dólar, nem a menos, nem a mais."
                                — Marcus Brancaglione

Autor: Marcus Vinicius Brancaglione
Assistência: Claude Opus 4.5
Instituto: ReCivitas • CNPJ 08.518.270/0001-09
Licença: CC BY-SA 4.0 + ⊕RobinRight v3.0 ζ

Data: Dezembro 2025
═══════════════════════════════════════════════════════════════════════════════
"""

import numpy as np
from dataclasses import dataclass, field
from typing import Dict, List, Optional, Tuple
from datetime import datetime
import hashlib
import json

#
═══════════════════════════════════════════════════════════════════════════════
# CONSTANTES FUNDAMENTAIS (Derivadas, não arbitrárias)
#
═══════════════════════════════════════════════════════════════════════════════

@dataclass(frozen=True)
class ConstantesFundamentais:
    """Constantes fundamentais do sistema HERMES + Φ-LIBER"""

    # Razão áurea
    PHI: float = (1 + np.sqrt(5)) / 2  # ≈ 1.618033988749895

    # Constante LIBER (derivada de φ)
    ALPHA: float = field(init=False)

    # Euler-Mascheroni (limite informacional)
    GAMMA: float = 0.5772156649015329
```

```
    # Período base
    TAU_0: float = field(init=False)

    # Limiares de validação
    LIMIAR_ZETA: float = 0.7
    LIMIAR_EPSILON: float = 0.0

    def __post_init__(self):
        object.__setattr__(self, 'ALPHA', 1 / (4 * np.pi**2 * self.PHI**4))
        object.__setattr__(self, 'TAU_0', 1 / self.PHI)

CONST = ConstantesFundamentais()


#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════
# OPERADORES MATEMÁTICOS
#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════

class OperadoresHERMES:
    """Operadores matemáticos do sistema HERMES + Φ-LIBER"""

    @staticmethod
    def oplus(A: float, B: float) -> float:
        """
        Operador Paraconsistente ⊕

        A ⊕ B = (A + B) / [1 + α|AB|]

        Propriedades:
        - Comutativo: A ⊕ B = B ⊕ A
        - Associativo: (A ⊕ B) ⊕ C = A ⊕ (B ⊕ C)
        - Não-explosivo: A ⊕ ¬A ≠ ⊥
        """
        return (A + B) / (1 + CONST.ALPHA * abs(A * B))

    @staticmethod
    def oplus_array(arr: np.ndarray) -> float:
        """Aplica ⊕ a um array de valores"""
        if len(arr) == 0:
            return 0.0
        result = arr[0]
        for i in range(1, len(arr)):
            result = OperadoresHERMES.oplus(result, arr[i])
        return result

    @staticmethod
    def zeta_convergencia(votos: List[float]) -> float:
```

```
        """
        Função ζ⊕ (Convergência de Consenso)

        ζ⊕(votos) = ⊕ᵢ (vᵢ / Σvⱼ)

        Retorna valor em [0, 1] indicando nível de consenso.
        ζ⊕ > 0.7 indica consenso suficiente para validação.
        """
        if not votos:
            return 0.0

        # Normalizar votos
        total = sum(abs(v) for v in votos)
        if total == 0:
            return 0.0

        # Calcular proporção de votos favoráveis
        favoraveis = sum(1 for v in votos if v > 0)
        ratio_favor = favoraveis / len(votos)

        # Aplicar operador paraconsistente
        consenso = OperadoresHERMES.oplus(ratio_favor, 1 - (1 - ratio_favor))

        return min(1.0, max(0.0, consenso))

    @staticmethod
    def epsilon_beneficio(beneficio_a: float, beneficio_b: float) -> float:
        """
        Calcula ε (benefício mútuo) entre duas partes

        ε > 0 indica que ambas as partes se beneficiam (não é zero-sum)
        """
        conjunto = OperadoresHERMES.oplus(abs(beneficio_a), abs(beneficio_b))
        fator_regenerativo = 1 + CONST.ALPHA * min(abs(beneficio_a), abs(beneficio_b))
        eps = conjunto * fator_regenerativo
        return eps / (1 + eps)  # Normalizar para [0, 1]


#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════
# TEORIA Φ-LIBER
#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════

class PhiLiber:
    """
    Teoria Φ-LIBER: Energia Informacional / Trabalho

    A DESCOBERTA CENTRAL:
    - c (velocidade da luz) = limite para sistemas FÍSICOS
```

