Modelo de Ising:
Apreciaciones Generales y Algoritmo Metrópolis.

Juan Esteban Aristizabal Zuluaga
Instituto de Física, Universidad de Antioquia.

(Dated: 16 de mayo de 2020)

Palabras clave: .

I. INTRODUCCIÓN

II. CONSIDERACIONES TEÓRICAS

A. Hamiltoniano del sistema

B. Microestados y contribuciones a la función partición

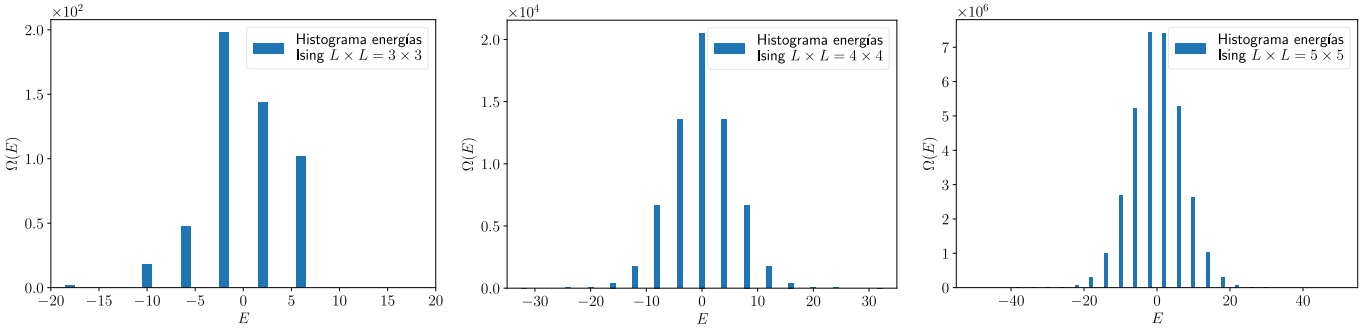

Figura 1.

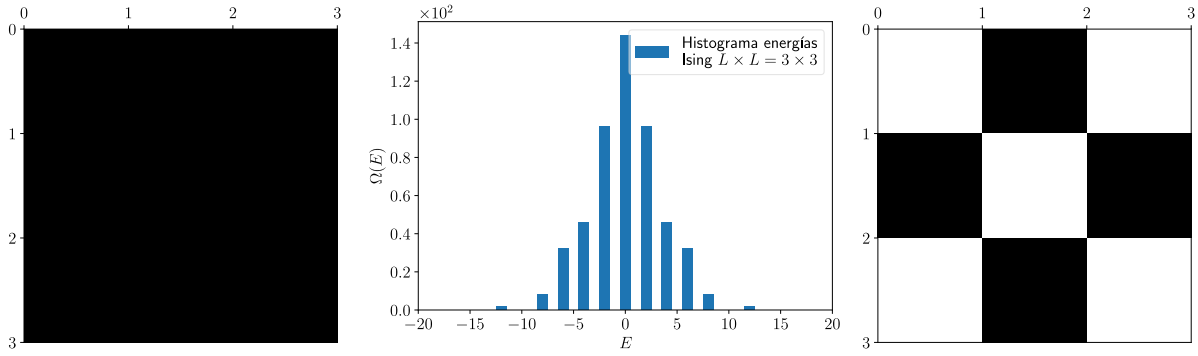

Figura 2.

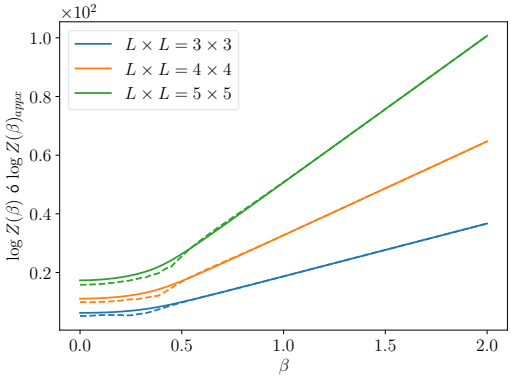

Figura 3.

