TEORÍA Σ

Códigos de Simulación Computacional

Fernando Figueroa Gutiérrez

1. Simulación de Percolación (Sección 15)

Esta simulación implementa la red de correlaciones C(i,j) = exp(-σ·d(i,j)) y muestra los tres regímenes de Σ mediante percolación.

1.1. Código Python

```
#!/usr/bin/env python3
"""
Teoría Σ - Simulación de Percolación
Sección 15: Evidencia Computacional

Implementa la función de correlación C(i,j) = exp(-σ·d(i,j))
y demuestra la emergencia de los tres regímenes sin parámetros libres.
"""

import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

# Parámetros fundamentales
N = 200  # Número de nodos (estados distinguibles de Σ)
SIGMA_C = np.log(2) * N  # Punto crítico teórico: σ_c = ln(2)·N ≈ 138.6

def build_network(N, sigma, seed=42):
    """
    Construir red de correlaciones para un valor dado de σ.

    Parámetros:
```

```
    - N: número de nodos
    - sigma: densidad de excitación del sustrato
    - seed: semilla para reproducibilidad

    Retorna:
    - G: grafo de NetworkX con nodos conectados según C(i,j) > 0.5
    """
    np.random.seed(seed)
    threshold = np.log(2) / sigma  # C > 0.5 ⇔ d < ln(2)/σ

    G = nx.Graph()
    G.add_nodes_from(range(N))

    # Generar distancias relacionales y conectar según correlación
    for i in range(N):
        for j in range(i + 1, N):
            d_ij = np.random.random()  # Distancia relacional primitiva
            C_ij = np.exp(-sigma * d_ij)  # Función de correlación

            if C_ij > 0.5:  # Criterio de conexión
                G.add_edge(i, j)

    return G

def compute_observables(G, N):
    """
    Calcular observables de la red.

    Retorna:
    - giant: fracción de nodos en componente gigante
    - second: fracción de nodos en segunda componente
    - n_comp: número total de componentes
    - avg_C: correlación media
    """
    # Componentes conectadas
    components = sorted(nx.connected_components(G), key=len, reverse=True)

    giant = len(components[0]) / N if components else 0.0
    second = len(components[1]) / N if len(components) >= 2 else 0.0
```

```
    n_comp = len(components)

    # Correlación media (estimada desde grado medio)
    avg_degree = sum(dict(G.degree()).values()) / N
    avg_C = avg_degree / (N - 1) if N > 1 else 0.0

    return giant, second, n_comp, avg_C

# ============================================================
# SIMULACIÓN PRINCIPAL
# ============================================================

sigma_values = np.linspace(0.1 * SIGMA_C, 3.0 * SIGMA_C, 100)
results = {
    'sigma': [],
    'giant': [],
    'second': [],
    'n_comp': [],
    'avg_C': []
}

print("Ejecutando simulación de percolación...")
for sigma in sigma_values:
    G = build_network(N, sigma)
    giant, second, n_comp, avg_C = compute_observables(G, N)

    results['sigma'].append(sigma / SIGMA_C)  # Normalizado
    results['giant'].append(giant)
    results['second'].append(second)
    results['n_comp'].append(n_comp)
    results['avg_C'].append(avg_C)

# ============================================================
# GRÁFICA: TRANSICIÓN DE FASE
# ============================================================

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(results['sigma'], results['giant'], 'g-', lw=2, label='Componente gigante')
```

```
ax.plot(results['sigma'], results['second'], 'y--', lw=2, label='2ª componente')
ax.plot(results['sigma'], results['avg_C'], 'b:', lw=2, label='⟨C(i,j)⟩')

ax.axvline(1.0, color='orange', ls='--', lw=1, label='σ_c')
ax.axvspan(0.85, 1.15, alpha=0.1, color='orange', label='Zona crítica')

ax.set_xlabel('σ / σ_c', fontsize=12)
ax.set_ylabel('Fracción / Correlación', fontsize=12)
ax.set_title('Transición de Fase de Σ (N = 200)', fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('teoria_sigma_percolacion.png', dpi=150)
plt.show()

print(f"\n✓ Simulación completada")
print(f"  Punto crítico teórico: σ_c = {SIGMA_C:.2f}")
print(f"  Observado: pico de 2ª componente en σ ≈ σ_c")
print(f"  Régimen lineal: σ < {0.85*SIGMA_C:.1f} (espacio conectado)")
print(f"  Régimen crítico: σ ≈ {SIGMA_C:.1f} (transición)")
print(f"  Régimen saturación: σ > {1.15*SIGMA_C:.1f} (fragmentación)")
```

2. Simulación Termodinámica Completa

Extensión con cálculos de entropía, calor específico, compresibilidad e información cuántica.

2.1. Código React/JavaScript (Artifact Interactivo)

Este código está disponible en los artifacts de Claude - ver archivo 'sigma-complete-simulation.jsx'

Características principales:

• Visualización de red 2D con nodos y conexiones • Propagación de ondas (ecos gravitacionales) • Cálculos termodinámicos: S_shannon, S_BH, C_v, χ • Verificación I_max = 0.057 bits • Animación de colapso gravitacional • 4 modos de visualización interactivos

3. Visualización de Red 2D

3.1. Código Python con NetworkX

```
#!/usr/bin/env python3
"""
Visualización 2D de la red de Σ
Muestra nodos, conexiones y componentes
"""

import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

def visualize_network_2d(sigma, N=120):
    """Visualizar red en 2D para un valor de σ"""

    # Construir red
    np.random.seed(42)
    threshold = np.log(2) / sigma

    G = nx.Graph()
    G.add_nodes_from(range(N))

    for i in range(N):
        for j in range(i + 1, N):
            if np.random.random() < threshold:
                G.add_edge(i, j)

    # Identificar componente gigante
    components = sorted(nx.connected_components(G), key=len, reverse=True)
    giant_nodes = components[0] if components else set()

    # Layout de la red
    pos = nx.spring_layout(G, seed=42, k=0.6)

    # Colores según componente
    node_colors = ['green' if n in giant_nodes else 'gray' for n in G.nodes()]
    edge_colors = ['blue' if (u in giant_nodes and v in giant_nodes)
                   else 'lightgray' for u, v in G.edges()]

    # Dibujar
```

```
    plt.figure(figsize=(10, 10))
    nx.draw_networkx_edges(G, pos, edge_color=edge_colors, alpha=0.3, width=0.5)
    nx.draw_networkx_nodes(G, pos, node_color=node_colors, node_size=30, alpha=0.8)

    plt.title(f'Red de Σ: σ/σ_c = {sigma/SIGMA_C:.2f}', fontsize=14, fontweight='bold')
    plt.axis('off')
    plt.tight_layout()
    plt.show()

# Visualizar los tres regímenes
SIGMA_C = np.log(2) * 120

print("Generando visualizaciones...")
visualize_network_2d(0.3 * SIGMA_C)  # Lineal
visualize_network_2d(0.95 * SIGMA_C)  # Crítico
visualize_network_2d(1.5 * SIGMA_C)   # Saturación
```

4. Notas de Implementación

Dependencias Python:

• numpy >= 1.24 • matplotlib >= 3.6 • networkx >= 3.0

Instalación:

```
pip install numpy matplotlib networkx
```

Ejecución:

```
python3 simulacion_percolacion.py
```

Códigos completos disponibles en:
https://github.com/ffigueroa/teoria-sigma