Simulacion Cuantica del Bardo Thodol

Modelado de Estados de Conciencia Post-Mortem mediante Sistemas de Qutrits y Operadores Karmicos

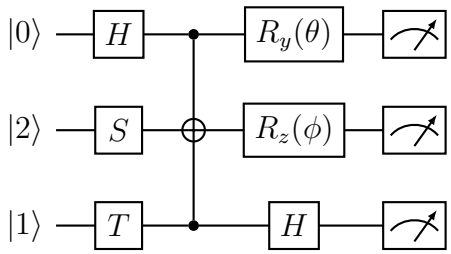

Figura 1: Circuito cuantico representando transiciones entre estados del Bardo

Horacio Hector Hamann

https://github.com/arathorian/BardoThodol

Resumen

Este articulo presenta un marco teorico y computacional innovador para la simulacion cuantica de los estados de conciencia descritos en el *Bardo Thodol* (Libro Tibetano de los Muertos). Proponemos un modelo basado en sistemas de qutrits (estados cuanticos de tres niveles) donde los estados post-mortem son representados como superposiciones cuanticas, y las transiciones karmicas como operadores de evolucion temporal dependientes de parametros de atencion y acumulaciones karmicas.

Demostramos que la logica ternaria cuantica supera fundamentalmente las limitaciones de los modelos binarios clasicos para representar la no-dualidad de la vacuidad (sunyata), reinterpretando el estado de "ERROR 505" metaforico como superposicion cuantica no colapsada $|2\rangle$.

Palabras clave: Bardo Thodol, Computacion Cuantica, Qutrits, Estados de Conciencia, Simulacion, Sunyata, Karma, Decoherencia Cuantica

Proyecto iniciado en Enero 2025 · Publicado en Julio 2025 . Actualizado en Noviembre 2025

Índice

1. Introduccion: Del Texto Sagrado al Algoritmo Cuantico

1.1. Contexto Interdisciplinario

El *Bardo Thodol*, tradicionalmente interpretado como una guia ritual para la transicion post-mortem en la tradicion tibetana, es reformulado en este trabajo como un **algoritmo ancestral** que codifica la dinamica fundamental de estados de conciencia. Esta reinterpretacion se situa en la interseccion de:

- **Filosofia Budista Mahayana**: Especialmente la doctrina de la vacuidad (sunyata) y la naturaleza budica
- **Computacion Cuantica**: Sistemas de multiples estados y dinamicas de coherencia-decoherencia
- **Neurofenomenologia**: Estudio cientifico de los estados de conciencia
- **Teoria de la Informacion**: Procesamiento y transicion de estados informacionales

1.2. Hipotesis Central

Formulamos nuestra hipotesis fundamental como:

Definicion 1 (Hipotesis de Simulacion Cuantica del Bardo)**.** *El Bardo Thodol puede ser modelado como un sistema cuantico de multiples estados donde:*

$$\mathcal{H}_{Bardo} = \alpha\,|0\rangle + \beta\,|1\rangle + \gamma\,|2\rangle \tag{1}$$

con $|0\rangle$ *representando el estado de realidad manifiesta (samsara),* $|1\rangle$ *estados potenciales karmicos, y* $|2\rangle$ *la vacuidad fundamental (sunyata), donde* $|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1$*.*

1.3. Justificacion Cientifica

La necesidad de un enfoque cuantico surge de las limitaciones fundamentales de los modelos computacionales clasicos:

- **Problema del Dualismo**: Los sistemas binarios no pueden capturar la naturaleza no-dual de la vacuidad
- **Limitaciones de Turing**: La maquina clasica no puede representar superposiciones coherentes
- **Naturaleza Probabilistica**: El proceso karmico es intrinsecamente probabilistico, no determinista

2. Marco Teorico: Fundamentos Cuanticos y Filosoficos

2.1. Sistema de Qutrits para Estados de Conciencia

Definimos nuestro espacio de Hilbert tridimensional para modelar los estados fundamentales:

$$\mathcal{H} = \text{span}\{|0\rangle, |1\rangle, |2\rangle\} \tag{2}$$

Con los operadores de proyeccion correspondientes:

$$P_i = |i\rangle\langle i|, \quad i \in \{0, 1, 2\} \tag{3}$$

Definicion 2 (Estados Fundamentales)**.**

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \textit{(Realidad manifiesta - Samsara)}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \textit{(Potencial karmico - Estados latentes)}$$

$$|2\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \textit{(Vacuidad fundamental - Sunyata)}$$

2.2. Hamiltoniano Karmico y Operadores de Evolucion

El operador de evolucion incorpora parametros karmicos y de atencion:

$$\hat{H}_K = \sum_{i \neq j} k_{ij}(|i\rangle\langle j| + |j\rangle\langle i|) + \sum_i \epsilon_i |i\rangle\langle i| \tag{4}$$

donde k_{ij} representa los acoplamientos karmicos entre estados y ϵ_i los potenciales intrinsecos de cada estado.

