Modelo de Ising:
Apreciaciones Generales y Algoritmo Metrópolis.

Juan Esteban Aristizabal Zuluaga
Instituto de Física, Universidad de Antioquia.
(Dated: 17 de mayo de 2020)

Presentamos en este artículo un estudio medianamente detallado del modelo de Ising 2-dimensional en retícula cuadrada de lado L con condiciones de frontera periódicas. Introducimos el sistema en general y hacemos un estudio por enumeración exacta de los microestados del sistema hasta un tamaño $L \times L = 5 \times 5$. Estudiamos el fenómeno de transición de fase de segundo orden para estos sistemas de L pequeños y evidenciamos la complejidad computacional de la enumeración exacta para este modelo. Con esto en mente, se realiza un muestreo inteligente de los microestados del sistema con la implementación del algoritmo Metrópolis para el modelo en cuestión, el cual permite subir el valor de L hasta 128. En el algoritmo metrópolis notamos la importancia de la termalización del sistema y logramos evidenciar de primera mano el comienzo de la formación de la discontinuidad en el calor específico c_v, lo cual es característica de una transición de fase de segundo orden.

Palabras clave: Modelo de Ising, conteo de microestados, ensamble canónico, calor específico, transición de fase, algoritmo metrópolis, termalización.

I. CONSIDERACIONES TEÓRICAS

A. Hamiltoniano del sistema y ensamble canónico.

El hamiltoniano del modelo de Ising está dado por

$$H = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j, \tag{1}$$

donde $\sigma_i = \pm 1$ (se conoce como espín), $\langle ij \rangle$ da cuenta de interacción entre primeros vecinos y J es la fuerza de la interacción entre primeros vecinos.

El modelo de Ising asume que los espines están fijos en sus posiciones y puede ser aplicado en cualquier dimensionalidad y con cualquier tipo de rejilla. En nuestro caso nos centraremos en el sistema 2-dimensional con rejilla cuadrada de lado L. Es decir, estudiaremos la interacción del modelo de ising entre $N = L \times L$ espines. Las condiciones de frontera del sistema pueden ser de diversos tipos. En este trabajo usaremos más que todo las condiciones de frontera periódicas y mencionaremos algunas diferencias interesantes que emergen al considerar condiciones de frontera libres. Además, trabajaremos en unidades de $J = k_B = 1$.

Cuando el sistema está inmerso en un baño térmico, podemos trabajar en el ensamble canónico. La función partición está dada por

$$Z(\beta) = \sum_n e^{-\beta E_n}, \tag{2}$$

donde E_n representa cada nivel de energía del sistema. Si el microestado $E_n = E$ tiene degeneración $\Omega(E)$ es claro que

$$Z(\beta) = \sum_E \Omega(E) e^{-\beta E}. \tag{3}$$

Notemos que esta última expresión es conceptualmente muy útil, ya que vemos que las contribuciones a la función partición se pueden agrupar de acuerdo a las diferentes energías del sistema, via su degeneración $\Omega(E)$. En esta misma línea, sabemos también que probabilidad de que el sistema esté en un nivel de energía E_n, p_n, está dada por

$$p_n = \frac{e^{-\beta E_n}}{Z} \tag{4}$$

y la probabilidad de que el sistema tenga energía E está dada por

$$p_E = \frac{\Omega(E) e^{-\beta E}}{Z}. \tag{5}$$

Cuando tenemos este sistema en el ensamble canónico y L es lo suficientemente grande, se presenta una transición de fase de segundo orden caracterizada por una discontinuidad en el calor específico c_v y se da a una temperatura crítica T_c. Cuando $L \to \infty$, se tiene $T_c = 2/\log\left(1+\sqrt{2}\right) \approx 2.269$ [1].

A pesar de que la escencia del modelo de Ising es muy sencilla, nos encontramos con un problema importante al estudiarlo (que se encuentra en gran parte de la física estadística), el cual es el conteo de microestados. El número de microestados del sistema 2-dimensional en rejilla cuadrada es $\Omega = 2^{L\times L}$. Para el caso $L = 10$ esto es $\Omega \approx 10^{30}$. El cálculo del total de éste número de microestados rebasa cualquier capacidad computacional disponible hasta el momento. Entre otras cosas, para darnos una idea de este problema, estudiaremos en las siguientes secciones el cálculo de los microestados por enumeración exacta.

B. Microestados y contribuciones a la función partición

Los microestados explícitos, es decir, las $\Omega = 2^{L\times L}$ configuraciones posbiles en nuestro sistems son isomorfas a los primeros Ω números binarios. Con esta estrategia se pueden generar todas las configuraciones posibles, cambiando los dígitos que tengan valor 0 de los números binarios por el valor -1. Una vez generadas todas las configuraciones para un L dado, se puede calcular la energía de cada uno de ellos usando la expresión 1 y con ellas, se pueden calcular los $\Omega(E)$.

Nota: el código con el que se generaron las figuras se puede ver en el apéndice A y usa un módulo propio que escribimos, el cual contiene todas las funciones necesarias para generar las gráficas y para el cual se adjunta el link en el anexo en cuestión.

La figura 1 muestra el histograma de $\Omega(E)$ para $L = 3,\ 4,\ 5$ con condiciones de frontera periódicas. Notamos que el caso $L = 3$ es evidentemente asimétrico con respecto a la energía $E = 0$, contrario al caso $L = 4$ que es simétrico con respecto a la misma energía. El caso $L = 5$ es aparentemente simétrico en el mismo sentido. Sin embargo, al calcular los valores de energía exactos encontramos que este caso es realmente asimétrico (por ejemplo, para la mínima energía $E = -50$ se tiene $\Omega(-50)$, mientras que $\Omega(50) = 0$). En general, se tiene que para L impar, el histograma de $\Omega(E)$ es asimétrico y para L par es simétrico. Sin embargo, para L lo suficientemente grande, la asimetría del histograma en cuestión es mínima y se puede considerar como simétrico. Como veremos a continuación, se trata de un efecto de tamaño finito que es consecuencia de las condiciones de frontera periódicas.

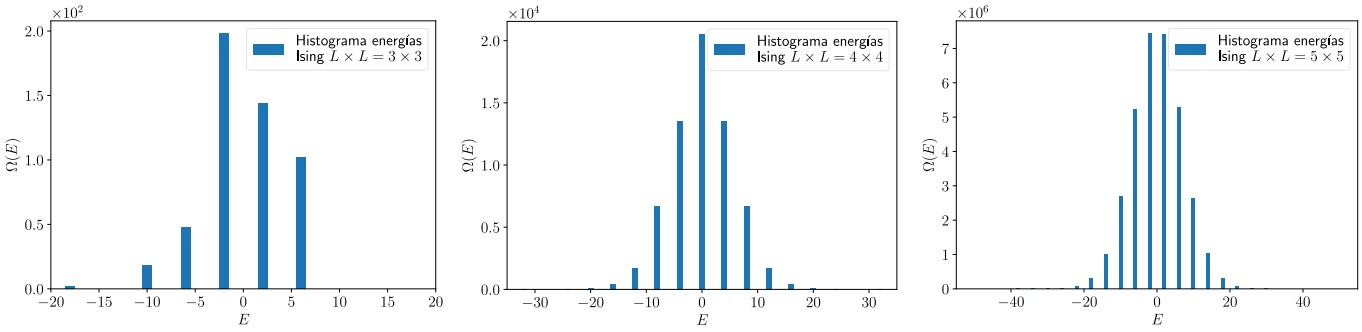

Figura 1. De izquierda a derecha: histograma de $\Omega(E)$ para $L = 3,\ 4,\ 5$, respectivamente, con condiciones de frontera periódicas. Notamos la asimetría evidente para $L = 3$, simetría para $L = 4$ y asimetría no evidente en $L = 5$. Este fenómeno se debe a las condiciones de frontera periódicas y es un efecto de tamaño finito, pues para L lo suficientemente grande, el histograma es aproximadamente simétrico independientemente de si L es impar o par.