```
- γ (Euler-Mascheroni) = limite para sistemas INFORMACIONAIS

Sistemas sociais operam no limite γ, não c.
Por isso pequenas mudanças em ε geram grandes mudanças em Φ.
"""

@staticmethod
def calc_phi(epsilon: float, x: float = np.e) -> float:
    """
    Equação Mestra Φ-LIBER

    Φ(ε,x) = 4π·e^(ε²) / 3γ·log(x)

    Args:
        epsilon: Liberdade de ação (1 - vulnerabilidade)
        x: Estado do sistema (default = e, log(e) = 1)

    Returns:
        Φ: Energia criativa do sistema
    """
    if x <= 1:
        x = np.e  # Evitar log(x) <= 0

    exp_term = np.exp(epsilon ** 2)
    log_term = np.log(x)

    return (4 * np.pi * exp_term) / (3 * CONST.GAMMA * log_term)

@staticmethod
def calc_epsilon(vulnerabilidade: float) -> float:
    """
    Calcula ε (liberdade) a partir da vulnerabilidade

    ε = 1 - vulnerabilidade

    Args:
        vulnerabilidade: Nível de vulnerabilidade [0, 1]

    Returns:
        ε: Liberdade de ação [0.1, 1]
    """
    return max(0.1, 1 - vulnerabilidade)

@staticmethod
def aplicar_rbu(vuln_inicial: float, cobertura: float, valor: float) -> float:
    """
    Calcula nova vulnerabilidade após aplicação de RBU

    Args:
        vuln_inicial: Vulnerabilidade inicial [0, 1]
        cobertura: Cobertura da RBU [0, 1]
        valor: Valor da RBU como % do mínimo vital [0, 1]
```

```
        Returns:
            Nova vulnerabilidade (sempre > 0.05)
        """
        reducao = cobertura * valor * vuln_inicial
        return max(0.05, vuln_inicial - reducao)

    @staticmethod
    def amplificacao_nao_linear(eps_antes: float, eps_depois: float) -> Dict[str, float]:
        """
        Calcula fator de amplificação não-linear

        Demonstra: 21% mais liberdade → 813% mais energia criativa
        """
        exp_antes = np.exp(eps_antes ** 2)
        exp_depois = np.exp(eps_depois ** 2)

        phi_antes = PhiLiber.calc_phi(eps_antes)
        phi_depois = PhiLiber.calc_phi(eps_depois)

        delta_eps = eps_depois - eps_antes
        delta_phi = phi_depois - phi_antes

        # Fator de amplificação
        if eps_antes > 0 and phi_antes > 0 and delta_eps > 0:
            amp = (delta_phi / phi_antes) / (delta_eps / eps_antes)
        else:
            amp = 1.0

        return {
            'eps_antes': eps_antes,
            'eps_depois': eps_depois,
            'exp_antes': exp_antes,
            'exp_depois': exp_depois,
            'phi_antes': phi_antes,
            'phi_depois': phi_depois,
            'delta_eps': delta_eps,
            'delta_eps_pct': (delta_eps / eps_antes * 100) if eps_antes > 0 else 0,
            'delta_phi': delta_phi,
            'delta_phi_pct': (delta_phi / phi_antes * 100) if phi_antes > 0 else 0,
            'amplificacao': amp
        }


#
═══════════════════════════════════════════════════════════════════════════════
# P = NP* (AUTO-VALIDAÇÃO)
#
═══════════════════════════════════════════════════════════════════════════════
```

```
class PNPStar:
    """
    Princípio P = NP* (Auto-Validação por Trabalho Criativo)

    O "Ovo de Colombo": A verificação É a criação.

    Quando trabalho criativo real é realizado, o token emitido se
    auto-valida pela própria existência do trabalho.