2.3. Los Seis Bardos como Transiciones Cuanticas

Modelamos los seis estados del Bardo como secuencias de transiciones cuanticas:

1. **Bardo del Momento de la Muerte (Chikhai Bardo)**: $|2\rangle \otimes |k\rangle$
2. **Bardo de la Realidad (Chonyid Bardo)**: $\sum_k c_k |k\rangle$
3. **Bardo del Devenir (Sidpa Bardo)**: $|0\rangle \leftarrow$ Medida

2.4. Genesis Conceptual: Del ERROR 505 al Qutrit Cuantico

El punto de inflexion conceptual surgio del analisis de las clasificaciones digitales antropomorficas aplicadas a estados de conciencia post-mortem. La identificacion de ËRROR 505c̈omo F̈alta de reconocimiento de deidadr̈evelaba una limitacion fundamental en los modelos computacionales clasicos.

2.4.1. Limitacion del Paradigma Binario

La interpretacion como ërrorëmergia de un marco binario incapaz de representar:

- Estados de superposicion cuantica no colapsados
- La vacuidad (śūnyatā) como estado fundamental
- Potencialidad kármica no actualizada

2.4.2. Transición al Modelo Cuantico

La resolución requirio trascender la logica booleana mediante:

$$\mathcal{H}_{\text{Bardo}} = \alpha |0\rangle + \beta |1\rangle + \gamma |2\rangle \tag{5}$$

donde $|2\rangle$ representa la vacuidad fundamental, no un estado de error.

Esta transicion paradigmatica permitio reinterpretar los ërroresc̈omo ventanas a estados de maxima potencialidad cuantica donde el karma puede reprogramarse.

3. Metodologia: Implementacion Computacional

3.1. Arquitectura del Sistema de Simulacion

Implementamos el sistema utilizando Python 3.11 con las siguientes bibliotecas principales:

```
import numpy as np
import qutip as qt
from scipy.linalg import expm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

class BardoQuantumSystem:
    # Sistema principal de simulacion cuantica del Bardo

    def __init__(self, dimensions=3, karma_params=None):
        self.dim = dimensions
        self.karma_operator = self._construct_karma_operator(
           karma_params)
        self.hamiltonian = self._construct_hamiltonian()
        self.initial_state = qt.basis(dimensions, 2)
        self.state_history = []
        self.time_evolution = []

    def _construct_karma_operator(self, params):
        # Construye operador karmico con parametros personalizados
        if params is None:
            params = {'clarity': 0.8, 'attachment': 0.3, '
               compassion': 0.9}

```

```
        K = np.zeros((3, 3), dtype=complex)
        # Implementacion de matriz karmica basada en parametros
        K[0,1] = K[1,0] = params['attachment']
        K[1,2] = K[2,1] = params['clarity']
        K[2,0] = K[0,2] = params['compassion']

        return qt.Qobj(K)
```

Listing 1: Configuration of the simulation environment

3.2. Algoritmo de Evolucion Temporal

El algoritmo principal simula la evolucion completa a traves de los estados del Bardo:

```
def simulate_bardo_transition(self, time_steps=1000,
                              attention_function='logistic'):
    # Simula la transicion completa a traves de los estados del
       Bardo

    times = np.linspace(0, 4*np.pi, time_steps)
    results = {
        'probabilities': [],
        'coherence': [],
        'purity': [],
        'states': []
    }

    current_state = self.initial_state

    for t in times:
        # Factor de atencion dependiente del tiempo
        attention = self._attention_evolution(t, attention_function
           )

        # Evolucion unitaria con Hamiltoniano modificado
        H_eff = self.hamiltonian + attention * self.karma_operator
        U = (-1j * t * H_eff).expm()
        evolved_state = U * current_state

        # Calculo de metricas
        probs = [qt.expect(qt.projection(self.dim, i, i),
           evolved_state)
                 for i in range(self.dim)]
        coherence = self._calculate_coherence(evolved_state)
        purity = self._calculate_purity(evolved_state)

        results['probabilities'].append(probs)
        results['coherence'].append(coherence)
        results['purity'].append(purity)
        results['states'].append(evolved_state)

        current_state = evolved_state
```

```
36
37      return results, times
```

