Tarea 2

Nicolás Cortés Meneses

25 de septiembre de 2024

Comentarios iniciales

En el repositorio de GitHub se encuentra el archivo `tarea_2.ipynb`, el cual puede ser utilizado para examinar los resultados, cambiar parámetros y jugar con las funciones escritas para cada problema. Para ello, cada problema de la tarea tiene su módulo asociado, por ejemplo, `P1_funciones.py` para el problema 1, el cual es necesario importar para ejecutar los cálculos y generar los gráficos correspondientes en cada problema. No es necesario importar todos los módulos juntos a la vez, ya que pueden existir funciones con el mismo nombre en diferentes módulos.

Para evitar conflictos, se recomienda encarecidamente importar cada módulo por separado conforme se vaya necesitando al ejecutar las funciones o celdas correspondiente a cada problema.

A excepción del problema 2, en los módulos asociados a cada problema se utilizó el módulo `joblib` para correr el código en múltiples núcleos de la CPU. Se debe ajustar el parámetro `n_jobs` presente en algunas funciones según la cantidad de núcleos que se deseen utilizar. En mi caso, se utilizó una laptop con 8 núcleos.

Problema 1

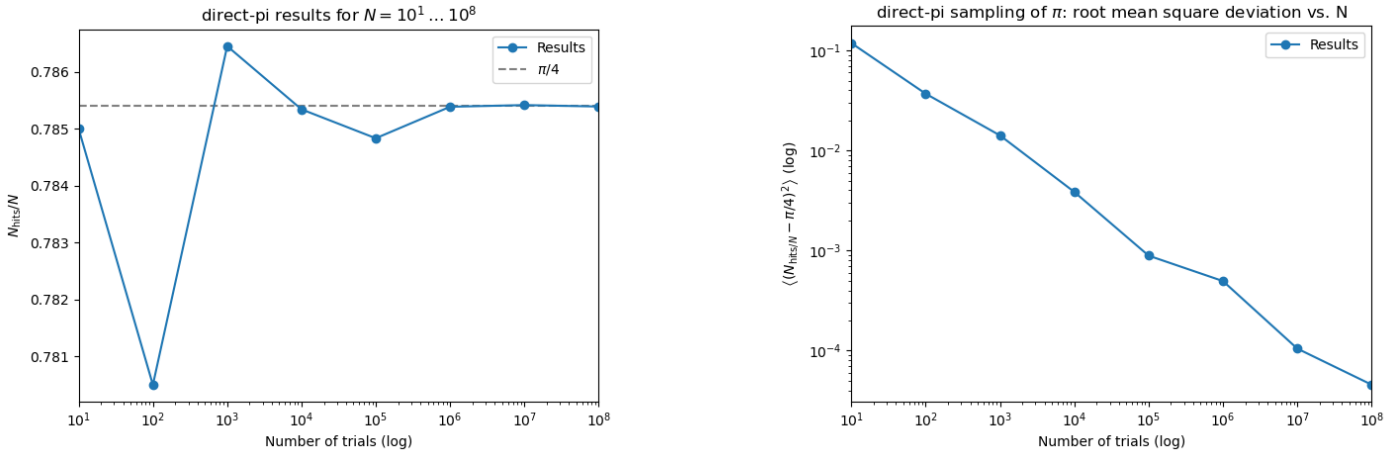
a)

Los resultados obtenidos utilizando `direct-pi` tomando $N = 10^1, \ldots, 10^8$ y 20 runs por cada valor de N y la desviación cuadrática media se encuentran resumidos en las figuras 1a y 1b. Como podemos notar en la figura 1a, a medida que N aumenta, el valor de $N_{\mathbf{hits}}/N$ se va estabilizando y converge a $\pi/4$. Por otra parte, desde la figura 1b es evidente que a medida que aumenta N, la desviación cuadrática media decae de manera cercana a $1/\sqrt{N}$.

b)

Para comenzar, los resultados de utilizar la función `markov_pi` para N_{hits}/N, tomando tomando $N = 10^1, \ldots, 10^8$, 50 runs por cada valor de N y $\delta = 0.3$ se encuentran en la figura 2.

Nuevamente, podemos notar que los valores de N_{hits}/N convergen a $\pi/4$. Por otra parte, podemos analizar la desviación cuadrática media y la taza de rechazo. Para ambos casos se utilizó $N = 2^4, \ldots, 2^{13}$ y $\delta = 0.125, 0.250, 0.500, 1.000, 2.000, 4.000$, los resultados de ambos análisis se encuentran en las figuras 3a y 3b.



(a) (b)

Figura 1: Resultados obtenidos utilizando `direct_pi`.