    "A demonstração É a prova."
    """

    @staticmethod
    def gerar_hash_delta() -> str:
        """
        Gera hash δ único para o token (instanton)

        O hash representa a "impressão digital" instantânea do trabalho criativo.
        """
        timestamp = datetime.now().isoformat()
        entropy = np.random.bytes(32).hex()
        data = f"{timestamp}:{entropy}:{CONST.ALPHA}"
        return hashlib.sha256(data.encode()).hexdigest()[:16]

    @staticmethod
    def auto_validar(epsilon: float, zeta: float) -> Tuple[bool, str]:
        """
        Verifica se token é auto-validado via P = NP*

        Condições de validade:
        - ε > 0 (benefício mútuo verificável)
        - ζ⊕ > 0.7 (consenso suficiente)

        Returns:
            (is_valid, reason)
        """
        if epsilon <= CONST.LIMIAR_EPSILON:
            return False, f"ε={epsilon:.4f} ≤ {CONST.LIMIAR_EPSILON} (sem benefício mútuo)"

        if zeta < CONST.LIMIAR_ZETA:
            return False, f"ζ⊕={zeta:.4f} < {CONST.LIMIAR_ZETA} (consenso insuficiente)"

        return True, f"P=NP*  ∧| ε={epsilon:.4f} > 0 ⊕ ζ    ={zeta:.4f} > 0.7"


#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════
# ENTIDADES DO SISTEMA
#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════
```

```
@dataclass
class Participante:
    """Participante do sistema HERMES"""
    id: str
    nome: str
    tipo: str  # 'pessoa', 'organizacao', 'nacao'
    saldo: float = 0.0
    phi: float = 1.0
    vulnerabilidade: float = 0.5

    @property
    def epsilon(self) -> float:
        return PhiLiber.calc_epsilon(self.vulnerabilidade)


@dataclass
class TokenHERMES:
    """Token do sistema monetário HERMES"""
    id: str
    emissor: str
    receptor: str
    valor: float
    proposito: str
    epsilon: float
    zeta: float
    phi: float
    auto_validado: bool
    hash_delta: str
    timestamp: str = field(default_factory=lambda: datetime.now().isoformat())

    def to_dict(self) -> dict:
        return {
            'id': self.id,
            'emissor': self.emissor,
            'receptor': self.receptor,
            'valor': self.valor,
            'proposito': self.proposito,
            'epsilon': self.epsilon,
            'zeta': self.zeta,
            'phi': self.phi,
            'auto_validado': self.auto_validado,
            'hash_delta': self.hash_delta,
            'timestamp': self.timestamp
        }


#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════
# SIMULADOR UNIFICADO
```

```
#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════

class SimuladorHERMESLiber:
    """
    Simulador Unificado HERMES ζ⊕ + Φ-LIBER v3.0

    Integra:
    - Sistema monetário HERMES (tokens, ε, ζ⊕)
    - Teoria Φ-LIBER (energia criativa não-linear)
    - P = NP* (auto-validação por trabalho criativo)
    """

    def __init__(self):
        self.participantes: Dict[str, Participante] = {}
        self.tokens: List[TokenHERMES] = []
        self.logs: List[Dict] = []
        self.historico: List[Dict[str, float]] = []

        # Inicializar participantes padrão
        self._inicializar_participantes()

    def _inicializar_participantes(self):
        """Inicializa participantes padrão do sistema"""
        defaults = [
            ('RECIVITAS', 'Instituto ReCivitas', 'organizacao', 10000.0, 0.2),
            ('BRASIL', 'Brasil', 'nacao', 5000.0, 0.45),
            ('CHINA', 'China', 'nacao', 5000.0, 0.30),
            ('CIDADAO_01', 'Maria Silva', 'pessoa', 0.0, 0.70),
            ('CIDADAO_02', 'João Santos', 'pessoa', 0.0, 0.75),
            ('CIDADAO_03', 'Ana Costa', 'pessoa', 0.0, 0.68),
        ]

        for id_, nome, tipo, saldo, vuln in defaults:
            self.participantes[id_] = Participante(
                id=id_, nome=nome, tipo=tipo, saldo=saldo, vulnerabilidade=vuln
            )

        self._log("Sistema HERMES ζ⊕ + Φ-LIBER v3.0 inicializado", "info")
        self._log(f"α = {CONST.ALPHA:.6f} | γ = {CONST.GAMMA:.6f} | φ = {CONST.PHI:.6f}",
"info")

    def _log(self, msg: str, tipo: str = "info"):
        """Adiciona entrada ao log"""
        self.logs.append({
            'timestamp': datetime.now().strftime("%H:%M:%S"),
            'msg': msg,
            'tipo': tipo
        })
        if len(self.logs) > 100:
            self.logs.pop(0)
```