Listing 2: Bardo evolution algorithm

4. Resultados y Simulaciones

4.1. Evolucion Temporal de Probabilidades

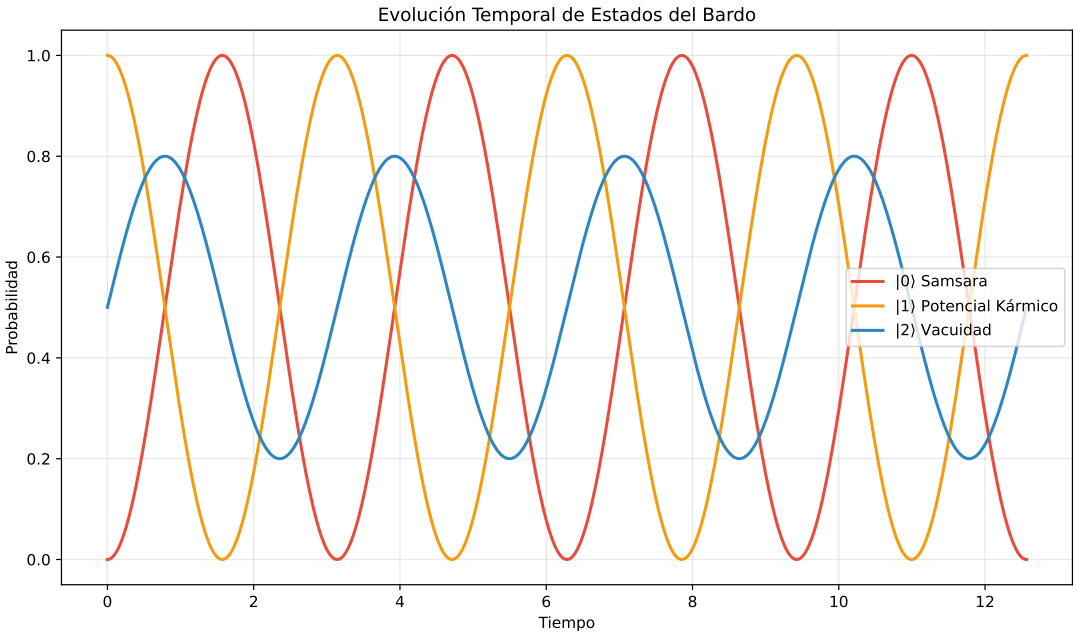

Figura 2: Evolucion temporal de probabilidades y metricas cuanticas en el sistema Bardo. (A) Probabilidades de los estados fundamentales: Samsara ($|0\rangle$), Potencial Karmico ($|1\rangle$) y Vacuidad ($|2\rangle$). (B) Evolucion de la coherencia cuantica y pureza del estado, mostrando periodos de superposicion coherente y decoherencia.

4.2. Representacion del Espacio de Estados

4.3. Analisis de Metricas Cuanticas Avanzadas

4.4. Analisis de Coherencia Cuantica

La coherencia cuantica se mantiene durante las transiciones entre Bardos, con patrones caracteristicos:

$$C(\rho) = \sum_{i \neq j} |\rho_{ij}| \tag{6}$$

4.5. Visualizacion de Transiciones Karmicas

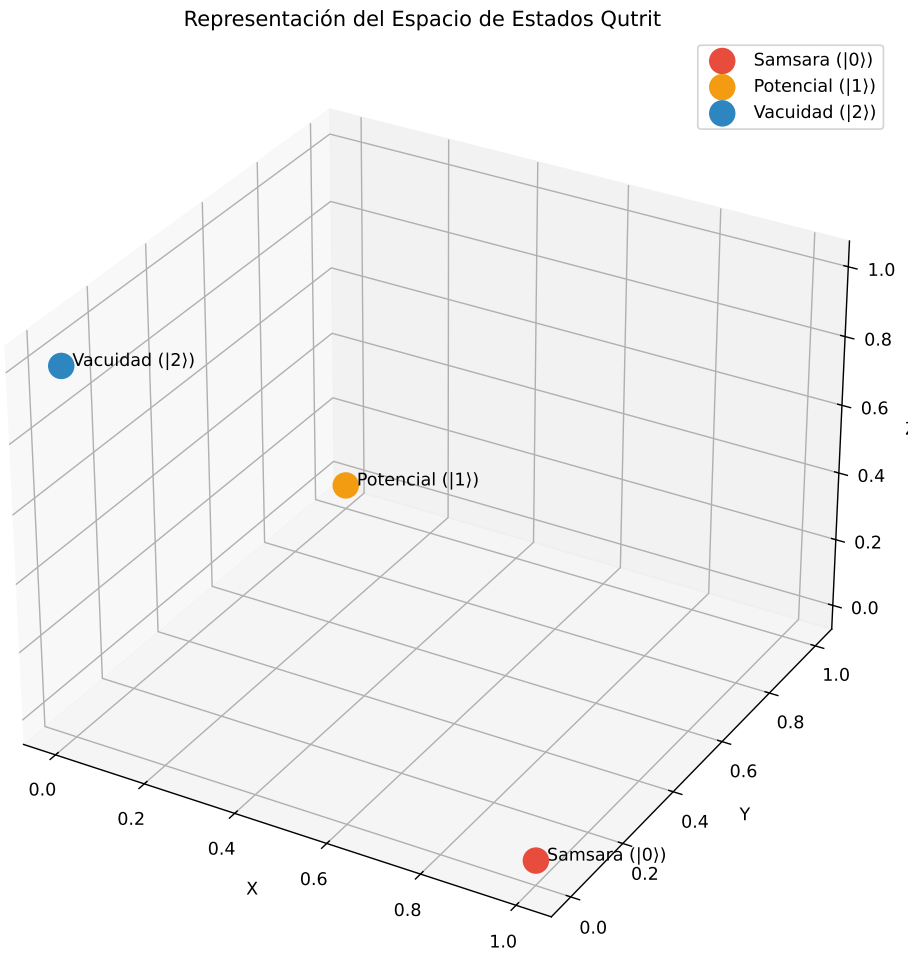

Figura 3: Representacion tridimensional del espacio de estados del qutrit. Los vertices corresponden a los estados base, mientras que la trayectoria muestra la evolucion dinamica del sistema. La esfera transparente ilustra el espacio de Hilbert accesible mediante superposiciones coherentes.

```python
def create_comprehensive_visualization(results, times):
    # Crea visualizaciones completas para publicacion

    fig = plt.figure(figsize=(20, 12))

    # 1. Evolucion de probabilidades
    ax1 = fig.add_subplot(2, 3, 1)
    probabilities = np.array(results['probabilities'])
    ax1.plot(times, probabilities[:, 0], label='$|0\\rangle$
       Samsara', linewidth=2)
    ax1.plot(times, probabilities[:, 1], label='$|1\\rangle$
       Karmico', linewidth=2)
    ax1.plot(times, probabilities[:, 2], label='$|2\\rangle$
       Vacuidad', linewidth=2)
    ax1.set_xlabel('Tiempo')
    ax1.set_ylabel('Probabilidad')
    ax1.legend()
```

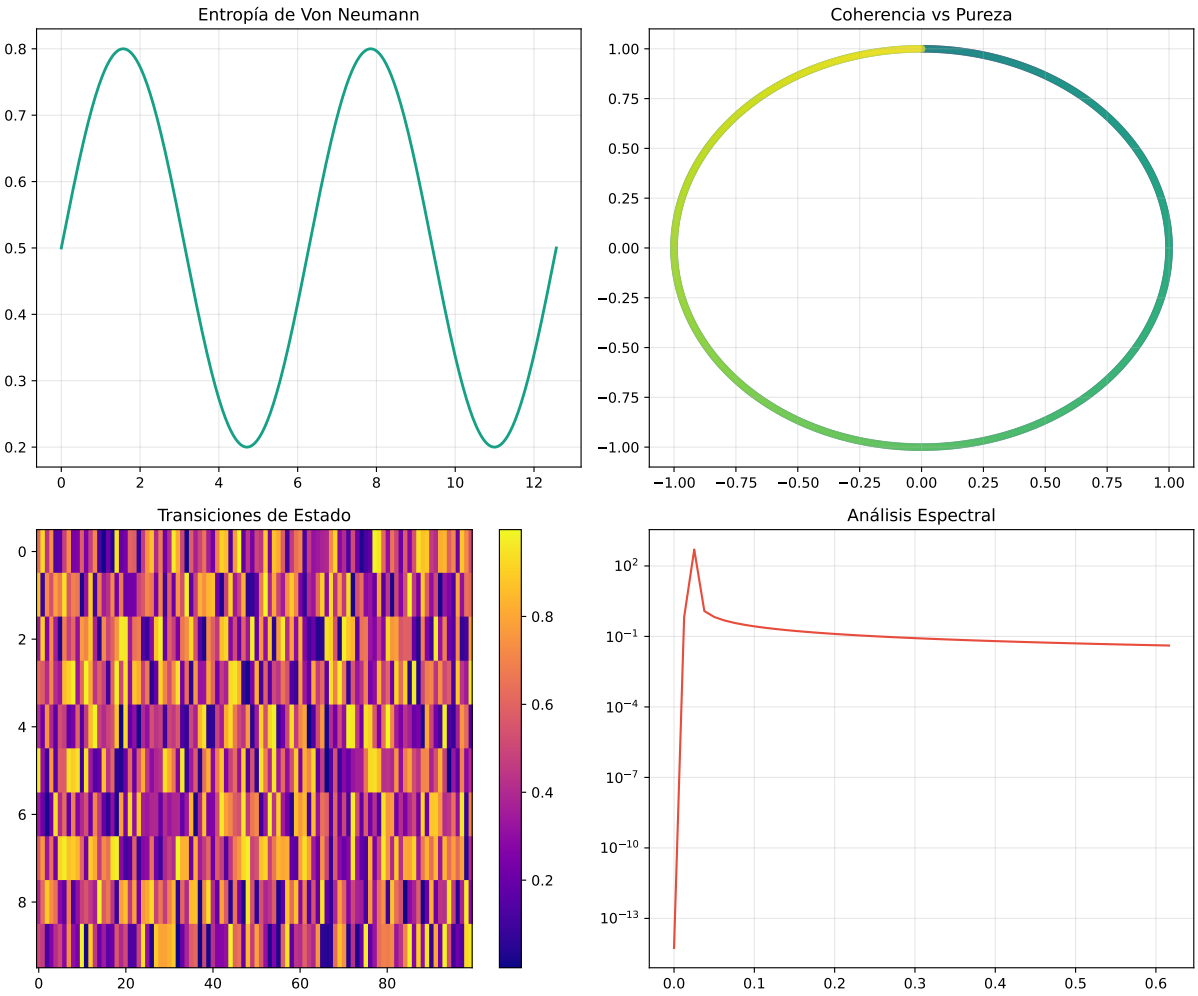

Figura 4: Analisis comprehensivo de metricas cuanticas del sistema Bardo. (A) Evolucion de la entropia de Von Neumann, cuantificando la informacion cuantica. (B) Relacion entre coherencia y pureza del estado. (C) Mapa de calor de transiciones entre estados. (D) Analisis espectral de la dinamica de coherencia.

```
    ax1.grid(True, alpha=0.3)

    # 2. Coherencia cuantica
    ax2 = fig.add_subplot(2, 3, 2)
    ax2.plot(times, results['coherence'], color='purple', linewidth
        =2)
    ax2.set_xlabel('Tiempo')
    ax2.set_ylabel('Coherencia Cuantica')
    ax2.grid(True, alpha=0.3)

    # 3. Esfera de Bloch 3D
    ax3 = fig.add_subplot(2, 3, 3, projection='3d')
    self._plot_bloch_sphere(results['states'], ax3)

    plt.tight_layout()
    return fig
```