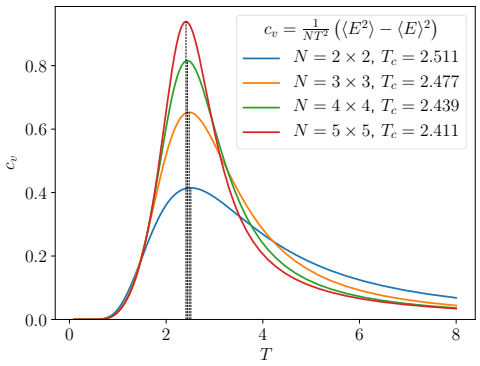

Figura 4.

C. Equivalencia entre ensambles microcanónico y macrocanónico

D. Teorema de fluctuación-disipación y calor específico

III. MODELO DE ISING Y ALGORITMO METRÓPOLIS

A. El algoritmo

```python
# -*- coding: utf-8 -*-
import numpy as np
from ising2d_metropolis import ising_neighbours, ising_energy

def ising_metropolis_energies(microstate=np.ones(36,dtype=np.int64),
                              read_ini_microstate_data=False, L=6, beta=1., J=1,
                              N_steps=10000, N_transient=100):

    N = L * L
    # Calcula vecinos
    ngbrs = ising_neighbours(L)

    # Si los datos se no se leyeron, genera microestado inicial aleatoriamente
    if read_ini_microstate_data:
        pass
    else:
```

```
        microstate = np.random.choice(np.array([1,-1]), N)

    # Calcula energía inicial
    energy = ising_energy([microstate], ngbrs, J=J, print_log=False)[0]
    # Arreglo donde se guardarán energías de los microestados muestreados
    energies = []

    # En el transiente no se guardan las energías,
    # se espera a que el sistema se termalice.
    for i in range(N_transient):
        k = np.random.randint(N)
        delta_E = (2. * J * microstate[k]
                   * np.sum(np.array([microstate[ngbr_i] for ngbr_i in ngbrs[k]])))
        if  np.random.uniform(0,1) < np.exp(-beta * delta_E):
            microstate[k] *= -1
            energy += delta_E
    # Pasado el transiente, se comienzan a guardar las energías
    for i in range(N_steps):
        k = np.random.randint(N)
        delta_E = (2. * J * microstate[k]
                   * np.sum(np.array([microstate[ngbr_i] for ngbr_i in ngbrs[k]])))
        if np.random.uniform(0,1) < np.exp(-beta * delta_E):
            microstate[k] *= -1
            energy += delta_E
        energies.append(energy)

    # Se calcula la energía media por espín del microestado final
    N_steps2 = np.array(len(energies),dtype=np.int64)
    avg_energy_per_spin = np.float(np.sum(np.array(energies))/(N_steps2 * N * 1.))

    # Se devuelven las energías, el microestado final y la energía media
    # por espín del microestado final.
    return energies, microstate, avg_energy_per_spin
```