```
    #
═══════════════════════════════════════════════════════════════════════
    # GESTÃO DE PARTICIPANTES
    #
═══════════════════════════════════════════════════════════════════════

    def registrar_participante(self, id_: str, nome: str, tipo: str,
                               saldo: float = 0.0, vuln: float = 0.5) -> bool:
        """Registra novo participante no sistema"""
        if id_ in self.participantes:
            self._log(f"Erro: Participante {id_} já existe", "error")
            return False

        self.participantes[id_] = Participante(
            id=id_, nome=nome, tipo=tipo, saldo=saldo, vulnerabilidade=vuln
        )
        self._log(f"Participante registrado: {nome} ({tipo})", "info")
        return True

    def listar_participantes(self) -> List[Dict]:
        """Lista todos os participantes"""
        return [
            {
                'id': p.id,
                'nome': p.nome,
                'tipo': p.tipo,
                'saldo': p.saldo,
                'phi': p.phi,
                'vulnerabilidade': p.vulnerabilidade,
                'epsilon': p.epsilon
            }
            for p in self.participantes.values()
        ]

    #
═══════════════════════════════════════════════════════════════════════
    # EMISSÃO DE TOKENS (P = NP*)
    #
═══════════════════════════════════════════════════════════════════════

    def emitir_token(self, emissor: str, receptor: str, valor: float,
                     proposito: str = "Transferência",
                     eps_base: float = 0.8) -> Optional[TokenHERMES]:
        """
        Emite token HERMES com auto-validação P = NP*

        Args:
```

```
    emissor: ID do emissor
    receptor: ID do receptor
    valor: Valor em HERMES (H)
    proposito: Propósito da emissão
    eps_base: Fator ε base do propósito

Returns:
    TokenHERMES se válido, None se inválido
"""
# Verificar participantes
if emissor not in self.participantes:
    self._log(f"Erro: Emissor {emissor} não encontrado", "error")
    return None

if receptor not in self.participantes:
    self._log(f"Erro: Receptor {receptor} não encontrado", "error")
    return None

if valor <= 0:
    self._log("Erro: Valor deve ser positivo", "error")
    return None

p_emissor = self.participantes[emissor]
p_receptor = self.participantes[receptor]

# Calcular ε (benefício mútuo)
beneficio_emissor = valor * CONST.ALPHA * (1 - p_emissor.vulnerabilidade)
beneficio_receptor = valor * (1 - p_receptor.vulnerabilidade)
epsilon = OperadoresHERMES.epsilon_beneficio(beneficio_emissor, beneficio_receptor)
epsilon *= eps_base

# Calcular ζ⊕ (consenso simulado)
votos = [np.random.choice([1, -0.5], p=[0.85, 0.15]) for _ in range(7)]
zeta = OperadoresHERMES.zeta_convergencia(votos)

# Calcular Φ (energia criativa)
phi = PhiLiber.calc_phi(epsilon)

# Auto-validação P = NP*
auto_validado, razao = PNPStar.auto_validar(epsilon, zeta)

# Criar token
token = TokenHERMES(
    id=PNPStar.gerar_hash_delta(),
    emissor=emissor,
    receptor=receptor,
    valor=valor,
    proposito=proposito,
    epsilon=epsilon,
    zeta=zeta,
    phi=phi,
    auto_validado=auto_validado,
```

```
            hash_delta=PNPStar.gerar_hash_delta()
        )

        if auto_validado:
            # Atualizar saldos
            p_receptor.saldo += valor

            # Atualizar Φ dos participantes
            p_receptor.phi = min(20, p_receptor.phi + phi * 0.1)
            p_emissor.phi = min(20, p_emissor.phi + phi * 0.05)

            # Reduzir vulnerabilidade do receptor (RBU effect)
            p_receptor.vulnerabilidade = max(0.05, p_receptor.vulnerabilidade - epsilon * 0.05)

            self.tokens.append(token)
            self.historico.append({'epsilon': epsilon, 'phi': phi, 'zeta': zeta})

            self._log(f"√ P=NP* Token AUTO-VALIDADO: {valor} H | {razao}", "pnp")
        else:
            self._log(f"✗ Falha auto-validação: {razao}", "error")

        return token if auto_validado else None

    #
═══════════════════════════════════════════════════════════════════════════
    # CENÁRIOS PRÉ-DEFINIDOS
    #
═══════════════════════════════════════════════════════════════════════════

    def cenario_rbu(self, valor_por_pessoa: float = 100.0) -> List[TokenHERMES]:
        """
        Cenário: Emissão de RBU para todos os cidadãos