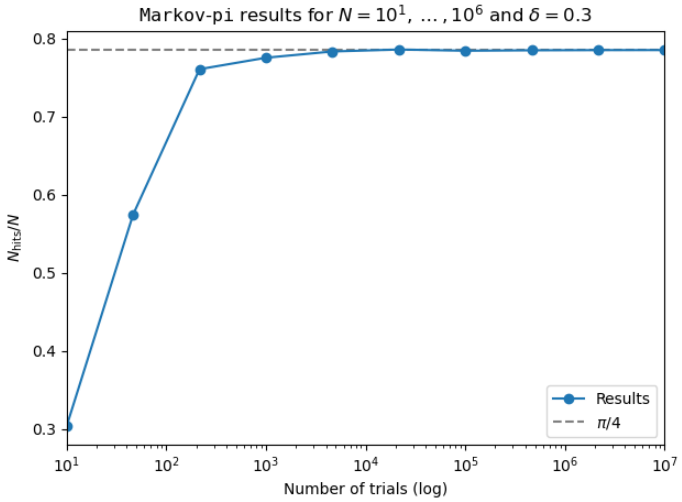


Figura 2: Caption

Como podemos notar en la fig. 3a, el valor de δ que logra menores valores para la desviación cuadrática media es $\delta = 1.0$, mientras que en la taza de rechazo es de los más bajos, junto con $\delta = 2.0$. Por lo tanto, el valor óptimo para realizar nuestras estimaciones utilizando `markov_pi` sería $\delta = 1.0$. Junto a esto, la línea de referencia en la fig. 3b corresponde a la fracción de área de un cuarto de círculo respecto a un cuadrado unitario

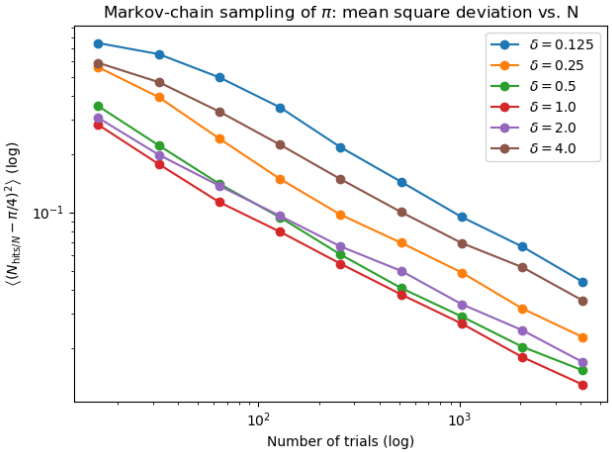
$$1 - \frac{A_{\text{círculo}/4}}{A_{\text{cuadrado}}} = 1 - \frac{\pi/4}{1} \approx 0.2146.$$

Por lo tanto, a medida que aumenta el N, es razonable esperar que se vaya cubriendo toda el área del cuadrado unitario y que la tasa de rechazo se estabilice en dicho valor.

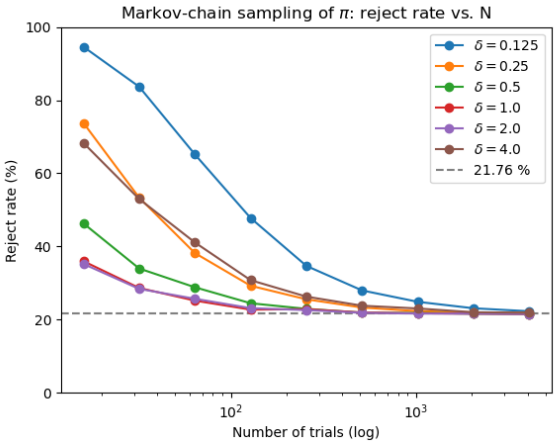
Problema 2

a)

En el módulo `P2_funciones.py` se encuentra la función `gray_show_binary` con la cual es posible obtener las posibles variaciones de una cadena de N spins, en el estado representado por N-0's. Para una red de



(a)



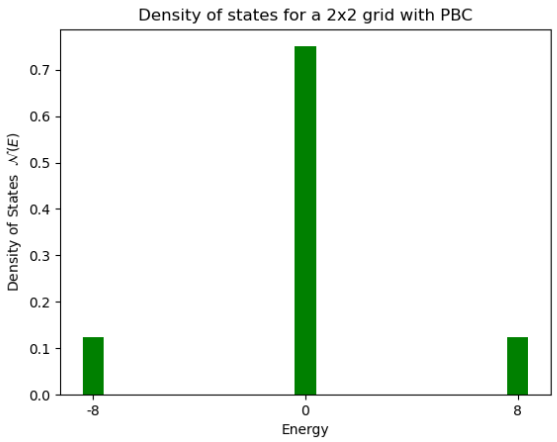
(b)

Figura 3: (a) Desviación cuadrática media y (b) tasa de rechazo para distintos valores de N utilizando la función `markov_pi`.