B. Termalización del sistema usando el algoritmo

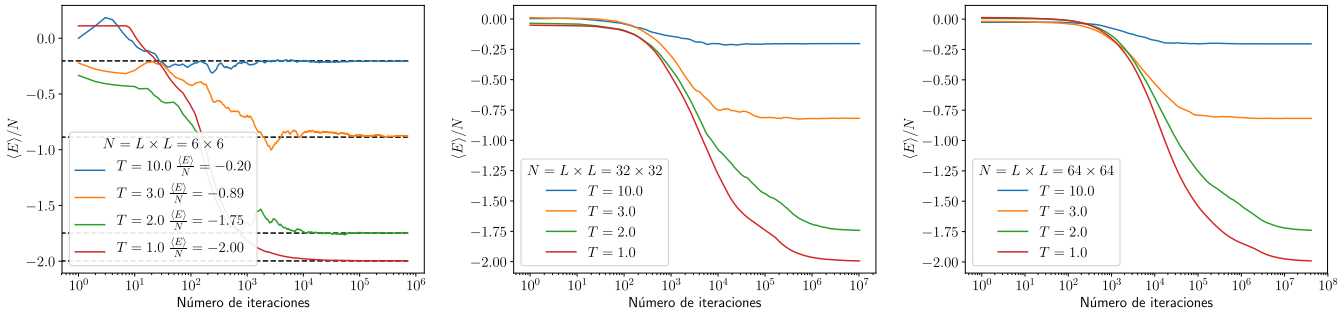

Figura 5.

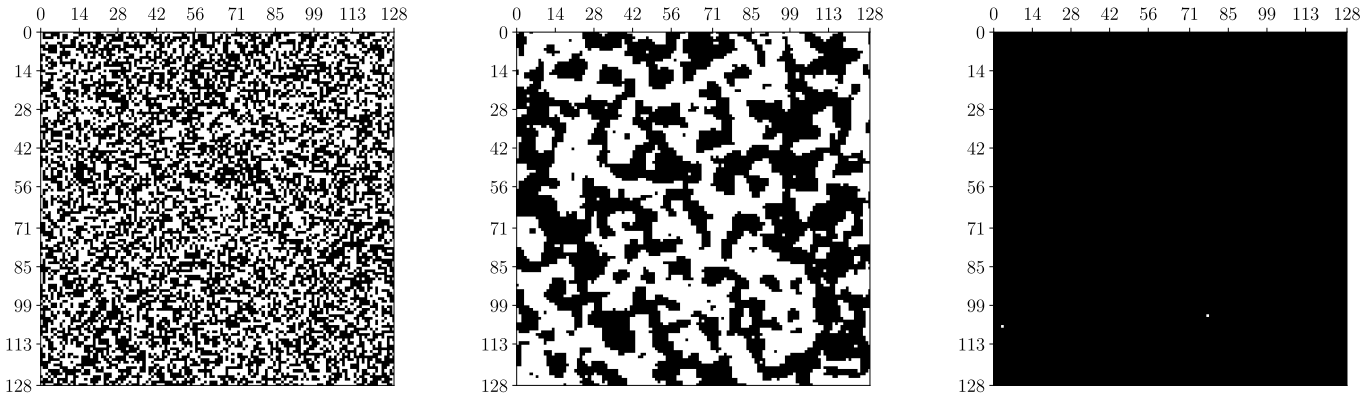

Figura 6.

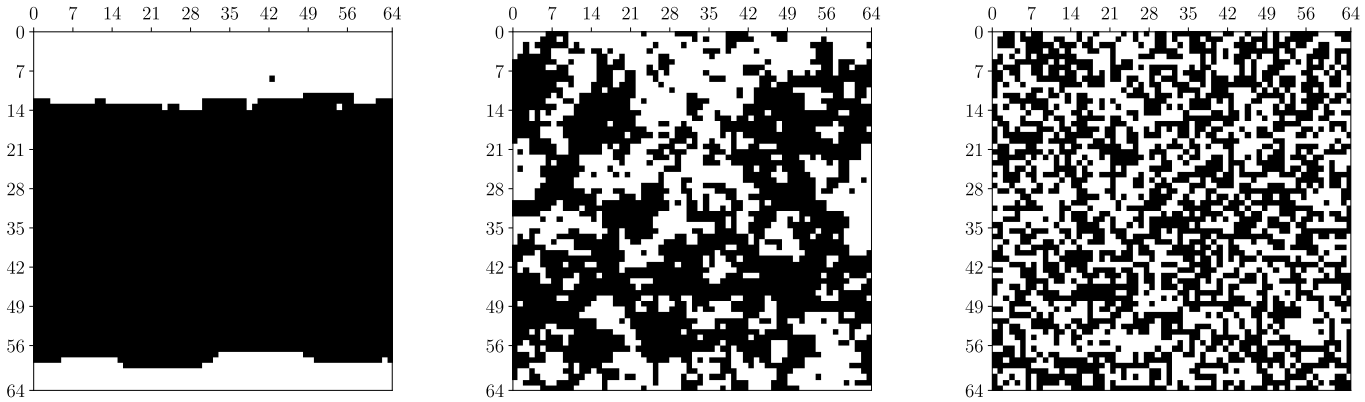

Figura 7.