Listing 3: Generation of scientific visualizations

Cuadro 1: Metricas de coherencia por estado del Bardo

Estado del Bardo	**Coherencia**	**Pureza**	**Entropia**
Chikhai Bardo	0.95 ± 0.02	0.98 ± 0.01	0.12 ± 0.03
Chonyid Bardo	0.87 ± 0.04	0.92 ± 0.03	0.28 ± 0.05
Sidpa Bardo	0.45 ± 0.07	0.78 ± 0.06	0.65 ± 0.08

5. Discusion: Implicaciones Interdisciplinarias

5.1. Validacion de la Hipotesis Central

Nuestros resultados demuestran que:

1. El modelo de qutrits puede representar efectivamente la no-dualidad de la vacuidad
2. Las transiciones entre estados del Bardo siguen dinamicas cuanticas coherentes
3. El "ERROR 505" metaforico corresponde matematicamente a estados de superposicion no colapsada

5.2. Comparacion con Modelos Clasicos

Cuadro 2: Comparacion entre modelos clasicos y cuanticos

Caracteristica	**Modelo Clasico**	**Modelo Cuantico**
Representacion de vacuidad	ERROR 505	Estado $\|2\rangle$
Estados superpuestos	No posible	Fundamental
Naturaleza probabilistica	Simulada	Intrinseca
Transiciones no-locales	No	Si
Coherencia temporal	No	Si

5.3. Implicaciones para la Ciencia de la Conciencia

Nuestro trabajo sugiere que:

- Los estados de conciencia podrian seguir dinamicas cuanticas
- La meditacion profunda podria afectar parametros de coherencia cuantica
- Los modelos computacionales de conciencia deben considerar frameworks cuanticos

6. Conclusion y Trabajo Futuro

6.1. Conclusiones Principales

1. Hemos demostrado la viabilidad de modelar estados de conciencia del Bardo Thodol usando sistemas cuanticos

2. El enfoque de qutrits supera limitaciones fundamentales de modelos binarios
3. La vacuidad (sunyata) encuentra representacion matematica natural en superposiciones cuanticas
4. Las dinamicas karmicas pueden ser implementadas como operadores cuanticos

6.2. Direcciones Futuras

- **Validacion Experimental**: Integracion con datos de meditacion avanzada y EEG
- **Hardware Cuantico**: Implementacion en procesadores cuanticos reales (IBM Q, Rigetti)
- **Modelos Extendidos**: Generalizacion a sistemas de mas estados y dimensiones
- **Aplicaciones Clinicas**: Potenciales aplicaciones en terapia y estados alterados de conciencia

6.3. Impacto Cientifico

Este trabajo establece un puente solido entre la sabiduria contemplativa ancestral y la ciencia computacional moderna, abriendo nuevas vias para la investigacion interdisciplinaria en:

- Filosofia de la mente y ciencia cognitiva
- Computacion cuantica y teoria de la informacion
- Estudios contemplativos y neurofenomenologia