Para entender un poco mejor la asimetría en los histogramas con L impar, consideremos momentáneamente el caso de condiciones de frontera libres. En la figura 2 (centro) se muestra dicho histograma para $L = 3$. Notamos que el histograma ahora contiene la energía $E = 0$ y que es simétrico (no aparentemente sino exactamente simétrico). Para entender mejor el efecto de las condiciones de frontera se muestran en la misma figura configuraciones de menor y mayor energía (izquierda y derecha, respectivamente). Al considerar condiciones de frontera periódicas, para el caso de menor energía (izquierda) con condiciones de frontera libres se tiene $E = -12$ y con condiciones de frontera periódicas, se tiene $E = -18$. Para el caso de mayor energía con condiciones de frontera libres se tiene $E = 12$, mientras que en el caso de condiciones de frontera periódicas se tiene $E = 6$.

Este efecto se da para el caso impar ya que al tener condiciones de frontera periódicas, en el caso de mayor energía, los vecinos de los sitios de los bordes tienen el mismo valor de espín, lo cual hace que la energía disminuya con respecto al valor absoluto de la energía menor del sistema y así se genera la asimetría, es decir $E_{\max} = |E_{\min}| - \Delta E$. Dicho

ΔE está relacionado con las condiciones de frontera. $\Delta E = 0$ para condiciones de frontera libres, lo cual genera la simetría en ese caso y es fácil ver que $\Delta E = 2L$ para condiciones de frontera periódicas. La asimetría se 'pierde' con $L \to \infty$ ya que para condiciones de frontera periódicas $E_{\min} = -2L^2$, luego

$$\begin{aligned} & E_{\max} = 2(L^2 - L) \\ \implies & \frac{E_{\max}}{L} = 2(L-1) \\ & \approx 2L = \frac{|E_{\min}|}{L}. \end{aligned} \tag{6}$$

Por otro lado, este efecto no se presenta en el caso $L =$ par, ya que todos los vecinos de los sitios de los bordes son de signos diferentes.

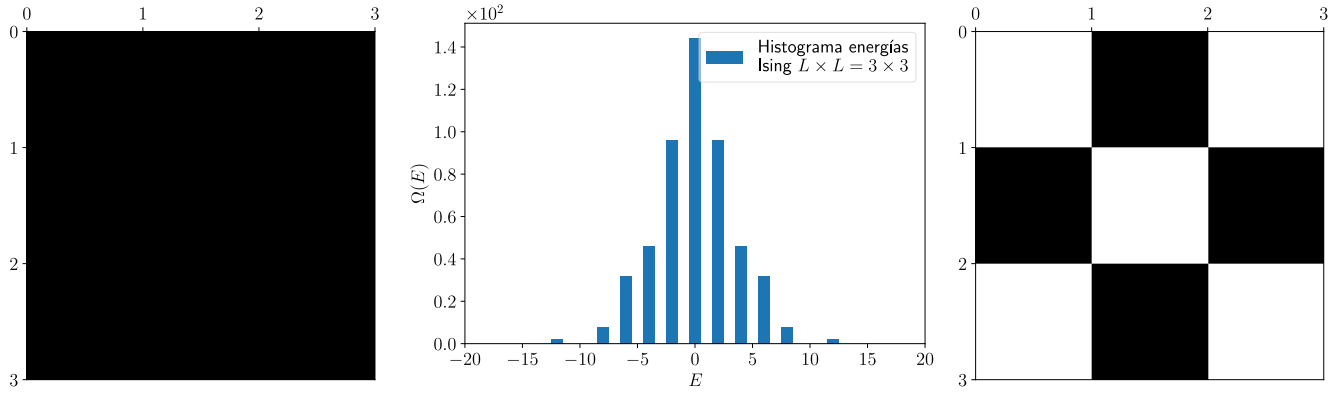

Figura 2. Centro: histograma $\Omega(E)$ para el caso $L = 3$ con condiciones de frontera periódicas. Izquierda: un microestado de menor energía (independientemente de las condiciones de frontera). Derecha: un microestado de mayor energía (independientemente de las condiciones de frontera).

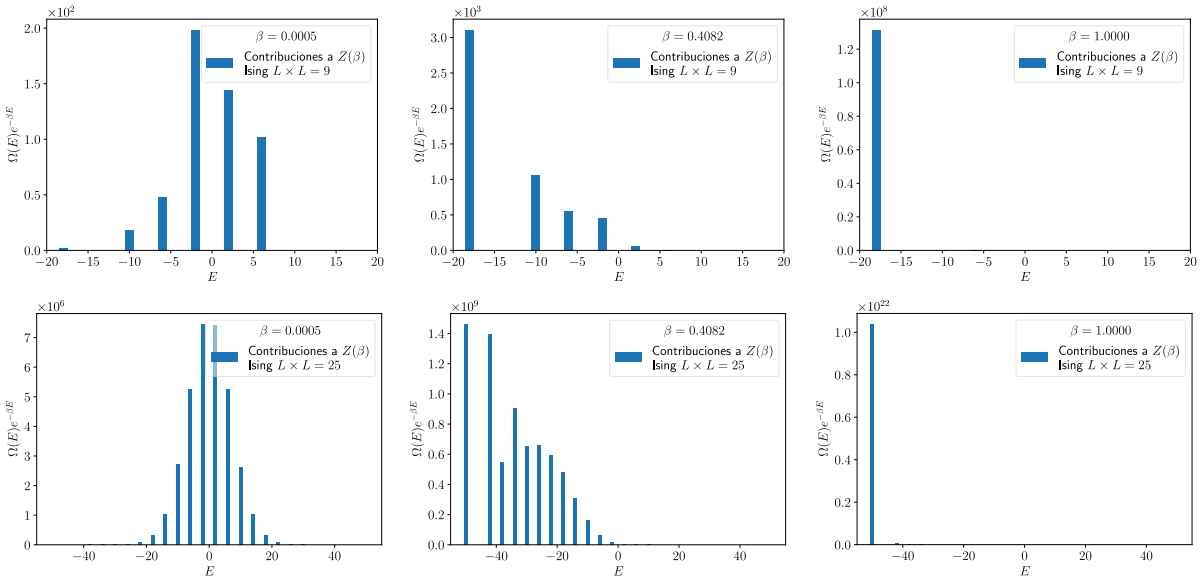

Figura 3. Contribuciones por energías a la función partición $\Omega(E)e^{-\beta E} \propto p_E$. Arriba: $N = 3 \times 3$, abajo: $N = 5 \times 5$. Izquierda: $T = 2000$, centro: $T = T_c \approx 2.45$ (temperatura crítica para valores pequeños de L), derecha: $T = 1$.

Por otro lado, sabemos que la función partición tiene gran importancia en la física estadística, ya que de ella se deduce la termodinámica del sistema. Notemos que las contribuciones de cada energía a la función partición, *i.e.*

$\Omega(E)e^{-\beta E}$ son proporcionales a la probabilidad p_E definida en 5. Conocerlas es básicamente conocer la probabilidad de que el sistema se encuentre en una energía E dada.

En la figura 3 mostramos dichas contribuciones por energías a la función partición $\Omega(E)e^{-\beta E} \propto p_E$. Arriba: $N = 3 \times 3$, abajo: $N = 5 \times 5$. Izquierda: $T = 2000$, centro: $T = T_c \approx 2.45$ (temperatura crítica para valores pequeños de L), derecha: $T = 1$. Para temperaturas muy altas, notamos, al comparar con la figura 1 que los histogramas son casi idénticos. Esto es esperable ya que $T \gg 0 \implies \beta \ll 1 \implies \Omega(E)e^{-\beta E} \approx \Omega(E)$. Por otro lado, cuando $T \approx T_c$ se tiene que hay mayores contribuciones de energías pequeñas pero aún hay muchas energías que contribuyen al sistema y las contribuciones son mayores cuanto menor es E. Éste es es un límite intermedio entre bajas y altas temperaturas, de algún modo T_c se convierte en una temperatura característica del sistema. Finalmente, el límite de bajas temperaturas es el esperado: prácticamente la única contribución es la del estado base.

C. Equivalencia entre ensambles microcanónico y macrocanónico

Usualmente encontramos en los textos de física estadística que en el caso del límite termodinámico hay una equivalencia entre los ensambles canónico y microcanónico, que se puede escribir en la forma

$$Z(\beta) = \sum_E \Omega(E)e^{-\beta E} \approx \Omega(\langle E\rangle)e^{-\beta\langle E\rangle} = Z(\beta)_{\text{appx}} \tag{7}$$

Aunque en esta sección no podemos calcular $Z(\beta)$ para sistemas muy grandes (por capacidad computacional solo se logró hacer hasta $L = 5$), se realizó una aproximación a este problema, para ver en cuáles son las condiciones en las cuales la expresión anterior se cumple en casos de baja dimensionalidad (L pequeño). El método usado fue, para cada β y L calcular la energía promedio $\langle E\rangle$, posteriormente se interpoló linealmente los histogramas de la figura 1 para obtener $\Omega(\langle E\rangle)$ y con esto se calculó Z_{appx} definida en (7).