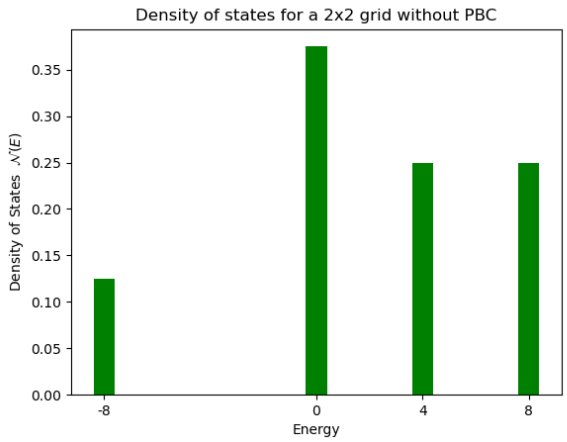
4 spins, el resultado entregado es:

```
[0, 0, 0, 0]
[0, 0, 0, 1]
[0, 0, 1, 1]
[0, 0, 1, 0]
[0, 1, 1, 0]
[0, 1, 1, 1]
[0, 1, 0, 1]
[0, 1, 0, 0]
[1, 1, 0, 0]
[1, 1, 0, 1]
[1, 1, 1, 1]
[1, 1, 1, 0]
[1, 0, 1, 0]
[1, 0, 1, 1]
[1, 0, 0, 1]
[1, 0, 0, 0]
```

Lo cual es análogo a la tabla presente en la slide 14, clase 5. Para cálculos posteriores, se hará uso de la función `gray_flip` dentro de los cálculos realizados por la función `enumerate_ising`. Posterior a esto, dadas las limitaciones del sistema, se realizó el calculo de las densidades de estados para redes cuadradas de dimensiones 2×2, 3×3, 4×4 y 5×5 utilizando condiciones de borde periódicas (PBC) y no-periódicas (non-PBC), los resultados se encuentran en las figuras 4, 5, 6 y 7, respectivamente.

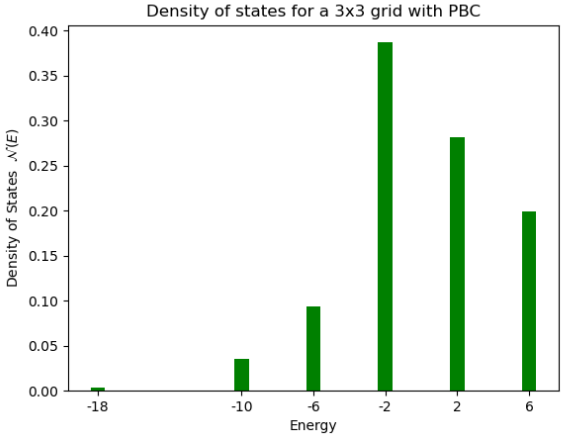


(a)

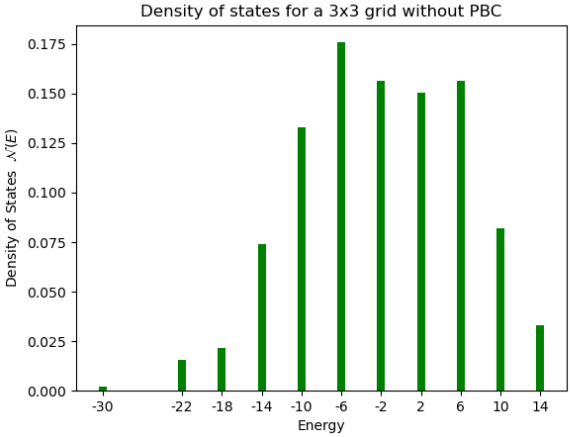


(b)

Figura 4: Resultados para densidad de estados (DoS) para una grilla de 2×2 (a) con PBC y (b) sin PBC.

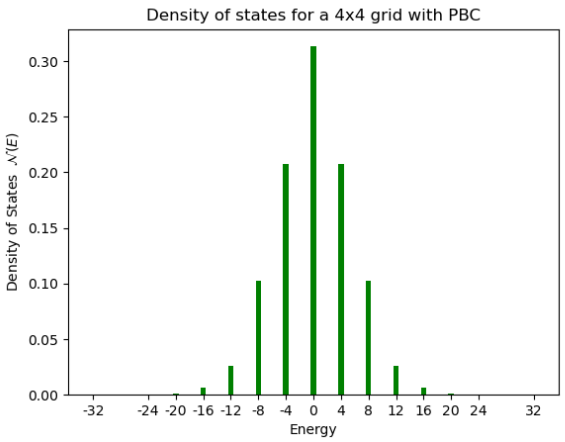


(a)

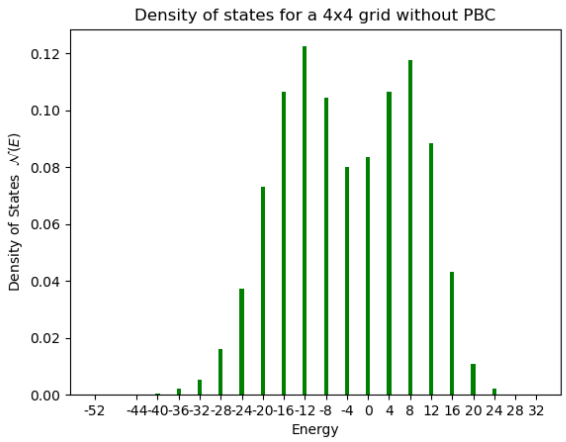


(b)

Figura 5: Resultados para densidad de estados (DoS) para una grilla de 3×3 (a) con PBC y (b) sin PBC.