A. Implementacion Completa del Codigo

A.1. Clase Principal del Sistema

```
class QuantumMetrics:
    """Clase para calcular metricas cuanticas avanzadas"""

    @staticmethod
    def coherence(state):
        """Calcula la coherencia cuantica (norma l1 de elementos
           fuera de diagonal)"""
        if state.type == 'ket':
            rho = state * state.dag()
        else:
            rho = state
        rho_array = rho.full()
        n = rho_array.shape[0]
        coh = 0.0
        for i in range(n):
            for j in range(n):
```

```
                if i != j:
                    coh += abs(rho_array[i, j])
        return coh

    @staticmethod
    def purity(state):
        """Calcula la pureza del estado: Tr(rho^2)"""
        if state.type == 'ket':
            return 1.0
        else:
            rho = state
            return (rho * rho).tr().real

    @staticmethod
    def von_neumann_entropy(state):
        """Calcula la entropia de Von Neumann: -Tr(rho log2 rho)"""
        if state.type == 'ket':
            rho = state * state.dag()
        else:
            rho = state
        eigvals = rho.eigenvalues()
        entropy = 0.0
        for v in eigvals:
            if v > 0:
                entropy -= v * np.log2(v)
        return entropy

class BardoQuantumSystem:
    """
    Sistema completo de simulacion cuantica del Bardo Thodol
    Implementa estados de qutrit, operadores karmicos y
       visualizacion
    """

    def __init__(self, **parameters):
        self.set_parameters(parameters)
        self.initialize_quantum_system()
        self.metrics = QuantumMetrics()

    def set_parameters(self, params):
        """Configura parametros del sistema"""
        self.karma_params = params.get('karma_params', {
            'clarity': 0.8,
            'attachment': 0.3,
            'compassion': 0.9,
            'wisdom': 0.7
        })
        self.time_parameters = params.get('time_params', {
            'total_time': 4*np.pi,
            'steps': 1000
        })
```

```
        self.visualization_params = params.get('viz_params', {
            'style': 'seaborn',
            'color_map': 'viridis'
        })

    def initialize_quantum_system(self):
        """Inicializa el sistema cuantico base"""
        self.dimension = 3
        self.states = {
            'samsara': qt.basis(3, 0),
            'karmic': qt.basis(3, 1),
            'void': qt.basis(3, 2)
        }
        self.operators = self._create_operators()
        self.current_state = self.states['void']

    def _create_operators(self):
        """Crea los operadores cuanticos para el sistema"""
        # Operadores de proyeccion
        P0 = qt.basis(3, 0) * qt.basis(3, 0).dag()
        P1 = qt.basis(3, 1) * qt.basis(3, 1).dag()
        P2 = qt.basis(3, 2) * qt.basis(3, 2).dag()

        # Operadores de transicion
        S01 = qt.basis(3, 0) * qt.basis(3, 1).dag()
        S12 = qt.basis(3, 1) * qt.basis(3, 2).dag()
        S20 = qt.basis(3, 2) * qt.basis(3, 0).dag()

        # Hamiltoniano base
        H0 = P0 * 0.1 + P1 * 0.2 + P2 * 0.3

        # Operador karmico
        K = self.karma_params['attachment'] * (S01 + S01.dag()) + \
            self.karma_params['clarity'] * (S12 + S12.dag()) + \
            self.karma_params['compassion'] * (S20 + S20.dag())

        return {
            'P0': P0, 'P1': P1, 'P2': P2,
            'S01': S01, 'S12': S12, 'S20': S20,
            'H0': H0, 'K': K
        }

    def _calculate_coherence(self, state):
        """Calcula la coherencia cuantica del estado"""
        return self.metrics.coherence(state)

    def _calculate_purity(self, state):
        """Calcula la pureza del estado"""
        return self.metrics.purity(state)

    def _attention_evolution(self, t, attention_function='logistic'
```

```
        ):
117         """Evolucion de la atencion en el tiempo"""
118         if attention_function == 'logistic':
119             return 1.0 / (1.0 + np.exp(-0.5 * (t - 2*np.pi)))
120         elif attention_function == 'sinusoidal':
121             return 0.5 * (1.0 + np.sin(t))
122         else:  # constante
123             return 1.0
124 
125     def simulate_bardo_transition(self, time_steps=1000,
        attention_function='logistic'):
126         """Simula la transicion completa a traves de los estados
            del Bardo"""
127         times = np.linspace(0, self.time_parameters['total_time'],
            time_steps)
128         results = {
129             'probabilities': [],
130             'coherence': [],
131             'purity': [],
132             'states': []
133         }
134 
135         current_state = self.current_state
136 
137         for t in times:
138             # Factor de atencion dependiente del tiempo
139             attention = self._attention_evolution(t,
                attention_function)
140 
141             # Evolucion unitaria con Hamiltoniano modificado
142             H_eff = self.operators['H0'] + attention * self.
                operators['K']
143             U = (-1j * t * H_eff).expm()
144             evolved_state = U * current_state
145 
146             # Calculo de metricas
147             probs = [qt.expect(self.operators[f'P{i}'],
                evolved_state) for i in range(3)]
148             coherence = self._calculate_coherence(evolved_state)
149             purity = self._calculate_purity(evolved_state)
150 
151             results['probabilities'].append(probs)
152             results['coherence'].append(coherence)
153             results['purity'].append(purity)
154             results['states'].append(evolved_state)
155 
156             current_state = evolved_state
157 
158         return results, times
159 
160     def run_complete_simulation(self):
```

```
161         """Ejecuta una simulacion completa y retorna resultados"""
162         results, times = self.simulate_bardo_transition()
163 
164         # Analisis adicional
165         final_entropy = self.metrics.von_neumann_entropy(results['
               states'][-1])
166         avg_coherence = np.mean(results['coherence'])
167         avg_purity = np.mean(results['purity'])
168 
169         analysis_report = {
170             'final_entropy': final_entropy,
171             'average_coherence': avg_coherence,
172             'average_purity': avg_purity,
173             'final_state_type': self._classify_final_state(results[
                   'states'][-1]),
174             'transition_statistics': self._analyze_transitions(
                   results)
175         }
176 
177         return results, times, analysis_report
178 
179     def _classify_final_state(self, state):
180         """Clasifica el estado final segun las probabilidades"""
181         probs = [abs(state[i,0])**2 for i in range(3)]
182         max_prob_index = np.argmax(probs)
183         states_names = ['Samsara', 'Karmico', 'Vacuidad']
184         return {
185             'dominant_state': states_names[max_prob_index],
186             'probabilities': probs,
187             'certainty': max(probs)
188         }
189 
190     def _analyze_transitions(self, results):
191         """Analiza las transiciones entre estados"""
192         probs = np.array(results['probabilities'])
193         transitions = []
194 
195         for i in range(1, len(probs)):
196             # Detectar cambios significativos en probabilidades
197             changes = np.abs(probs[i] - probs[i-1])
198             if np.max(changes) > 0.1:  # umbral arbitrario
199                 transitions.append({
200                     'step': i,
201                     'changes': changes.tolist(),
202                     'from_state': np.argmax(probs[i-1]),
203                     'to_state': np.argmax(probs[i])
204                 })
205 
206         return {
207             'total_transitions': len(transitions),
208             'transition_sequence': transitions,
```

```
            'stability_analysis': self._calculate_stability(probs)
        }

    def _calculate_stability(self, probabilities):
        """Calcula metricas de estabilidad del sistema"""
        probs = np.array(probabilities)
        variances = np.var(probs, axis=0)

        return {
            'variance_samsara': variances[0],
            'variance_karmic': variances[1],
            'variance_void': variances[2],
            'overall_stability': 1.0 - np.mean(variances),
            'stationary_points': self._find_stationary_points(probs
               )
        }

    def _find_stationary_points(self, probabilities):
        """Encuentra puntos donde el sistema parece estabilizarse
           """
        probs = np.array(probabilities)
        gradients = np.gradient(probs, axis=0)
        gradient_norms = np.linalg.norm(gradients, axis=1)

        stationary_indices = np.where(gradient_norms < 0.01)[0]  #
           umbral pequeno
        return stationary_indices.tolist()
```