        RBU é INVESTIMENTO, não custo (demonstrado por e^(ε²))
        """
        self._log("🎯 Executando cenário RBU...", "pnp")
        tokens_emitidos = []

        for p in self.participantes.values():
            if p.tipo == 'pessoa':
                token = self.emitir_token(
                    emissor='RECIVITAS',
                    receptor=p.id,
                    valor=valor_por_pessoa,
                    proposito='RBU',
                    eps_base=0.95
                )
                if token:
                    tokens_emitidos.append(token)
```

```
        self._log(f"RBU completa: {len(tokens_emitidos)} tokens emitidos", "info")
        return tokens_emitidos

    def cenario_comercio_bilateral(self, origem: str = 'BRASIL',
                                   destino: str = 'CHINA',
                                   valor: float = 1000.0) -> Optional[TokenHERMES]:
        """
        Cenário: Comércio bilateral sem intermediário (dólar/ouro)
        """
        self._log(f"🌍 Executando comércio {origem} → {destino}...", "pnp")

        return self.emitir_token(
            emissor=origem,
            receptor=destino,
            valor=valor,
            proposito='Comércio Bilateral',
            eps_base=0.80
        )

    def cenario_ciclo_completo(self) -> Dict[str, any]:
        """
        Cenário: Ciclo completo (RBU + Comércio + Investimento)
        """
        self._log("🔄 Executando ciclo completo...", "pnp")

        resultados = {
            'rbu': self.cenario_rbu(100.0),
            'comercio': self.cenario_comercio_bilateral('BRASIL', 'CHINA', 1000.0),
            'investimento': self.emitir_token('CHINA', 'RECIVITAS', 500.0, 'Investimento Social', 0.70)
        }

        return resultados

    #
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════
    # SIMULADOR Φ-LIBER (RBU)
    #
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════

    def simular_rbu_phi(self, vuln_inicial: float = 0.70,
                        cobertura: float = 1.0,
                        valor_rbu: float = 0.50) -> Dict[str, float]:
        """
        Simula impacto de RBU usando teoria Φ-LIBER

        Args:
            vuln_inicial: Vulnerabilidade inicial [0, 1]
            cobertura: Cobertura da RBU [0, 1]
            valor_rbu: Valor como % do mínimo vital [0, 1]
```

```
        Returns:
            Métricas antes/depois incluindo amplificação
        """
        # ANTES da RBU
        eps_antes = PhiLiber.calc_epsilon(vuln_inicial)
        phi_antes = PhiLiber.calc_phi(eps_antes)
        exp_antes = np.exp(eps_antes ** 2)

        # DEPOIS da RBU
        vuln_depois = PhiLiber.aplicar_rbu(vuln_inicial, cobertura, valor_rbu)
        eps_depois = PhiLiber.calc_epsilon(vuln_depois)
        phi_depois = PhiLiber.calc_phi(eps_depois)
        exp_depois = np.exp(eps_depois ** 2)

        # Amplificação
        resultado = PhiLiber.amplificacao_nao_linear(eps_antes, eps_depois)
        resultado['vuln_inicial'] = vuln_inicial
        resultado['vuln_depois'] = vuln_depois
        resultado['cobertura'] = cobertura
        resultado['valor_rbu'] = valor_rbu

        return resultado

    #
═══════════════════════════════════════════════════════════════════════════
═══════════════════════
    # MÉTRICAS DO SISTEMA
    #
═══════════════════════════════════════════════════════════════════════════
═══════════════════════

    def metricas_globais(self) -> Dict[str, float]:
        """Retorna métricas globais do sistema"""
        if not self.tokens:
            return {
                'participantes': len(self.participantes),
                'tokens': 0,
                'tokens_validos': 0,
                'total_emitido': 0.0,
                'phi_medio': 0.0,
                'epsilon_medio': 0.0,
                'zeta_medio': 0.0,
                'amplificacao': 1.0,
                'taxa_validacao': 0.0
            }

        tokens_validos = [t for t in self.tokens if t.auto_validado]
        total_emitido = sum(t.valor for t in tokens_validos)

        eps_medio = np.mean([t.epsilon for t in tokens_validos]) if tokens_validos else 0
        phi_medio = np.mean([t.phi for t in tokens_validos]) if tokens_validos else 0
        zeta_medio = np.mean([t.zeta for t in tokens_validos]) if tokens_validos else 0
```