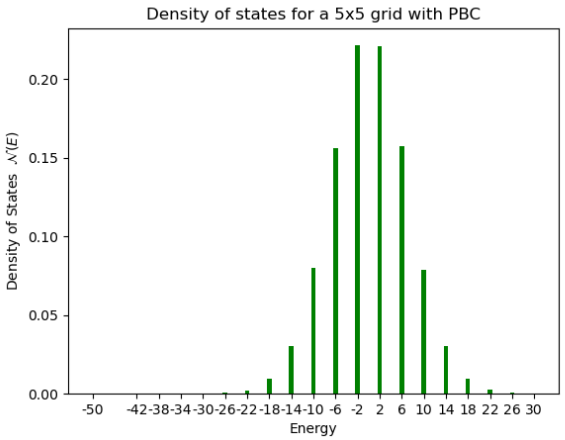


(a)

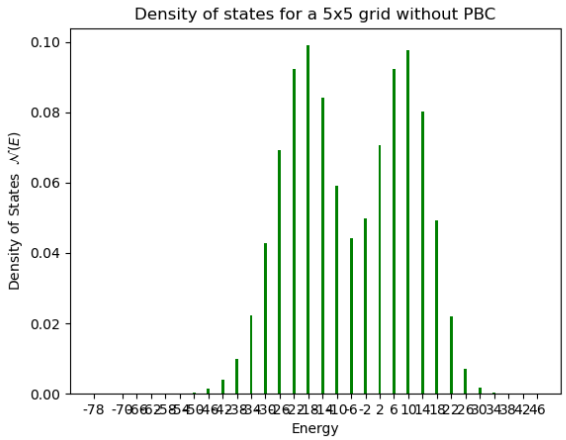


(b)

Figura 6: Resultados para densidad de estados (DoS) para una grilla de 4×4 (a) con PBC y (b) sin PBC.



(a)



(b)

Figura 7: Resultados para densidad de estados (DoS) para una grilla de 5×5 (a) con PBC y (b) sin PBC.

Para los casos sin PBC, podemos notar que a medida que aumenta el tamaño de la grilla, es posible ver que comienza a formarse una distribución bimodal. Para redes con L-par dicha distribución está centrada en $E = 0$. Mientras que el caso con PBC, podemos notar que la distribución es completamente unimodal, nuevamente para L-par se encuentra centrada en $E = 0$ y en este caso, es simétrica.

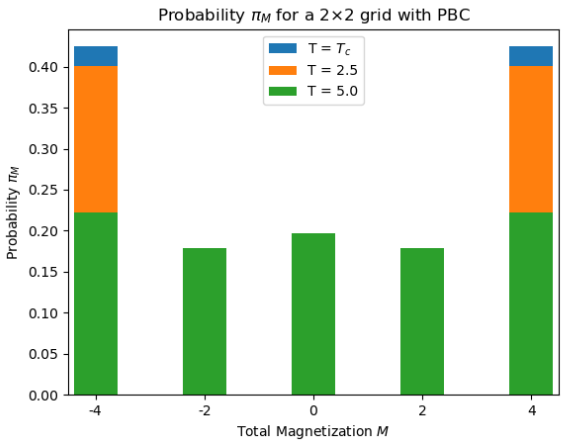
b)

Para esta sección se utilizó la función `enumerate_ising_magnetization` para generar diccionarios donde se registre la densidad de estados $\mathcal{N}(E, M)$. En primer lugar, para recuperar los resultados de la tabla de la slide 15, clase 5 se utilizó la función `check_e_sum` para sumar sobre las magnetizaciones M de una energía dada E. Los resultados obtenidos al correr el código fueron:

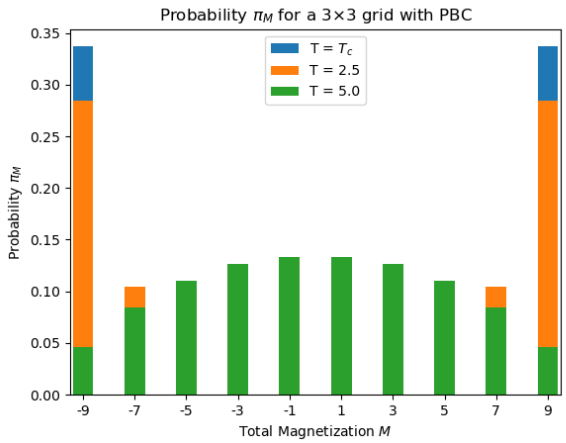
```
+--------------------+
Table for L = 2
-8 2
0 12
8 2
+--------------------+
Table for L = 3
-18 2
-10 18
-6 48
-2 198
2 144
6 102
+--------------------+
Table for L = 4
-32 2
-24 32
-20 64
-16 424
-12 1728
-8 6688
-4 13568
0 20524
4 13568
8 6688
12 1728
16 424
24 32
20 64
32 2
```

```
+--------------------+
Table for L = 5
-50 2
-42 50
-38 100
-34 850
-30 3140
-26 16300
-22 74500
-18 311800
-14 1014900
-10 2696080
-6 5230300
-2 7431800
2 7413900
6 5276500
10 2645740
14 1024150
18 314300
22 82750
26 14800
30 2470
```

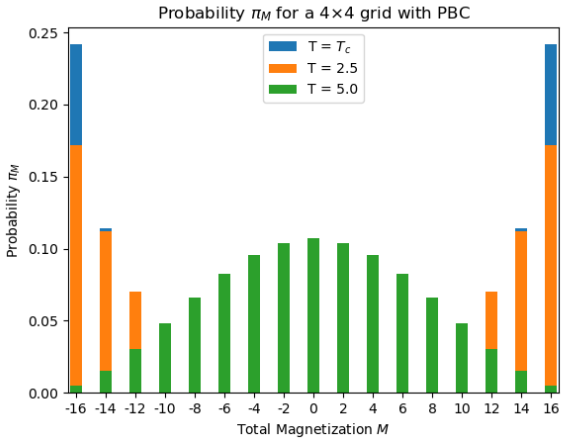
Lo cual concuerda con los valores presentados en la tabla de la slide 15, clase 5. Posterior a esto, se realizó el cálculo de la probabilidad de la magnetización π_M por spin M/N mediante la función `magnetization_probability`. Con esta función fue posible obtener los resultados de la figura 8. Es posible notar que a medida que aumenta la temperatura, pasando por $T_c = \frac{2}{\log\left(1+\sqrt{2}\right)} \approx 2.269$, pasamos de una distribución bimodal para valores bajos de T a una distribución unimodal simétrica centrada en $M = 0$ para L-par.



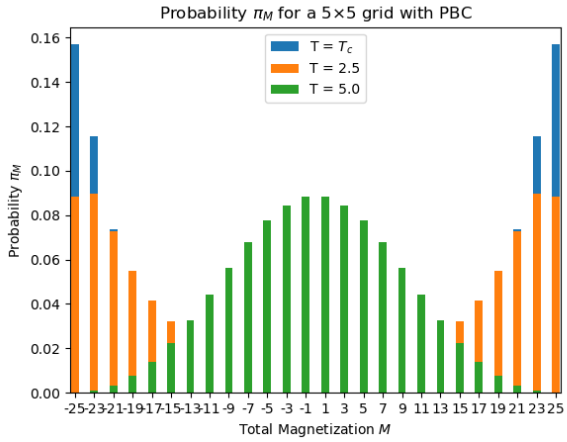
(a) Histograma para $\pi_M(T)$ en red de 2×2.



(b) Histograma para $\pi_M(T)$ en red de 3×3.



(c) Histograma para $\pi_M(T)$ en red de 4×4.



(d) Histograma para $\pi_M(T)$ en red de 5×5.

Figura 8: Histogramas de distribución de $\pi_M(T)$ en distintas redes tomando distintos valores para la temperatura T.

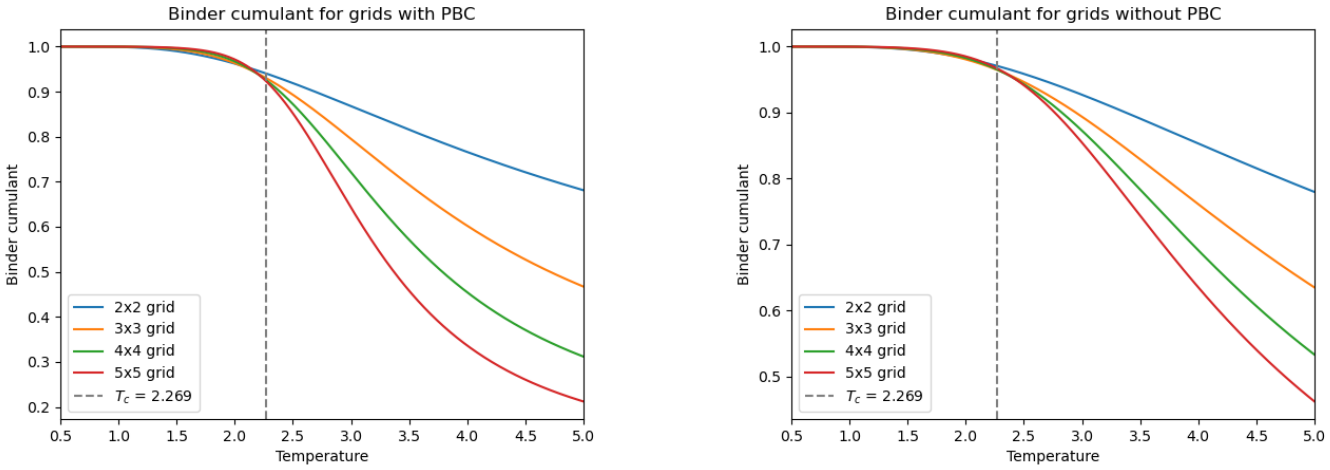
Finalmente, en el cálculo de los Binder cumulants, dado por la ec. (1):