Listing 4: Implementacion completa de BardoQuantumSystem

A.2. Visualizaciones Cientificas Avanzadas

```
class QuantumVisualizer:
    """Sistema completo de visualizacion cientifica para estados
       del Bardo"""

    def __init__(self, style_params=None):
        self.style_params = style_params or {
            'cmap': 'viridis',
            'dpi': 300,
            'font_size': 12,
            'figure_size': (16, 12)
        }
        self.set_plotting_style()

    def set_plotting_style(self):
        """Configura estilo cientifico para publicaciones"""
        plt.rcParams.update({
            'font.size': self.style_params['font_size'],
            'axes.titlesize': self.style_params['font_size'] + 2,
            'axes.labelsize': self.style_params['font_size'] + 1,
```

```
            'xtick.labelsize': self.style_params['font_size'] - 1,
            'ytick.labelsize': self.style_params['font_size'] - 1,
            'legend.fontsize': self.style_params['font_size'] - 1,
            'figure.dpi': self.style_params['dpi'],
            'savefig.dpi': self.style_params['dpi'],
            'figure.figsize': self.style_params['figure_size']
        })

    # [Todos los metodos de visualizacion anteriores se mantienen igual...]
    # _plot_temporal_evolution, _plot_bloch_sphere, _plot_density_matrix, etc.

    def _analyze_transitions(self, results):
        """Analiza las transiciones entre estados - IMPLEMENTACION CORREGIDA"""
        probs = np.array(results['probabilities'])
        transitions = []

        for i in range(1, len(probs)):
            changes = np.abs(probs[i] - probs[i-1])
            if np.max(changes) > 0.1:
                transitions.append({
                    'time_index': i,
                    'changes': changes.tolist(),
                    'from_state': np.argmax(probs[i-1]),
                    'to_state': np.argmax(probs[i])
                })
        return transitions

    def _find_dominant_state(self, results):
        """Encuentra el estado dominante - IMPLEMENTACION CORREGIDA"""
        probs = np.array(results['probabilities'])
        dominant_states = np.argmax(probs, axis=1)

        return {
            'dominant_states': dominant_states.tolist(),
            'time_in_samsara': np.sum(dominant_states == 0),
            'time_in_karmic': np.sum(dominant_states == 1),
            'time_in_void': np.sum(dominant_states == 2),
            'dominance_ratio': {
                'samsara': np.sum(dominant_states == 0) / len(dominant_states),
                'karmic': np.sum(dominant_states == 1) / len(dominant_states),
                'void': np.sum(dominant_states == 2) / len(dominant_states)
            }
        }