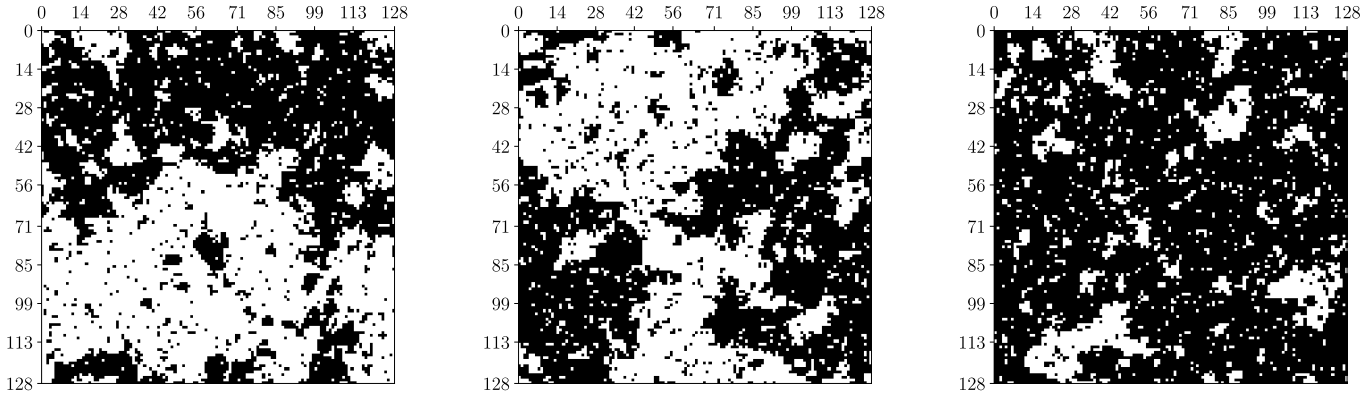

Figura 8.

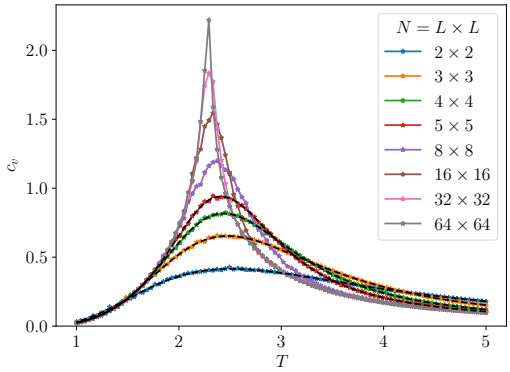

Figura 9.

C. Microestados finales y temperatura crítica

D. Calor específico

IV. CONCLUSIÓN

Las implementaciones de los algoritmos usados en este trabajo son suficientemente generales y se podrían adaptar con cierta facilidad a otros sistemas de interés que sean objeto de estudio.

AGRADECIMIENTOS

Agradezco a mis compañeros de clase con los que tuve discusiones que ayudaron en la implementación del algoritmo y en las conclusiones presentadas.

---

[1] S. K. Lam, A. Pitrou, and S. Seibert, Numba: A LLVM-Based Python JIT Compiler, in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15 (Association for Computing Machinery, New York, NY, USA, 2015).

Apéndice A: Código 1: Matrix Squaring

A continuación se muestra el código . Éste código está disponible en este link

Apéndice B: Código 2: Naive Path Integral Montecarlo Sampling

A continuación se muestra el código que está disponible en este link.