En la figura 4, con líneas punteadas se muestra el resultado del método $\log Z_{\text{appx}}$ y con líneas continuas el valor real calculado directamente, $\log Z$. En general, encontramos que para temperaturas bajas *i.e.* β alto, la equivalencia se da casi de inmediato. Esto se debe básicamente a que la energía mínima es la más dominante en el sistema (por gran diferenca con respecto a las demás) y es muy cercana a la energía promedio. Por tanto, el sistema tiene prácticamente energía constante (casi sin fluctuaciones) *i.e.* el ensamble canónico y el microcanónico son equivalentes. Para poder concluir acerca de la validez de (7) en el límite termodinámico es necesario hacer este estudio para L grandes, lo cual está por fuera de nuestras capacidades de cómputo.

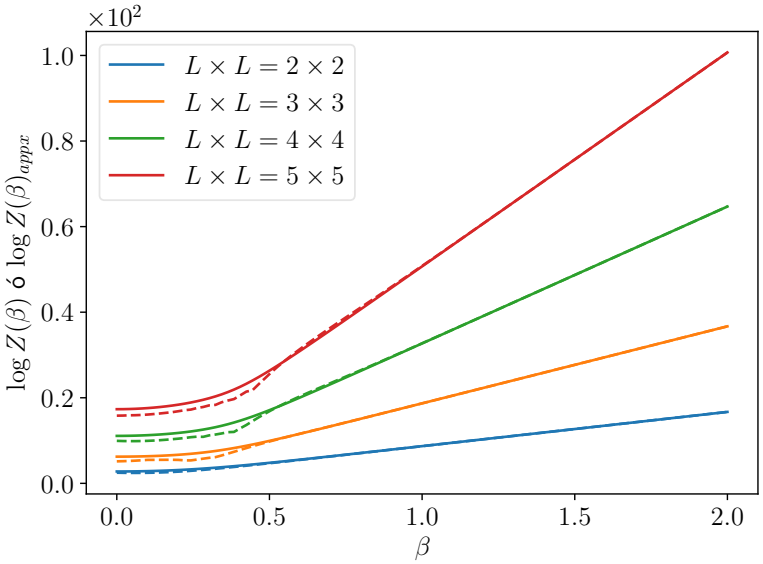

Figura 4. En líneas punteadas se muestra $\log Z_{\text{appx}}$ calculado con el método descrito en el artículo y en líneas continuas $\log Z$ calculado directamente por enumeración exacta. Esto muestra la equivalencia a bajas temperaturas entre ensambles microcanónico y macrocanónico en el caso de bajo valor L, expresada por la ecuación (7).

D. Calor específico

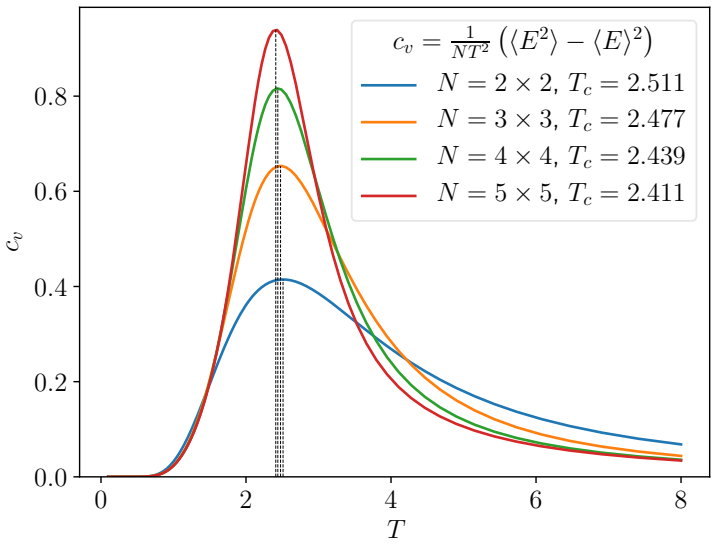

Figura 5. Calor específico c_v dado por (10) en función de la temperatura para diferentes valores de L. Los picos corresponden con la 'temperatura crítica', que para L grande se convierten en puntos discontinuos, demostrando así una transición de fase de segundo orden en el sistema.

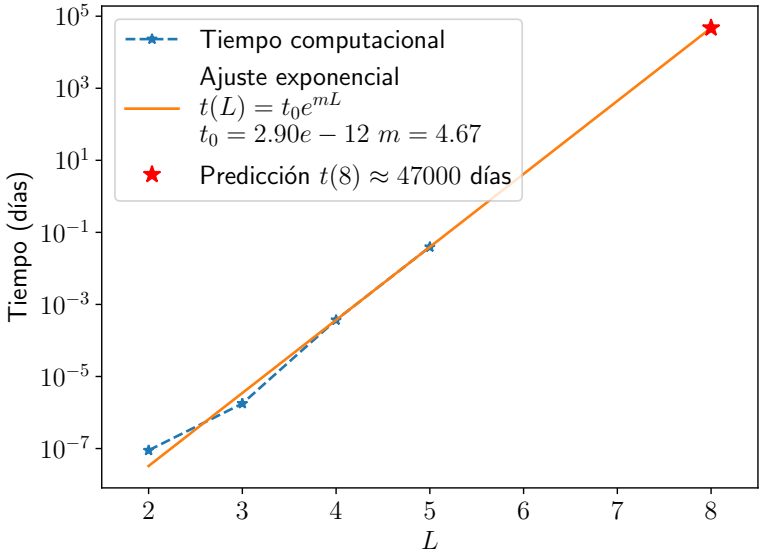

Figura 6. Tiempo de cómputo para las gráficas de calor específico y ajuste exponencial para predecir tiempo de cómputo para el sistema $N = 8\times 8$. Suponiendo que se tiene la memoria RAM necesaria, el proceso de obtención de la gráfica $c_v(T)$ para dicho sistema sería aproximadamente 47 mil días o 128 años.

Con las probabilidades para las energías del sistema dadas por (5) es fácil calcular el valor medio de E y el valor

medio de E^2

$$\langle E \rangle = \sum_E E p_E = \sum_E E \frac{\Omega(E) e^{-\beta E}}{Z} \tag{8}$$

$$\langle E^2 \rangle = \sum_E E^2 p_E = \sum_E E^2 \frac{\Omega(E) e^{-\beta E}}{Z}. \tag{9}$$

Con esto en mente es fácil calcular el calor específico, que en el ensamble canónico está dado por

$$c_v = \frac{\beta^2}{N} \left(\langle E^2 \rangle - \langle E \rangle^2 \right). \tag{10}$$

De los cálculos en las secciones anteriores tenemos todos los elementos para calcular este calor específico para $L = 2,\ 3,\ 4,\ 5$. La figura 5 muestra dichos valores en función de la temperatura, T. Como nos encontramos aún en valores relativamente bajos de L, no se presenta una discontinuidad evidente en el calor específico. Sin embargo, conforme se aumenta L, notamos que el pico de c_v se va haciendo más puntiagudo. Para valores más altos de L, como se verá en la sección II, el pico se convierte en el punto de discontinuidad, demostrando así una transición de fase de segundo orden, para la cual, en el límite $L \to \infty$, la temperatura crítica es $T_c = 2/\log\left(1+\sqrt{2}\right)$. En las gráficas se muestran los valores T_c para cada L, asociados a los picos de las gráficas.

En la última figura de esta sección, 6, presentamos un ajuste exponencial para los tiempos de cómputo de las gráficas $c_v(L)$ para cada L. El ajuste predice un tiempo de cómputo de 47000 días para el sistema 8×8. Aquí notamos la inminencia de la complejidad computacional de este problema, que es de tipo exponencial y la incapacidad de tratarlo de forma directa 'contando microestados' como lo hemos hestado haciendo hasta ahora. En la siguiente sección presentaremos el algoritmo metrópolis que nos ayudará a aproximarnos a las soluciones de este problema para sistemas más grandes.