$$B(T) = \frac{1}{2}\left[3 - \frac{\langle m^4(T)\rangle}{\langle m^2(T)\rangle^2}\right], \tag{1}$$

se lograron obtener los siguientes resultados para redes con PBC y sin PBC:
Para ambos casos, se cruzan cerca de T_c, aunque para redes sin PBC dicho cruce se encuentra más cerca de T_c.

Problema 3

Para comenzar, verificamos la energía media $\langle e\rangle$ y el calor específico c_V para una red de 6x6 con condiciones de borde periódicas utilizando $T = 2.0$ con el fin de comparar con los valores de la tabla presente en la slide 17, clase 5 mediante la función `parallel_thermo_markov_ising` del módulo `P3_funciones.py`. Los resultados obtenidos fueron:



(a) (b)

Figura 9: Binder cumulant para redes (a) con y (b) sin PBC.

```
Run: 1, <E/N> = -1.74772, c_V = 0.70854
Run: 2, <E/N> = -1.74339, c_V = 0.70234
Run: 3, <E/N> = -1.74638, c_V = 0.70266
Run: 4, <E/N> = -1.74142, c_V = 0.72948
Run: 5, <E/N> = -1.74492, c_V = 0.70737
```

Como podemos notar, los valores para $\langle e \rangle$ son muy similares, mientras que, si bien son cercanos, los obtenidos para c_V presentan mayor variación respecto a los presentados en la tabla de referencia de la slide 17, clase 5.

Posterior a esto, se realizó el cálculo de la magnetización absoluta promedio como función de la temperatura para distintas redes utilizando la función `abs_mag_for_T`, obteniendo los resultados de la figura 10.
Es posible notar que para la grilla de 32x32 tenemos resultados no esperados, en particular la presencia de valores negativos para magnetización absoluta, esto se puede deber a que al realizar el cálculo se presentó el error:

```
RuntimeWarning: overflow encountered in scalar add M_avg = sum(r[2] for r in results) /
N_samples
```

Uno de los problemas vistos en distintas iteraciones del código fue el uso de `N_samples` que se utiliza para correr cada configuración, ya que a medida que aumenta el tamaño de la red, se vió que también lo debe hacer el número de samples para tener una curva *"suave"*. Sin embargo, aparentemente debido a limitaciones del sistema en el cual se realizaron los cálculos, se tuvo que jugar con `N_samples` para que se ejecutara en un tiempo razonable de tiempo (el gráfico de la figura 10 tomó cerca de 4 horas).

Problema 4

Nuevamente, comenzamos por verificar los cálculos para la energía media $\langle e \rangle$ y calor específico c_V para una grilla de 6x6 variando la temperatura para compararlos con la tabla presente en la slide 34, clase 5. Los resultados se obtuvieron mediante la función `cluster_ising`, los cuales fueron:

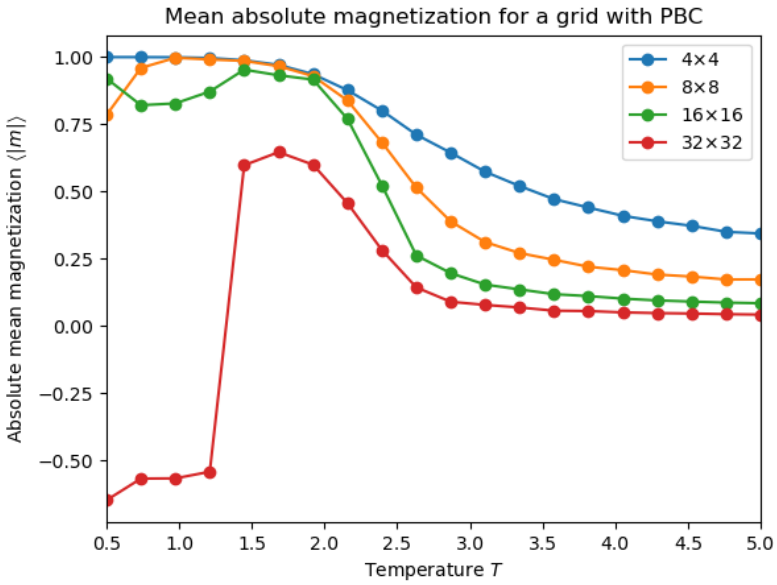


Figura 10: Magnetización absoluta promedio como función de la temperatura para redes de 4×4, $8\times$, 16×16 y 32×32 con PBC.

```
T = 0.5, <e> = -2.0000, c_V = 0.00000
T = 1.0, <e> = -1.9970, c_V = 0.02482
T = 1.5, <e> = -1.9510, c_V = 0.19858
T = 2.0, <e> = -1.7452, c_V = 0.70124
T = 2.5, <e> = -1.2821, c_V = 1.00153
T = 3.0, <e> = -0.8866, c_V = 0.55494
T = 3.5, <e> = -0.6857, c_V = 0.29737
T = 4.0, <e> = -0.5665, c_V = 0.18782
```

Presentando una buena concordancia con los valores de la tabla presente en la slide 17, clase 5. Posterior a esto, se realizó el cálculo para los histogramas de la magnetización y gráficos de Binder cumulant para redes de 6×6, 16×16 y 32×32. Nuevamente, debido a limitaciones del sistema en el cual se realizaron los cálculos, se debió abortar el cálculo para una red de 64×64. Para los histogramas de la magnetización absoluta promedio por spin, se encuentran resumidos en la figuras 11, 12 y 13.

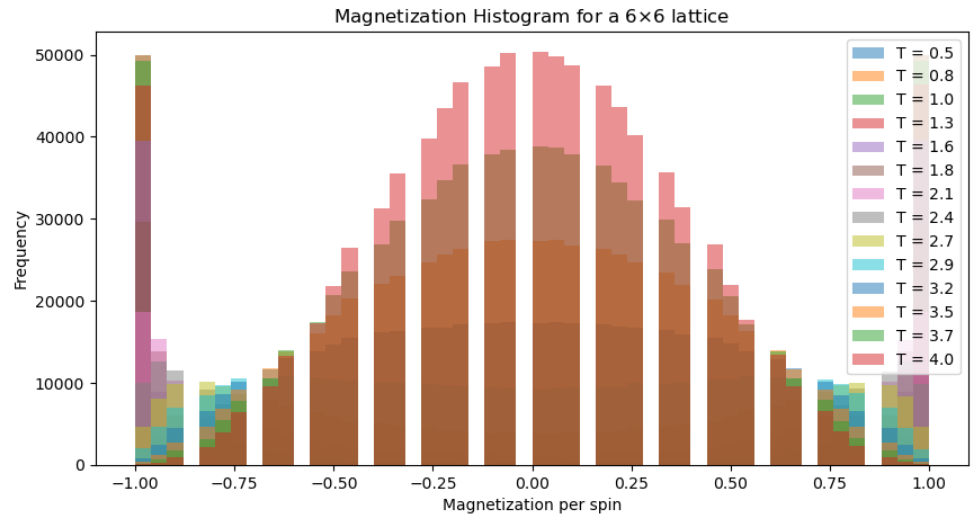


Figura 11: Histograma para la magnetización absoluta promedio como función de la temperatura T para una red de 6×6.

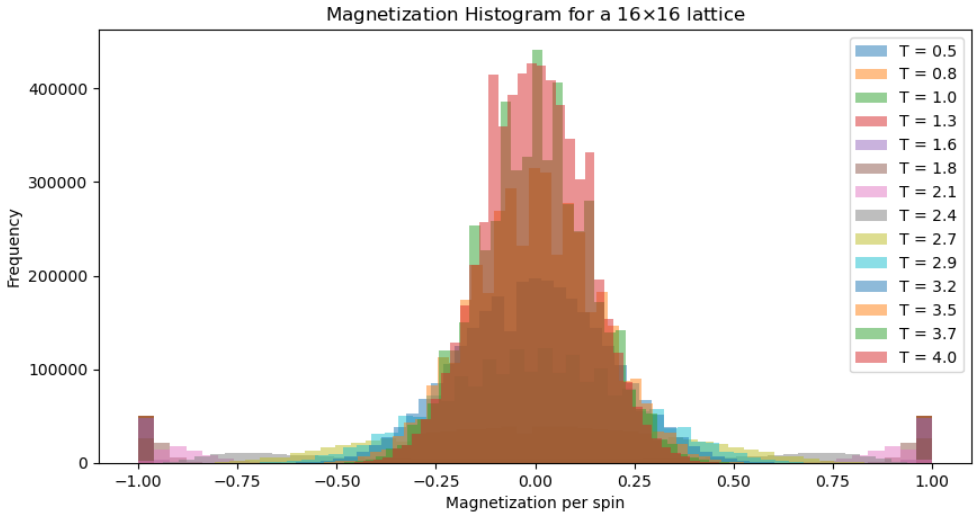


Figura 12: Histograma para la magnetización absoluta promedio como función de la temperatura T para una red de 16×16.