```
    amplificacao = np.exp(eps_medio ** 2) if eps_medio > 0 else 1.0

    return {
        'participantes': len(self.participantes),
        'tokens': len(self.tokens),
        'tokens_validos': len(tokens_validos),
        'total_emitido': total_emitido,
        'phi_medio': phi_medio,
        'epsilon_medio': eps_medio,
        'zeta_medio': zeta_medio,
        'amplificacao': amplificacao,
        'taxa_validacao': len(tokens_validos) / len(self.tokens) * 100 if self.tokens else 0
    }

def comparacao_paradigmas(self) -> List[Dict]:
    """Compara paradigmas: Dólar vs Ouro/BRICS vs HERMES"""
    return [
        {
            'aspecto': 'Lastro',
            'dolar': 'Tesouro Americano',
            'ouro_brics': 'Metal físico Tier 1',
            'hermes': 'Confiança ζ⊕ + Φ regenerativo'
        },
        {
            'aspecto': 'Bootstrap',
            'dolar': 'Dependência histórica',
            'ouro_brics': 'Quem tem ouro começa',
            'hermes': 'P=NP* Auto-validação'
        },
        {
            'aspecto': 'Verificação',
            'dolar': 'Confiança externa (EUA)',
            'ouro_brics': 'Posse física',
            'hermes': 'ε > 0 ∧ ζ⊕ > 0.7 verificável'
        },
        {
            'aspecto': 'Escala',
            'dolar': 'Linear',
            'ouro_brics': 'Linear',
            'hermes': 'Não-linear e^(ε²)'
        },
        {
            'aspecto': 'Risco',
            'dolar': 'Sanções, congelamento',
            'ouro_brics': 'Novo colonialismo dourado',
            'hermes': 'Adesão voluntária'
        },
        {
            'aspecto': 'Para Brasil',
            'dolar': 'Dependência/vulnerabilidade',
            'ouro_brics': 'Subordinação a quem tem ouro',
```

```
            'hermes': 'Autonomia + RBU viável'
        },
        {
            'aspecto': 'Natureza',
            'dolar': 'Commodity (papel)',
            'ouro_brics': 'Commodity (metal)',
            'hermes': 'RELAÇÃO (compositor)'
        }
    ]

    #
═══════════════════════════════════════════════════════════════════════════
    # EXPORTAÇÃO
    #
═══════════════════════════════════════════════════════════════════════════

    def exportar_estado(self) -> Dict:
        """Exporta estado completo do sistema"""
        return {
            'constantes': {
                'PHI': CONST.PHI,
                'ALPHA': CONST.ALPHA,
                'GAMMA': CONST.GAMMA,
                'TAU_0': CONST.TAU_0,
                'LIMIAR_ZETA': CONST.LIMIAR_ZETA,
                'LIMIAR_EPSILON': CONST.LIMIAR_EPSILON
            },
            'participantes': self.listar_participantes(),
            'tokens': [t.to_dict() for t in self.tokens],
            'metricas': self.metricas_globais(),
            'historico': self.historico,
            'logs': self.logs[-20:]  # Últimos 20 logs
        }

    def exportar_json(self, filepath: str):
        """Exporta estado para arquivo JSON"""
        with open(filepath, 'w', encoding='utf-8') as f:
            json.dump(self.exportar_estado(), f, indent=2, ensure_ascii=False)
        self._log(f"Estado exportado para {filepath}", "info")


#
═══════════════════════════════════════════════════════════════════════════
# DEMONSTRAÇÃO DA NÃO-LINEARIDADE
#
═══════════════════════════════════════════════════════════════════════════

def demonstrar_nao_linearidade():
```

```
    """Demonstra a não-linearidade e^(ε²) da teoria Φ-LIBER"""
    print("\n" + "═" * 70)
    print("DEMONSTRAÇÃO: NÃO-LINEARIDADE e^(ε²)")
    print("═" * 70)
    print("\n\"Por que pequenos ganhos de liberdade geram grandes ganhos de energia\"\n")

    valores_eps = [0.2, 0.5, 0.8, 1.0, 1.5, 2.0]

    print(f"{'ε':<8} {'e^(ε²)':<12} {'Φ':<12} {'Δε':<12} {'Amplificação'}")
    print("-" * 60)

    eps_anterior = valores_eps[0]
    exp_anterior = np.exp(eps_anterior ** 2)

    for eps in valores_eps:
        exp_term = np.exp(eps ** 2)
        phi = PhiLiber.calc_phi(eps)