II. MODELO DE ISING Y ALGORITMO METRÓPOLIS

En trabajos anteriores ya hemos estudiado el algoritmo Metrópolis, el cual consiste en un muestreo inteligente de distribuciones de probabilidad y que es muy útil para sistemas que presentan un alto número de configuraciones posibles. A continuación explicaremos la implementación de este algoritmo en el caso del modelo de Ising.

A. El algoritmo Metrópolis para el modelo de Ising 2d

En esta sección mostramos el retazo de programa más importante que usamos en nuestra implamentación y que corresponde con el algoritmo Metrópolis para el modelo de Ising.

En la línea 3 importamos el módulo `ising2d_metropolis` que escribimos y contiene todas las funciones necesarias para generar las gráficas que se muestran en esta y las siguientes secciones (las gráficas se generaron con el script `ising2d_metropolis_run.py` que se puede encontrar en el anexo B)

La línea 11 calcula los vecinos de cada uno de los sitios que se necesitarán para calcular las energías de cada configuración. El método del cálculo de los vecinos se explica directamente en los comentarios del módulo `ising2d_metropolis` y consiste en tener en cuenta las condiciones de frontera periódicas de manera compacta usando operadores división suelo (*floor division*) y módulo.

Las líneas $14-17$ deciden si se leyó un microestado inicial para comenzar el algoritmo o si se generará uno de forma aleatoria.

La línea 20 calcula la energía del microestado inicial.

Las líneas $26-32$ corren el algoritmo Metrópolis para un número de pasos transiente `N_transient`, con el fin de que el sistema se termalice. En esta parte NO se guardan los datos de las energías de cada paso, precisamente porque es una etapa de transiente y no muestra las características del estado de equilibrio, en el cual estamos interesados. La probabilidad de aceptación es dada por la razón de los factores de boltzmann del microestado propuesto sobre el microestado anterior, ya que estamos trabajando en el ensamble canónico.

Las líneas $34-41$ corren el algoritmo Metrópolis para un número de pasos `N_steps`, en el cual se espera que el sistema ya esté termalizado, es decir, que se encuentre en un estado de equilibrio del ensamble canónico. En esta parte se guardan las energías de cada configuración por la que pasa el sistema. El resultado final del algoritmo es una lista con las energías por las cuales pasó el sistema durante el número de pasos `N_steps`, éstas constituyen un muestreo inteligente de la distribución de probabilidad p_E dada por (5) y por tanto nos permiten calcular las propiedades termodinámicas del sistema.

El resto del módulo `ising2d_metropolis` está bien explicado en los comentarios del mismo.