Podemos notar la tendencia en los histogramas de la magnetizatición de que a medida quela temperatura aumenta, también lo hace la densidad de estados cercana a $\langle |m| \rangle = 0$. Finalmente, para los Binder cumulants, los resultados se encuentran en la figura 14:

Los resultados no muestran una buena intersección sobre T_c, esto podría deberse a la cantidad de puntos utilizados para generar los valores de T, 14 en nuestro caso, por lo que si se aumentara la cantidad de puntos, podría haber una mayor concordancia respecto a resultados esperados y los obtenidos en el problema 2.

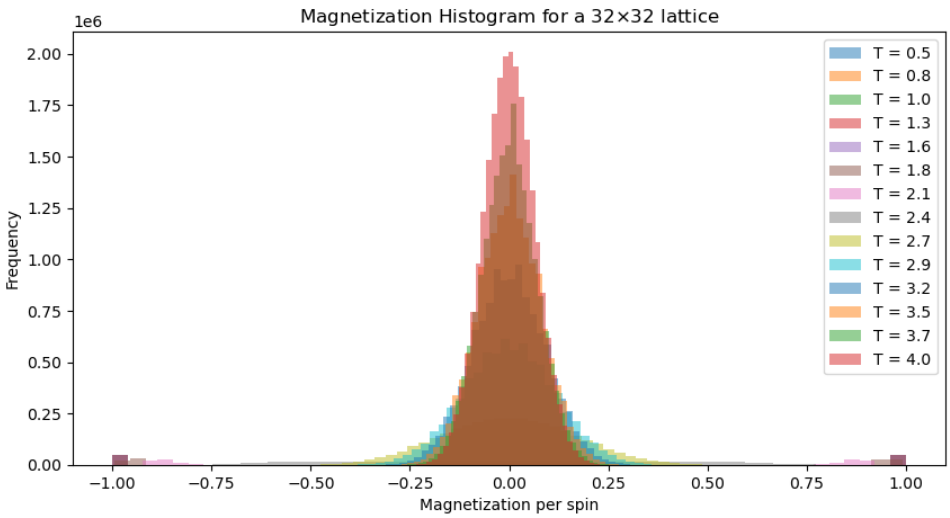


Figura 13: Histograma para la magnetización absoluta promedio como función de la temperatura T para una red de 32×32.

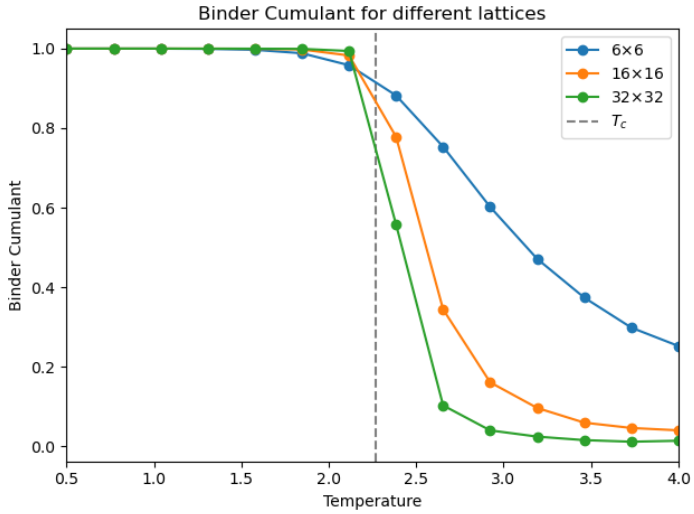


Figura 14: Binder cumulants para redes 6×6, 16×16 y 32×32 con PBC.

Comentarios finales

Si bien los tiempos de cálculo para algunas funciones son bastante altos, se realizaron diversos intentos por optimizarlo, desde reducir la cantidad de *"loops"* dentro de las definiciones de las funciones requeridas, el uso del módulo `Cython` para traducir el código a `C`, también el uso del módulo `numba`, incluso en algunas iteraciones se utilizó `cupy` para correr el código en una GPU en lugar de una CPU. Sin embargo, la mayoría de las funciones para cada problema se realizaron utilizando el módulo `joblib` para optimizar la ejecución del código utilizando varios núcleos del procesador a la vez.

El código de este trabajo sin dudas puede ser optimizado, sin embargo debido a falta de conocimiento, sería necesario aún más tiempo del ya otorgado para realizarlo. Finalmente, hubieron algunas iteraciones en las cuales se intentó escribir el código en `C++`, sin grandes resultados en algunos casos, por lo cual, debido a limitaciones de tiempo, se decidió utilizar este "*approach*".