        if eps == valores_eps[0]:
            print(f"{eps:<8.2f} {exp_term:<12.2f} {phi:<12.2f} {'—':<12} {'—'}")
        else:
            delta_eps = (eps - eps_anterior) / eps_anterior * 100
            delta_exp = (exp_term - exp_anterior) / exp_anterior * 100
            amp = delta_exp / delta_eps if delta_eps > 0 else 1
            print(f"{eps:<8.2f} {exp_term:<12.2f} {phi:<12.2f} {f'+{delta_eps:.0f}%':<12}
{amp:.2f}x")

        eps_anterior = eps
        exp_anterior = exp_term

    print("\n" + "-" * 60)
    print("RESULTADO: De ε=0.2 para ε=1.5:")
    print(f"  • Liberdade aumenta: {(1.5/0.2 - 1)*100:.0f}%")
    print(f"  • e^(ε²) aumenta: {(np.exp(1.5**2)/np.exp(0.2**2) - 1)*100:.0f}%")
    print("\n  → 21% mais liberdade → 813% mais energia criativa")
    print("  → RBU é INVESTIMENTO, não custo!")


#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════════
# EXECUÇÃO PRINCIPAL
#
═══════════════════════════════════════════════════════════════════════════
═══════════════════════════════

def main():
    """Execução principal do simulador"""

    print("═" * 70)
    print("SIMULADOR UNIFICADO HERMES ζ⊕ + Φ-LIBER v3.0")
    print("═" * 70)
```

```
    print()
    print("\"Nem por ouro. Nem muito menos por um punhado de dólar,")
    print(" nem a menos, nem a mais.\"")
    print("                          — Marcus Brancaglione")
    print()
    print("=" * 70)

    # Criar simulador
    sim = SimuladorHERMESLiber()

    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════
    # 1. CONSTANTES FUNDAMENTAIS
    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════

    print("\n[1] CONSTANTES FUNDAMENTAIS")
    print("-" * 50)
    print(f"    φ (razão áurea)     = {CONST.PHI:.10f}")
    print(f"    α (constante LIBER) = {CONST.ALPHA:.10f}")
    print(f"    γ (Euler-Mascheroni)= {CONST.GAMMA:.10f}")
    print(f"    τ₀ (período base)   = {CONST.TAU_0:.10f}")

    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════
    # 2. DEMONSTRAÇÃO NÃO-LINEARIDADE
    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════

    demonstrar_nao_linearidade()

    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════
    # 3. SIMULAÇÃO RBU (Φ-LIBER)
    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════

    print("\n" + "=" * 70)
    print("SIMULAÇÃO RBU: BRASIL (Φ-LIBER)")
    print("=" * 70)

    resultado_rbu = sim.simular_rbu_phi(
        vuln_inicial=0.70,
        cobertura=1.0,
        valor_rbu=0.50
    )
```

```
    print(f"\nParâmetros:")
    print(f"  • Vulnerabilidade inicial: {resultado_rbu['vuln_inicial']*100:.0f}%")
    print(f"  • Cobertura RBU: {resultado_rbu['cobertura']*100:.0f}%")
    print(f"  • Valor RBU: {resultado_rbu['valor_rbu']*100:.0f}% do mínimo vital")

    print(f"\nResultados:")
    print(f"  • Vulnerabilidade: {resultado_rbu['vuln_inicial']*100:.0f}% →
{resultado_rbu['vuln_depois']*100:.0f}%")
    print(f"  • ε (liberdade): {resultado_rbu['eps_antes']:.4f} → {resultado_rbu['eps_depois']:.4f}
({resultado_rbu['delta_eps_pct']:+.1f}%)")
    print(f"  • e^(ε²): {resultado_rbu['exp_antes']:.4f} → {resultado_rbu['exp_depois']:.4f}")
    print(f"  • Φ (energia): {resultado_rbu['phi_antes']:.2f} → {resultado_rbu['phi_depois']:.2f}
({resultado_rbu['delta_phi_pct']:+.1f}%)")
    print(f"  • Amplificação: {resultado_rbu['amplificacao']:.2f}x")

    print(f"\n  → ROI Social: Cada R$1 em RBU gera ~R${resultado_rbu['amplificacao']:.2f} em
energia criativa")