Hay que anotar enfáticamente que en las implementaciones del algoritmo metrópolis se usó la librería Numba que permite un tipo de compilación de los programas denominado JIT (*Just In Time Compilation*). Sin el uso de esta librería, no hubiese sido posible el cálculo tan preciso de las datos usados para generar las gráficas acá presentadas, ya que los tiempos de cómputo al usar este tipo de compilación permiten ganancias en la optimización de los tiempos de cómputo hasta en 100 veces [2].

```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. Importancia de la termalización del sistema al usar el algoritmo Metrópolis

Como se dijo en la sección anterior, se espera que en la primera parte del algoritmo (en los primeros `N_transient` pasos) el sistema se termalice. Como estamos en el ensamble canónico, la termalización del sistema está caracterizada por la energía promedio del sistema $\langle E\rangle/N$. Cuando esta cantidad se mantenga aproximadamente constante con el cambio del número de iteraciones, podemos asegurar que el sistema se habrá termalizado.

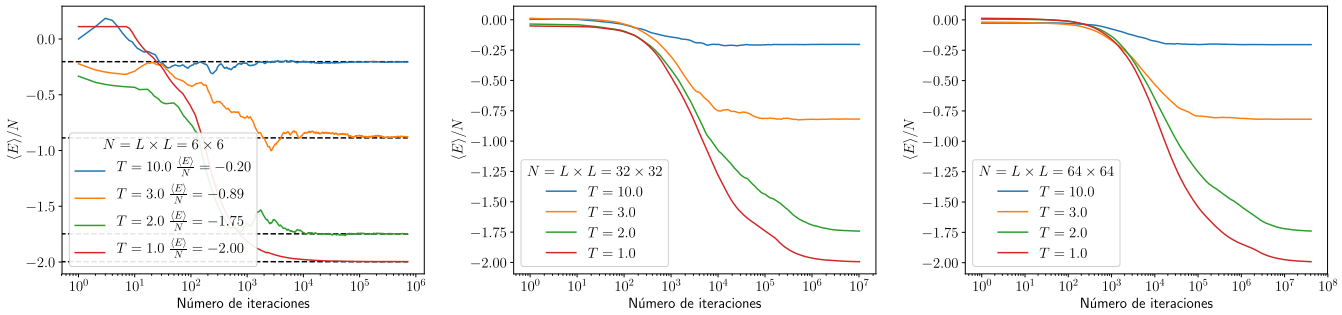

Figura 7. Visualización gráfica del paso al equilibrio en el algoritmo Metrópolis para el modelo de Ising en el ensamble canónico, medido por el paso al equilibrio de la energía promedio del sistema. Izquierda: $N = 6 \times 6$. Centro: $N = 32 \times 32$. Derecha: $N = 64 \times 64$.

Para entender mejor lo anterior, en la figura 7 se muestra $\langle E\rangle/N$ en función del número de iteraciones `N_steps`. Como en este caso estamos interesados en ver direectamente la termalización, se escoge `N_transient` igual a cero. En la izquierda se muestra la termalización para el caso $N = 6 \times 6$, en el cual se puede hacer la comparación de $\langle E\rangle/N$ para diferentes teperaturas por enumeración exacta, ya que en el enunciado de la parte 1 se dio el histograma $\Omega(E)$ para este caso y eso basta para calcular $\langle E\rangle/N$. En dicha figura se muestran los valores para cada temperatura y notamos que la termalización se da en escalas diferentes, dependiendo de la temperatura. Así, para temperaturas altas, el sistema se termaliza mucho más rápido que para temperaturas bajas, pero esto es más evidente en sistemas con L más grande. Es interesante notar que los límites están bien definidos. Para temperaturas bajas, el sistema tiende a estados ferromagnéticos que implican $\langle E\rangle/N \approx -2$, mientras que para temperaturas altas el sistema se termaliza en $\langle E\rangle/N$ cada vez más cercano a cero, como se espera, según los histogramas de $\Omega(E)$ y $e^{-\beta E}\Omega(E)$ mostrados en la sección I B.

En el centro de la figura 7 encontramos la termalización para $L = 32$ y diferentes temperaturas. Notamos que en general la termalización demora más que para el caso anterior ($N = 6 \times 6$). Esto es esperable, ya que el número de configuraciones posibles en este caso ($\Omega = 2^{32\times32}$!!!) es muchísimo mayor que el número de configuraciones en el caso anterior, por lo cual, el sistema debe pasar primero por muchos más estados antes de llegar al equilibrio. Sucede de forma similar con el caso $N = 64 \times 64$, el cual necesita muchas más iteraciones para llegar al equilibrio.

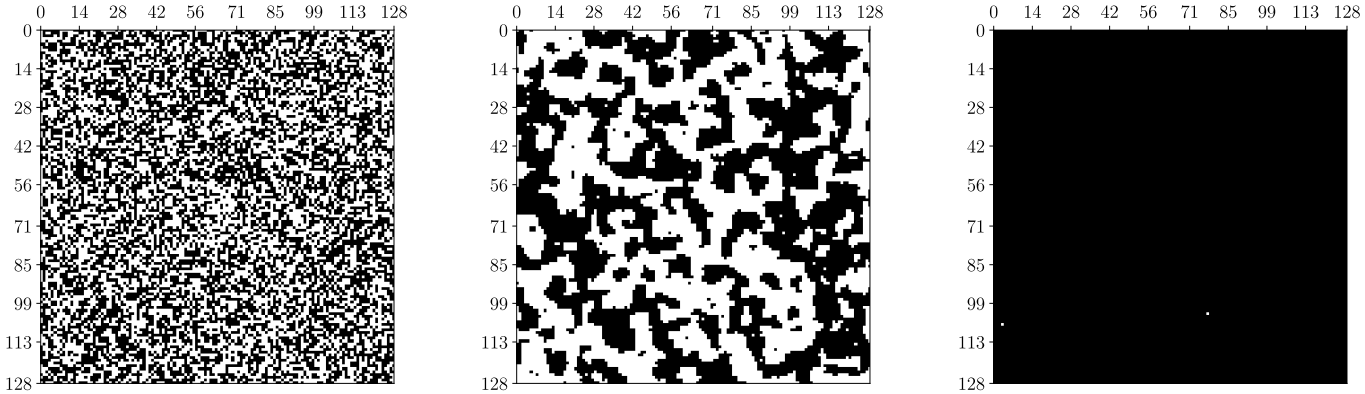

Figura 8. Microestados por los cuales pasa el sistema de $N = 128 \times 128$ en el principio del transiente ($N_{steps} = 10^3$, izquierda), en medio del transiente ($N_{steps} = 10^5$, centro) y cuando el sistema ya está termalizado ($N_{steps} = 10^8$, derecha).

Para ilustrar un poco mejor el tema de la termalización, en la figura 8 mostramos los microestados por los cuales pasa el sistema de $N = 128 \times 128$ en el principio del transiente ($\mathrm{N_{steps}} = 10^3$), en medio del transiente ($\mathrm{N_{steps}} = 10^5$) y cuando el sistema ya está termalizado ($\mathrm{N_{steps}} = 10^8$), todo para temperatura baja $T = 1$ y, como queremos ver explícitamente la termalización, se usó $\mathrm{N_{transient}} = 0$. Vemos que el sistema pasa de una configuración prácticamente aleatoria ($\langle E \rangle / N \approx 0$), luego una configuración con pequeños parches con dominio ferromagnético ($\langle E \rangle / N \approx -1$) y finalmente a una configuración casi completamente ferromagnética donde ($\langle E \rangle / N \approx -2$), estos valores se deducen aproximadamente a partir de la figura 7 (derecha) que muestra la termalización para el caso más cercano al $N = 128 \times 128$ que se pudo obtener.

C. Microestados finales y temperatura crítica

Una vez el sistema está termalizado, podemos obtener microestados típicos del modelo de Ising en el ensamble canónico a una temperatura fija. En la figura 9 encontramos los microestados mencionados del sistema $L = 64$ para temperaturas $T = 1,\ 2.5$ y 10 –izquierda, centro y derecha, respectivamente.

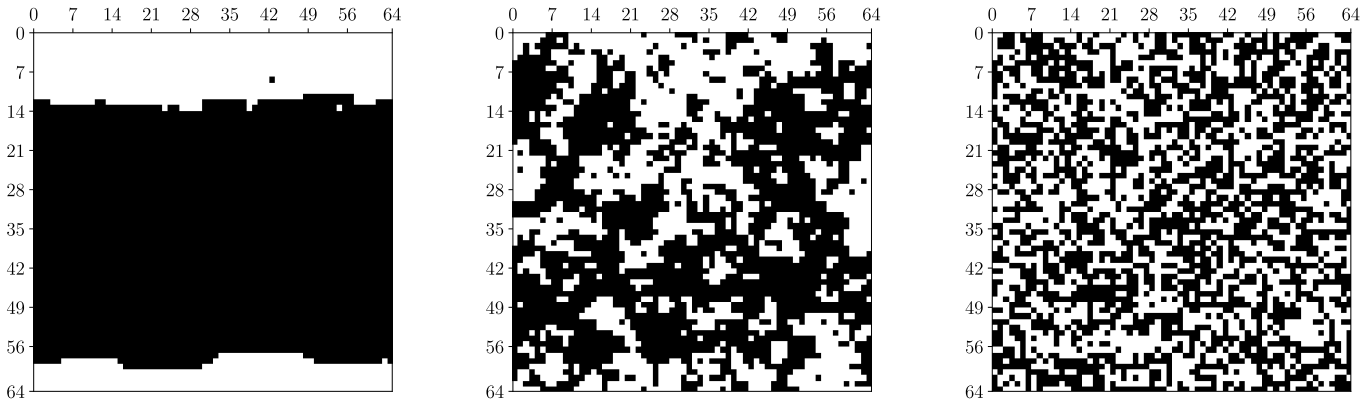

Figura 9. Microestados típicos del sistema $L = 64$ para temperaturas $T = 1,\ 2.5$ y 10 –izquierda, centro y derecha, respectivamente.

Para $T = 1$ encontramos un sistema con dos grandes dominios ferromagnéticos, esta configuración es estable a bajas temperaturas para sistemas con L grande, ya que el aumento en la energía debido a los bordes entre los dos diferentes dominios ferromagnéticos aumentan poco la energía del sistema *i.e* $\delta E > 0$ de los bordes es mucho menor a la energía de un sistema con un solo dominio ferromagnético $E_{\min} = -2L^2 < 0$. Cuanto más baja sea la temperatura, este tipo de microestados con dos grandes dominios ferromagnéticos serán menos probables y en cambio serán mucho más probables microestados con un solo dominio ferromagnético (todos los espines 1 o todos -1).

El caso $T = 2.5$ es más cercano a la temperatura crítica del sistema con L infinito. Aquí observamos que hay pequeños pero muchos más dominios ferromagnéticos (pequeños parches blancos o negros), que corresponden con un sistema típico de temperatura que no es muy alta ni muy baja.

Finalmente, el caso de alta temperatura, $T = 10$ es casi completamente aleatorio, aunque aún se notan algunos dominios ferromagnéticos son casi imperceptibles. Este es un microestado típico de alta temperatura, donde todos los microestados son casi igualmente probables y en el que predominan energías promedio $\langle E \rangle / N$ cercanas a cero, como se puede ver en la figura 7 (derecha).