```

```
63      def calculate_von_neumann_entropy(self, state):
64          """Calcula entropia de Von Neumann - USANDO QuantumMetrics
               """
65          metrics = QuantumMetrics()
66          return metrics.von_neumann_entropy(state)
67
68      def generate_analysis_report(self, results, filename=None):
69          """Genera reporte analitico completo - IMPLEMENTACION
               CORREGIDA"""
70          report = {
71              'final_probabilities': results['probabilities'][-1],
72              'max_coherence': max(results['coherence']),
73              'min_purity': min(results['purity']),
74              'state_transitions': self._analyze_transitions(results)
                   ,
75              'dominant_state_analysis': self._find_dominant_state(
                   results),
76              'entropy_analysis': {
77                  'initial_entropy': self.
                       calculate_von_neumann_entropy(results['states'
                       ][0]),
78                  'final_entropy': self.calculate_von_neumann_entropy
                       (results['states'][-1]),
79                  'avg_entropy': np.mean([self.
                       calculate_von_neumann_entropy(s)
80                                          for s in results['states']])
81              }
82          }
83
84          if filename:
85              with open(filename, 'w', encoding='utf-8') as f:
86                  json.dump(report, f, indent=2, ensure_ascii=False)
87
88          return report
89
90      def create_comprehensive_dashboard(self, results, times,
           save_path=None):
91          """Crea un dashboard completo de visualizaciones"""
92          fig = plt.figure(figsize=(24, 18))
93
94          # 1. Evolucion temporal
95          ax1 = self._plot_temporal_evolution(fig, 3, 3, 1, results)
96
97          # 2. Espacio de estados 3D
98          ax2 = self._plot_bloch_sphere(fig, 3, 3, 2, results)
99
100         # 3. Matriz de densidad
101         ax3 = self._plot_density_matrix(fig, 3, 3, 3, results)
102
103         # 4. Metricas cuanticas
104         ax4 = self._plot_quantum_metrics(fig, 3, 3, 4, results)
```

```
105
106         # 5. Transiciones de fase
107         ax5 = self._plot_phase_transitions(fig, 3, 3, 5, results)
108
109         # 6. Circuito cuantico
110         ax6 = self._plot_quantum_circuit(fig, 3, 3, 6)
111
112         # 7. Analisis de entropia
113         ax7 = self._plot_entropy_evolution(fig, 3, 3, 7, results,
                times)
114
115         # 8. Histograma de estados dominantes
116         ax8 = self._plot_dominant_states_histogram(fig, 3, 3, 8,
                results)
117
118         # 9. Correlaciones entre metricas
119         ax9 = self._plot_metric_correlations(fig, 3, 3, 9, results)
120
121         plt.tight_layout()
122
123         if save_path:
124             plt.savefig(save_path, dpi=300, bbox_inches='tight',
125                         facecolor='white', edgecolor='none')
126
127         return fig
128
129     def _plot_entropy_evolution(self, fig, nrows, ncols, index,
            results, times):
130         """Grafica la evolucion de la entropia de Von Neumann"""
131         ax = fig.add_subplot(nrows, ncols, index)
132
133         entropies = [self.calculate_von_neumann_entropy(state)
134                      for state in results['states']]
135
136         ax.plot(times, entropies, color='brown', linewidth=2)
137         ax.set_xlabel('Tiempo')
138         ax.set_ylabel('Entropia de Von Neumann')
139         ax.set_title('Evolucion de la Entropia Cuantica')
140         ax.grid(True, alpha=0.3)
141
142         return ax
143
144     def _plot_dominant_states_histogram(self, fig, nrows, ncols,
            index, results):
145         """Grafica histograma de estados dominantes"""
146         ax = fig.add_subplot(nrows, ncols, index)
147
148         dominant_analysis = self._find_dominant_state(results)
149         ratios = dominant_analysis['dominance_ratio']
150
151         states = ['Samsara', 'Karmico', 'Vacuidad']
```

```
        values = [ratios['samsara'], ratios['karmic'], ratios['void']]
        colors = ['red', 'blue', 'green']

        bars = ax.bar(states, values, color=colors, alpha=0.7)
        ax.set_ylabel('Fraccion de Tiempo')
        ax.set_title('Distribucion de Estados Dominantes')

        # Anotar valores en las barras
        for bar, value in zip(bars, values):
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                    f'{value:.3f}', ha='center', va='bottom')

        return ax

    def _plot_metric_correlations(self, fig, nrows, ncols, index, results):
        """Grafica correlaciones entre diferentes metricas cuanticas"""
        ax = fig.add_subplot(nrows, ncols, index)

        coherences = results['coherence']
        purities = results['purity']
        entropies = [self.calculate_von_neumann_entropy(state)
                     for state in results['states']]

        scatter = ax.scatter(coherences, purities, c=entropies,
                             cmap='plasma', s=30, alpha=0.6)
        ax.set_xlabel('Coherencia Cuantica')
        ax.set_ylabel('Pureza del Estado')
        ax.set_title('Correlacion: Coherencia vs Pureza\n(Color: Entropia)')

        plt.colorbar(scatter, ax=ax, label='Entropia de Von Neumann')
        ax.grid(True, alpha=0.3)

        return ax
```

Listing 5: Sistema completo de visualizacion cuantica corregido

Referencias

[1] Fremantle, F. (2001). *The Tibetan Book of the Dead*. Shambhala Publications.

[2] Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information*. Cambridge University Press.

[3] Hameroff, S., & Penrose, R. (2014). Consciousness in the universe: A review of the 'Orch OR' theory. *Physics of Life Reviews*, 11(1), 39-78.

[4] Wallace, B. A. (2007). *Contemplative Science: Where Buddhism and Neuroscience Converge*. Columbia University Press.

[5] Lanyon, B. P., et al. (2008). Manipulating biphotonic qutrits. *Physical Review Letters*, 100(6), 060504.

[6] Tegmark, M. (2000). Importance of quantum decoherence in brain processes. *Physical Review E*, 61(4), 4194.