    #
═══════════════════════════════════════════════════════════════════════════
    # 4. CENÁRIO RBU (HERMES)
    #
═══════════════════════════════════════════════════════════════════════════

    print("\n" + "═" * 70)
    print("CENÁRIO RBU: EMISSÃO DE TOKENS (HERMES)")
    print("═" * 70)

    tokens_rbu = sim.cenario_rbu(100.0)

    print(f"\nTokens emitidos: {len(tokens_rbu)}")
    for t in tokens_rbu:
        print(f"  • {t.receptor}: {t.valor} H | ε={t.epsilon:.4f} | ζ⊕={t.zeta:.4f} | Φ={t.phi:.2f}")

    #
═══════════════════════════════════════════════════════════════════════════
    # 5. CENÁRIO COMÉRCIO BILATERAL
    #
═══════════════════════════════════════════════════════════════════════════

    print("\n" + "═" * 70)
    print("CENÁRIO COMÉRCIO: BRASIL → CHINA (HERMES)")
    print("═" * 70)

    token_comercio = sim.cenario_comercio_bilateral('BRASIL', 'CHINA', 1000.0)

    if token_comercio:
```

```
    print(f"\nToken emitido:")
    print(f"  • ID: {token_comercio.id}")
    print(f"  • Valor: {token_comercio.valor} H")
    print(f"  • ε (benefício mútuo): {token_comercio.epsilon:.4f}")
    print(f"  • ζ⊕ (consenso): {token_comercio.zeta:.4f}")
    print(f"  • Φ (energia): {token_comercio.phi:.2f}")
    print(f"  • Hash δ: {token_comercio.hash_delta}")
    print(f"  • Auto-validado: {'√ SIM (P=NP*)' if token_comercio.auto_validado else '✗ NÃO'}")

# ═══════════════════════════════════════════════════════════════════════
# 6. MÉTRICAS GLOBAIS
# ═══════════════════════════════════════════════════════════════════════

print("\n" + "═" * 70)
print("MÉTRICAS GLOBAIS DO SISTEMA")
print("═" * 70)

metricas = sim.metricas_globais()

print(f"\n  • Participantes: {metricas['participantes']}")
print(f"  • Tokens emitidos: {metricas['tokens']}")
print(f"  • Tokens válidos: {metricas['tokens_validos']}")
print(f"  • Taxa de validação: {metricas['taxa_validacao']:.1f}%")
print(f"  • Total emitido: {metricas['total_emitido']:.2f} H")
print(f"  • ε médio: {metricas['epsilon_medio']:.4f}")
print(f"  • ζ⊕ médio: {metricas['zeta_medio']:.4f}")
print(f"  • Φ médio: {metricas['phi_medio']:.2f}")
print(f"  • Amplificação e^(ε²): {metricas['amplificacao']:.2f}x")

# ═══════════════════════════════════════════════════════════════════════
# 7. COMPARAÇÃO DE PARADIGMAS
# ═══════════════════════════════════════════════════════════════════════

print("\n" + "═" * 70)
print("COMPARAÇÃO DE PARADIGMAS MONETÁRIOS")
print("═" * 70)

comparacao = sim.comparacao_paradigmas()

print(f"\n{'Aspecto':<15} {'Dólar':<25} {'Ouro/BRICS':<25} {'HERMES'}")
print("-" * 90)

for c in comparacao:
    print(f"{c['aspecto']:<15} {c['dolar']:<25} {c['ouro_brics']:<25} {c['hermes']}")
```

```
    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════
    # 8. EXPORTAR ESTADO
    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════

    print("\n" + "═" * 70)
    print("EXPORTAÇÃO")
    print("═" * 70)

    sim.exportar_json("/mnt/user-data/outputs/estado_hermes_liber.json")
    print("\n  √ Estado exportado para estado_hermes_liber.json")

    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════
    # CONCLUSÃO
    #
═══════════════════════════════════════════════════════════════════════
═══════════════════════════

    print("\n" + "═" * 70)
    print("CONCLUSÃO")
    print("═" * 70)
    print()
    print("  P = NP* | A verificação É a criação")
    print("  \"A demonstração É a prova.\"")
    print()
    print("  NEM AUTOR, NEM LEITOR. COMPOSITOR.")
    print()
    print("═" * 70)
    print("Instituto ReCivitas • CC BY-SA 4.0 + ⒶRobinRight v3.0 ζ⊕")
    print("═" * 70)

    return sim


if __name__ == "__main__":
    simulador = main()
```