Un caso interesante de estudio son las configuraciones típicas de un sistema relativamente grande a temperatura cercana a la temperatura crítica. Para entender qué sucede en este caso, en la figura 10 se muestran tres configuraciones típicas para el sistema $L = 128$ a tempreatura $T_c \approx 2.27$. Encontramos que a esta temperatura hay grandes dominios ferromagnéticos que ya están prácticamente formados pero contienen "ruido" en su interior, es decir, grandes regiones blancas o negras que en su interior contienen esporádicamente el color (espín) contrario. Esta es la manifestación del cambio de fase de segundo orden, de la cual ya hablamos en la sección I D. Para temperaturas menores a la temperatura crítica esperamos obtener menos ruido en esos dominios ferromagnéticos bien formados, paulatinamente hasta llegar a un estado totalmente ferromagnético en el límite $T \to 0$. Para temperaturas más altas, esperamos tener estados un poco más aleatorios con dominios ferromagnéticos cada vez menos evidentes.

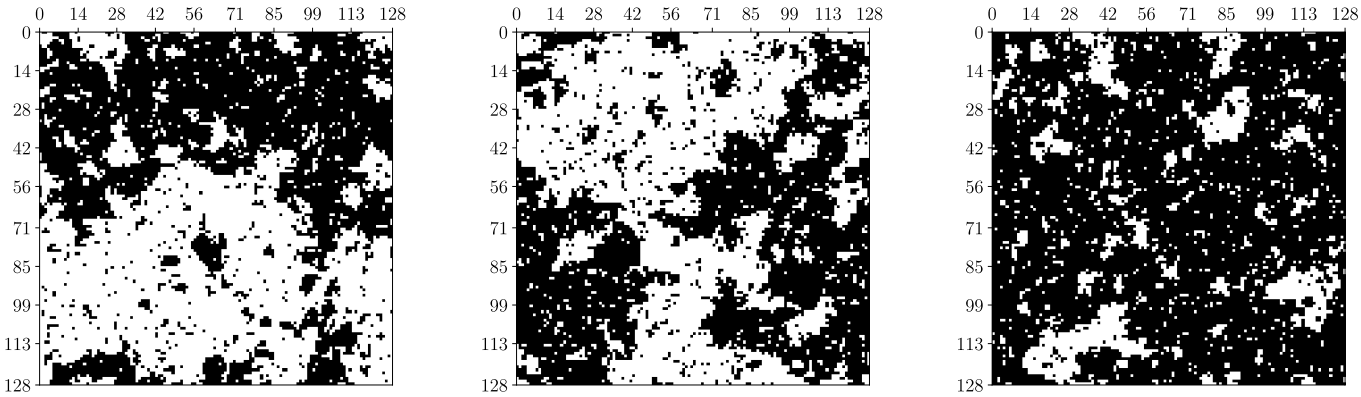

Figura 10. Tres configuraciones típicas para el sistema $L = 128$ a tempreatura $T_c \approx 2.27$.

D. Calor específico

Para finalizar nuestros análisis, presentamos en la figura 11 el calor específico calculado para varios valores de L, hasta $L = 64$. En primer lugar hay que mencionar que obtener esta gráfica no sería posible de manera sencilla por enumeración exacta ya que el número de microestados $\Omega = 2^{64\times64}$ es un número casi inconmesurable que sobrepasa cualquier capacidad computacional. En cambio, mediante un muestreo inteligente con el algoritmo metrópolis es posible obtenerla. En los casos $L = 2,\ 3,\ 4$ y 5 se muestra para comparación y en línea negra punteada los resultados obtenidos por enumeración exacta. Notamos que el algoritmo metrópolis logra capturar estas curvas con gran precisión.

Para valores más grandes de L no es posible obtener la curva 'teórica' ya que no se tiene acceso a la ennumeración exacta de los microestados. Sin embargo, notamos que las curvas para estos valores son continuas y presentan pocas fluctuaciones, lo cual da un buen presagio para la exactitud del método.

Encontramos también evidente que el pico de las gráficas se hace cada vez más puntiagudo, conforme aumenta L, lo cual evidencia una transición de fase de segundo orden, según se comentó ya en la sección I D.

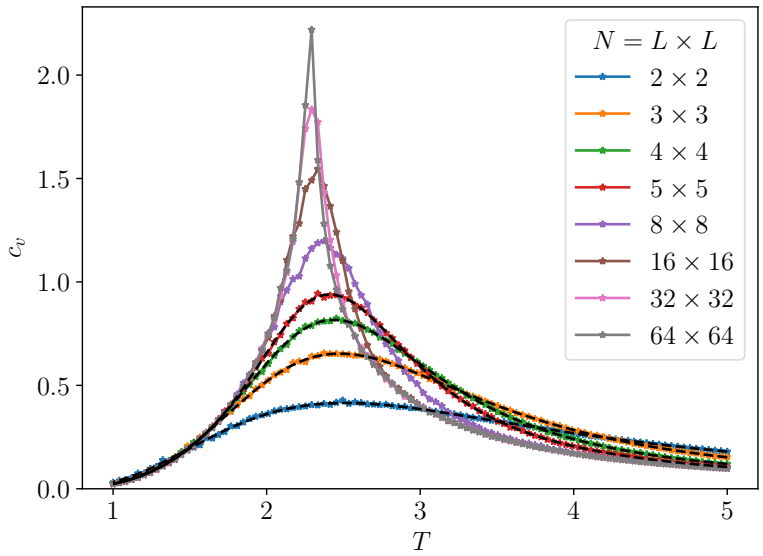

Figura 11. Calor específico en función de la temperatura para varios tamaños L del sistema calculados con el algoritmo metrópolis. Se nota que cuanto mayor es L, la discontinuidad en la curva $c_v(T)$ comienza a formarse, característica propia de una transición de fase de segundo orden. Los parámetros usados fueron $\mathrm{N_{steps}} = 80000L \times L$, $\mathrm{N_{transient}} = 0.7\mathrm{N_{steps}}$ y se graficaron 100 datos de c_v para valores entre $T = 1$ y $T = 5$. El tiempo de cómputo fue de 12000s.

III. CONCLUSIÓN

El modelo de Ising es un sistema con una gran riqueza conceptual, debido a la sencillez de su planteamiento y a la gran cantidad de información que podemos obtener de él. Esto lo pudimos observar alrededor de todo el artículo.

La sencillez de su planteamiento no se traduce en sencillez de cómputo, pues como se vio en la primera sección, éste es un sistema en el cual el número de microestados crece muy rápidamente conforme L crece. Sin embargo, para valores pequeños de L fue posible evidenciar características como la asimetría en los histogramas de $\Omega(E)$ para L impar, que lo asociamos a un efecto de tamaño finito debido a las condiciones de frontera periódicas usadas. Pudimos también evidenciar en esta parte que las contribuciones a la función partición son mayores para estados menos energéticos conforme disminuimos la temperatura del sistema, tal y como se espera. Además, encontramos que para temperaturas bajas, a pesar de que L fuese bajo, hay una equivalencia entre los ensambles micro y macrocanónico. En lo que a esto concierne se podría investigar a futuro, con el método Metrópolis si para sistemas con L grande esta equivalencia la podemos evidenciar para cualquier temperatura. Finalmente, para estos sistemas de L pequeño pudimos calcular por enumeración exacta y usando el teorema de fluctuación-disipación las curvas $c_v(T)$ en las que se comienza a dar indicio de la transición de fase de segundo orden, que es mucho más evidente para sistemas con L grande, tal y como se mostró en la sección del algoritmo metrópolis. Evidenciamos también en esta parte que el tiempo computacional para sistemas levemente mayores $L = 8$ el tiempo computacional desborda las capacidades de un computador 'sencillo'.

Dadas las limitaciones del método de enumeración exacta, nos vimos en la necesidad de acudir a una técnica de muestreo inteligente para poder obtener información de sistemas más grandes (hasta $N = 128 \times 128$): el algoritmo Metrópolis nos permitió reproducir gran cantidad de la información obtenida para sistemas pequeños pero para sistemas mucho más grandes. Encontramos que uno de los aspectos más delicados es lograr termalizar el sistema, de manera que los datos obtenidos mediante el algoritmo sean fidedignos. Pudios evidenciar en esta parte configuraciones típicas de microestados a diferentes temperaturas y para diferentes tamaños, encontrando que para temperaturas bajas los sistemas tienden a dominios casi completamente ferromagnéticos, en la temperatura crítica existen pocos y grandes dominios ferromagnéticos con 'ruido' que son característicos de la transición de fase y en temperaturas altas, los microestados tienden a ser muy aleatorios. Finalmente pudimos evidenciar mucho mejor la transición de fase de segundo orden como una (casi) discontinuidad en la curva c_v para sistemas con L grande (en nuestro caso logramos llegar hasta $L = 64$ y fue suficiente para notar el inicio de la formación de unadiscontinuidad, recordemos que la discontinuidad como tal se nota para $L \to \infty$).

Es de recalcar que para la parte del algoritmo Metrópolis, el cálculo computacional necesario para obtener las gráficas no hubiese sido posible sin el uso de la librería Numba, como se mencionó en la sección II A.

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] K. Huang, *Statistical Mechanics* (John Wiley & Sons, 1963).
[2] 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).
[3] W. Greiner, L. Neise, and H. Stöcker, *Thermodynamics and statistical mechanics* (Springer Science & Business Media, 2012).
[4] B. M. McCoyand T. T. Wu, *The two-dimensional Ising model* (Courier Corporation, 2014).
[5] P. M. Chaikin, T. C. Lubensky, and T. A. Witten, *Principles of condensed matter physics*, Vol. 10 (Cambridge university press Cambridge, 1995).
[6] J. Ortínand J. M. Sancho, *Curso de Física Estadística*, Vol. 28 (Edicions Universitat Barcelona, 2006).

Apéndice A: Código 1: enumeración exacta de microestados

A continuación se muestra el código usado para generar las gráficas de la sección I. Éste código usa el módulo `ising2d_microstates` en el que están todas las funciones que realizan los algoritmos. Éste último código está disponible en este link.

```
from ising2d_microstates import *


#############################################################################################################
# PANEL DE CONTROL
#############################################################################################################

# Decide si corre algoritmo para calcular microestados de energía
run_microstates_algorithm = False

# Decide si corre algoritmo para cálculo de contribuciones a la función partición
# por cada valor de energía
run_Z_contributions_algorithm = True

# Decide si corre algoritmo de aproximación de función partición
run_Z_approx_algorithm = False

# Decide si corre algoritmo para optimización de dx y beta_ini
run_specific_heat_algorithm = False

# Decide si corre demostración de asimetría para L impares
run_odd_asymmetry = False



#############################################################################################################
# PARÁMETROS GENERALES PARA LAS FIGURAS
#############################################################################################################

# Usar latex en texto de figuras y agrandar tamaño de fuente
plt.rc('text', usetex=True)
plt.rcParams.update({'font.size':15,'text.latex.unicode':True})

# Obtenemos path para guardar archivos en el mismo directorio donde se ubica el script
script_dir = os.path.dirname(os.path.abspath(__file__))


# Algoritmo para calcular microestados
if run_microstates_algorithm:
    # Tamaño del sistema
    L = 2
    # Decide si pone condiciones de frontera libres
    free_boundary_conditions = False
    energy_plot_kwargs = {
                          'microstate_energies': None,
                          'L': L,
                          'read_data': True,
                          'energy_data_file_name': None,
                          'interpolate_energies': False,
                          'show_plot': True,
                          'save_plot': False,
                          'plot_file_Name': None,
```

```
                        }

    print('---------------------------------------')
    print('---------------------------------------')
    print('Microstates algorithm')
    print('---------------------------------------\n')
    print('-----------------------------------------------')
    print('Grid: L x L = %d x %d'%(L, L))
    print('-----------------------------------------------')

    # Calcula los microestados del sistema solo si read_data=False.
    if not energy_plot_kwargs['read_data']:

        # Genera todos los microestados posibles
        microstates = ising_microstates(L)

        # Calcula los vecinos
        neighbours = ising_neighbours_free(L) if free_boundary_conditions \
                                              else ising_neighbours(L)

        # Cálculo de energía para cada microestado
        t_0 = time()
        energies = ising_energy(microstates, neighbours,
                                save_data = not free_boundary_conditions)
        t_1 = time()
        comp_time = t_1-t_0
        # Imprime log del algoritmo
        print('------------------------------------------------------------\n'
            + 'Explicit energies:  L = %d --> computation time = %.3f \n'%(L,comp_time)
            + '------------------------------------------------------------\n')

        energy_plot_kwargs['microstate_energies'] = energies
        print('---------------------------------------')
        print('All microstates, each in a single row:')
        print('---------------------------------------')
        print(pd.concat([pd.DataFrame(microstates),
                         pd.DataFrame({'Energy': energies})],
                         axis=1,
                         sort=False
                         ),
              '\n')

    # Grafica histograma de energías \Omega(E)
    ising_energy_plot(**energy_plot_kwargs)

    microstate_rand = np.random.choice([-1,1], L*L)
    print('-----------------------------------')
    print('One random microstate as a 2D grid:')
    print('-----------------------------------')
    print(pd.DataFrame(microstate_rand.reshape((L,L))), '\n')

    # Grafica un microestado aleatorio
    ising_microstate_plot(microstate_rand, save_plot=True)


# Algoritmo para calcular contribuciones a la función partición:
# Omega(E)*e^{-beta E}, que es proporcional a p_E
if run_Z_contributions_algorithm:
```

```
111     print('---------------------------------------')
112     print('---------------------------------------')
113     print('Z contributions algorithm')
114     print('---------------------------------------')
115     kwargs = {
116         'microstate_energies': None,
117         'L': 5,
118         'beta': 1.,
119         'beta_max': None,
120         'N_beta': 100,
121         'read_data': True,
122         'energy_data_file_name': None,
123         'plot_histogram': True,
124         'show_plot': True,
125         'save_plot': True,
126         'plot_file_Name': None,
127         }
128 
129     Z_array, statistical_weights_array, beta_array, energies, omegas = \
130         partition_func_stat_weights(**kwargs)
131 
132 # Algoritmo de aproximación de la función partición: equivalencia con ensamble
133 # microcanónico
134 if run_Z_approx_algorithm:
135     print('---------------------------------------')
136     print('---------------------------------------')
137     print('Z approximation algorithm')
138     print('---------------------------------------')
139     approx_partition_func(read_data=True, save_plot=True)
140 
141 # Algoritmo para graficar calor específico
142 if run_specific_heat_algorithm:
143     print('---------------------------------------')
144     print('---------------------------------------')
145     print('Specific Heat algorithm')
146     print('---------------------------------------\n')
147     kwargs = {
148         'microstate_energies_array': [None, None, None, None],
149         'L_array': [2, 3, 4, 5],
150         'beta_min': 1/5,
151         'beta_max': 1.,
152         'N_beta': 1000,
153         'read_data': True,
154         'energy_data_file_name': None,
155         'show_plot': True,
156         'save_plot': True,
157         'plot_file_Name': None,
158         'save_cv_data': True,
159         }
160 
161     plot_specific_heat_cv(**kwargs)
162 
163 # Algoritmo para mostrar que la asimetría en histograma de Omega para L impar
164 # se debe a las condiciones de frontera periódicas
165 if run_odd_asymmetry:
166     print('---------------------------------------')
167     print('---------------------------------------')
168     print('L odd energy asymmetry demonstration')
```

```
    print('-------------------------------------\n')
    L = 3
    ising_odd_L_energy_asymmetry(L, save_plot=True)
```

Apéndice B: Código 2: algoritmo Montecarlo

A continuación se muestra el código usado para generar las gráficas de la sección II. Éste código usa el módulo `ising2d_metropolis` en el que están todas las funciones que realizan los algoritmos. Éste último código está disponible en este link.

```
from ising2d_metropolis import *

###############################################################################################
# PANEL DE CONTROL
###############################################################################################

# Decide si corre algoritmo para calcular microestados de energía
run_metropolis_energies_algorithm = False

# Decide si corre algoritmo que muestra la termalización
run_thermalization_algorithm = False

# Decide si corre algoritmo de calor específico
run_specific_heat_algorithm = True



###############################################################################################
# PARÁMETROS GENERALES PARA LAS FIGURAS
###############################################################################################

# Usar latex en texto de figuras y agrandar tamaño de fuente
plt.rc('text', usetex=True)
plt.rcParams.update({'font.size':15,'text.latex.unicode':True})



# Muestreo de energías usando algoritmo Metrópolis
if run_metropolis_energies_algorithm:

    # Decide si lee o guarda datos y asigna nombres a los archivos
    read_ini_microstate_data = False
    save_final_microstate_data = True
    microstate_data_file_name = None
    save_energy_data = False
    energy_data_file_name = None

    # Muestra parámetros y tiempo de cómputo
    print_log = True

    # Decide si grafica microestado final
    plot_microstate = True
    # Parámetros para figura de microestado
    show_microstate_plot = True
    save_microstate_plot = True
    microstate_plot_file_name = None

```

```
    # Parámeros del algoritmo metrópolis para calcular energías
    T = 2.27
    beta = 1/T
    L = 128
    # Como se está usando numba, en microstate siempre hay que
    # entregar el siguiente array con dtype=np.int64
    microstate = np.ones(L * L, dtype=np.int64)
    J = 1
    N_steps = int(1e8)  # int(L * L * 10000)
    N_transient = 0     # int( N_steps)

    # Asigna nombre a archivo con datos de microestado inicial/final
    if read_ini_microstate_data or save_final_microstate_data:
        if not microstate_data_file_name:
            microstate_data_file_name = \
                ('ising-metropolis-final-config-L_%d-temp_%.3f'%(L, T)
                 + '-N_steps_%d-N_transient_%d.csv'%(N_steps, N_transient))
        microstate_data_file_name = script_dir + '/' + microstate_data_file_name
        if read_ini_microstate_data:
            microstate = pd.read_csv(microstate_data_file_name, index_col=0, comment='#')
            microstate = np.array(microstate.to_numpy().transpose()[0], dtype=np.int64)

    metropolis_args = (microstate, read_ini_microstate_data,
                       L, beta, J, N_steps, N_transient)


    # Decide si grafica histograma de energías
    # (contribuciones proporcionales al factor de boltzmann  Omega(E) * e**(-beta E) / Z(beta))
    plot_energy_hist = False
    # Parámetros para graficar histograma de energías.
    energy_hist_plot_file_name = \
        ('ising-metropolis-Z_contributions-plot-L_'
         + '%d-temp_%.3f-N_steps_%d-N_transient_%d.pdf'%(L, T, N_steps, N_transient))
    energy_plot_kwargs = {
                          'microstate_energies': None,
                          'L': L,
                          'read_data': False,
                          'energy_data_file_name': None,
                          'interpolate_energies': False,
                          'show_plot': True,
                          'save_plot': True,
                          'normed': True,
                          'plot_file_Name': energy_hist_plot_file_name,
                          'x_lim': None,
                          'y_label': '$\Omega(E) e^{-\\beta E}/Z(\\beta)$',
                          'legend_title': 'Metrópolis. $T=%.3f$'%T,
                          }

    # Corre algoritmo metrópolis con parámetros dados e imprime tiempo de cómputo
    t_0 = time()
    energies, microstate, avg_energy_per_spin = ising_metropolis_energies(*metropolis_args)
    t_1 = time()

    # Imprime información relevante
    if print_log:
        comp_time = t_1 - t_0
        print_params = (L, T, N_steps, N_transient)
        print('\n-------------------------------------------------------------------------------\n'
```

```
106             + 'Ising 2D Metropolis:\n'
107             + 'L = %d,  T = %.3f,  N_steps = %d,  N_transient = %d\n'%print_params
108             + '<E>/N = %.4f\n'%avg_energy_per_spin
109             + '--> computation time = %.3f \n'%comp_time
110             + '-------------------------------------------------------------------------\n')
111
112     # Guarda datos de energías muestreadas o microestado final en archivo CSV
113     if save_energy_data:
114         if not energy_data_file_name:
115             energy_data_file_name = ('ising-metropolis-energy-data-L_%d-temp_%.3f'%(L, T)
116                                      + '-N_steps_%d-N_transient_%d.csv'%(N_steps, N_transient))
117         energy_data_file_name = script_dir + '/' + energy_data_file_name
118         relevant_info = ['2D Ising ENERGIES, metropolis algorithm: L=%d   T=%.4f'%(L, T)
119                          + '   N_steps=%d   N_transient=%d'%(N_steps, N_transient)]
120         headers = ['energy']
121         save_csv(energies, data_headers=headers, file_name=energy_data_file_name,
122                  relevant_info=relevant_info, print_data=False)
123     if save_final_microstate_data:
124         relevant_info = ['2D Ising FINAL MICROSTATE, metropolis algorithm: L=%d  '%(L)
125                          + 'T=%.4f  N_steps=%d  N_transient=%d'%(T, N_steps, N_transient)]
126         headers = ['spin']
127         save_csv(microstate, data_headers=headers, file_name=microstate_data_file_name,
128                  relevant_info=relevant_info, print_data=False)
129
130     # Grafica microestado final.
131     if plot_microstate:
132         mstate_plot_args = (np.array(microstate), L, beta, J, N_steps, N_transient,
133                             show_microstate_plot, save_microstate_plot,
                                ↪  microstate_plot_file_name)
134         ising_metropolis_microstate_plot(*mstate_plot_args)
135
136     if plot_energy_hist:
137         energy_plot_kwargs['microstate_energies'] = np.array(energies)
138         ising_energy_plot(**energy_plot_kwargs)
139
140     del energies
141
142
143 # Algoritmo de termalización
144 if run_thermalization_algorithm:
145
146     # Decide si imprime info del algoritmo
147     print_log = True
148
149     # Parámetros de algoritmo de termalización
150     beta = np.array([1 /10., 1/3., 1/2., 1/1.])
151     L = 2
152     microstates_ini = np.ones( (len(beta), L * L), dtype=np.int64)
153     read_ini_microstate_data = False
154     J = 1
155     N_steps = int(L * L * 1e4)
156     N_transient = 0
157
158     thermalization_args = \
159         (microstates_ini, read_ini_microstate_data, L, beta, J, N_steps, N_transient)
160
161     # Corre algoritmo de termalización
162     t_0 = time()
```

```
    avg_energy_per_spin_array, beta, *dont_need = thermalization_demo(*thermalization_args)
    t_1 = time()

    if print_log:
        comp_time = t_1 - t_0
        print_params = (L, N_steps, N_transient)
        print('\n-------------------------------------------------------------------------------\n'
            + 'Ising 2D Metropolis thermalization:\n'
            + 'T = ' + str(list(1/np.array(beta))) + '\n'
            + 'L = %d,  N_steps = %d,  N_transient = %d\n'%print_params
            + '<E>/N = ' + str([E_over_N[-1] for E_over_N in avg_energy_per_spin_array]) + '\n'
            + '--> computation time = %.3f \n'%comp_time
            + '-------------------------------------------------------------------------------\n')

    # Parámetros de figura de termalización
    thermaization_data_file_name = None
    show_plot = True
    save_plot = True
    plot_file_Name = None

    thermalization_plot_args = (avg_energy_per_spin_array, beta, L, J, N_steps,
                                N_transient, thermaization_data_file_name, show_plot,
                                save_plot, plot_file_Name)

    plot_thermalization_demo(*thermalization_plot_args)

# Algoritmo de calor específico
if run_specific_heat_algorithm:

    # Si read_cv_data=True, el algoritmo no corre, sino que se leen los datos de un archivo.
    read_cv_data = True
    save_cv_data = True
    cv_data_file_name = None


    # Decide si imprime info del algoritmo
    print_log = True
    # Parámetros del algoritmo
    L_array = np.array([2, 3, 4, 5, 8, 16, 32, 64])
    N_steps_factor = int(8e4)
    N_transient_factor = 0.7
    J = 1
    T_min = 1.0
    T_max = 5.0
    N_temp = 100

    several_cv_args = (L_array, N_steps_factor, N_transient_factor,
                       J, T_min, T_max, N_temp)

    # Corre el algoritmo
    t_0 = time()
    if not read_cv_data:
        cv_arrays, T_arrays, L_array, N_steps_factor = \
                                      several_specific_heats(*several_cv_args)
    else:
        cv_arrays, T_arrays = None, None
    t_1 = time()

```

```
    # Imprime info del algoritmo
    if print_log or save_energy_data:
        comp_time = t_1 - t_0
        line0 = '------------------------------------------------------------------------------\n'
        line1 = 'Ising 2D Metropolis specific heat (cv) plot:'
        line2 = 'T_min = %.3f,  T_max = %.3f,  N_temp = %d'%(T_min, T_max, N_temp)
        line3 = 'L = ' + str(list(L_array))
        line4 = ('N_steps_factor = %d '%N_steps_factor
                 + '(N_steps = L*L*N_steps_factor, '
                 + 'N_transient = %.2f N_steps)'%N_transient_factor)
        line5 = '--> computation time = %.3f'%comp_time
        if print_log and not read_cv_data:
            print('\n' + line0 + line1 + '\n' + line2 + '\n' + line3 + '\n' + line4 + '\n'
                  + line5 + '\n' + line0)
        if print_log and read_cv_data:
            print('Los datos se leyeron de un archivo, no se generaron en este momento.')

    # Guarda datos de energías muestreadas o microestado final en archivo CSV
    if save_cv_data and not read_cv_data:
        if not cv_data_file_name:
            L_string = '_'.join([str(L) for L in L_array])
            cv_data_file_name = ('ising-metropolis-specific_heat-plot-L_' + L_string
                + '-N_steps_factor_%d-N_transient_factor_%d-T_min_%.3f-T_max_%.3f-N_temp_%d.csv'
                % (N_steps_factor, N_transient_factor, T_min, T_max, N_temp))
        cv_data_file_name = script_dir + '/' + cv_data_file_name
        relevant_info = [line1, line2 , line3, line4, line5]
        headers = np.array([ ['Temperature', 'cv (L=%d)'%L] for L in L_array]).flatten()
        shape = (2*len(L_array), len(cv_arrays[0]))
        cv_data = np.array([[T, cv_arrays[i]] for i, T in enumerate(T_arrays)]).reshape(shape)
        save_csv(cv_data.transpose(), data_headers=headers, file_name=cv_data_file_name,
                 relevant_info=relevant_info, print_data=False)
    if save_cv_data and read_cv_data:
        print('Se escogió leer los datos del calor específico de un archivo de texto.')

    # Parámetros de la gráfica
    show_plot = True
    save_plot = True
    read_cv_data_part_1 = True
    plot_file_Name = None

    cv_plot_args = (cv_arrays, T_arrays, L_array, N_steps_factor, N_transient_factor,
                    T_min, T_max, N_temp, J, read_cv_data_part_1, read_cv_data,
                    cv_data_file_name, show_plot, save_plot, plot_file_Name)

    specific_heat_plot(*cv_plot_args)

    pass
```