Voluntario 2: Simulación del modelo de Ising con la dinámica de Kawasaki.

Física Computacional. Grado en Física. Universidad de Granada.

Alejandro José Zarzuela Moreno

15 de mayo, 2024

1. Objetivos

Usar el algoritmo de Metrópolis explicado en clase para simular y estudiar las propiedades de un modelo de Ising en dos dimensiones mediante la dinámica de Kawasaki.

2. Planteamiento del problema y funciones

Este programa se basa en el obligatorio 2, donde se usa el algoritmo de Metrópolis de manera similar.

Se empieza definiendo las constantes y parámetros del sistema: la dimensión de la cuadrícula N, la temperatura T (se tomará entre 0 y 5), el número de pasos Monte Carlo `pmc` de la simulación y el parámetro `skip`, que indica cada cuántas iteraciones se muestrean los resultados del algoritmo de Metrópolis. Además, para elegir la magnetización total del sistema se puede cambiar la variable `condIni`, cuyos valores posibles son $'magnAleatoria'$, $'magnNula'$ o bien $'magnX'$ (con valor $M = X$).

Para comenzar se explicará cómo se crea la matriz de espines s en función del valor de `condIni`. Habrá que tener en cuenta una parte de las condiciones de contorno, ya que los bordes izquierdo y derecho están conectados pero el superior e inferior no, sino que cada uno tiene espín de un signo diferente. En nuestro caso se ha elegido que el borde superior tenga espín -1 y el inferior $+1$.

- Creación de `s` (M aleatoria):

Se usará una matriz s de dimensión $N \times N$ que se rellenará con ± 1 de manera aleatoria usando `np.random.choice()`. Luego, se colocarán los espines -1 y $+1$ en los bordes superior e inferior, respectivamente, usando un bucle:

```
if condIni == "magnAleatoria":
    s = np.random.choice([-1,+1], size=(N,N)).astype(np.int8)
    for j in range(N):
        s[0,j] = -1
        s[N-1,j] = 1
```

• Creación de `s` (M nula):

Para empezar se rellena una matriz s de dimensión $N \times N$ con espines de valor 1. Una vez hecho esto, se elige un espín aleatorio que no pertenezca a la fila 0 (fila superior) ni a la $N-1$ (fila inferior), y se le otorga un valor -1. Dado que el objetivo es que la magnetización total del sistema sea nula, esto se hace para $N^2/2 - N$ espines, ya que se busca que la mitad de los espines sean de cada tipo.

```
elif condIni == "magnNula":
    s = np.ones((N,N)).astype(np.int8)
    for _ in range(int(N**2/2)-N):
        i = np.random.randint(1,N-1)
        j = np.random.randint(0,N-1)
        while s[i,j] == -1:
            i = np.random.randint(1,N-1)
            j = np.random.randint(0,N-1)
        s[i,j] = -1
    for j in range(N):
        s[0,j] = -1
        s[N-1,j] = 1
```

El término $-N$ está porque al terminar el bucle se rellenará una fila completa con espines -1 (la superior) para cumplir las condiciones de contorno en los bordes superior e inferior.

• Creación de `s` ($M = X$):

Se crea una matriz de magnetización nula de la misma forma que cuando `condIni = "magnNula"`. A continuación, si se quiere que la magnetización sea, por ejemplo, 8, habrá que cambiar entonces 8/2=4 espines negativos a positivos. Es por esto que en el bucle se itera hasta $X/2$.

```
(...)
if X > 0:
    for _ in range(int(X/2)):
        i = np.random.randint(1,N-1)
        j = np.random.randint(0,N-1)
        while s[i,j] == 1:
            i = np.random.randint(1,N-1)
            j = np.random.randint(0,N-1)
        s[i,j] = 1
else:
    for _ in range(int(X/2)):
        i = np.random.randint(1,N-1)
        j = np.random.randint(0,N-1)
        while s[i,j] == -1:
            i = np.random.randint(1,N-1)
            j = np.random.randint(0,N-1)
        s[i,j] = -1
```

• Condiciones de contorno: periodicidad horizontal

Se implementa la función `condContornoH()` para conocer qué espines rodean al estudiado. Cuando el espín que se está estudiando está en el borde derecho de la matriz $(i, N-1)$, considera que el espín $i, 0$ está a la derecha. Cuando se está estudiando un espín del borde izquierdo $(i, 0)$, se considera que el espín a su izquierda es el $(i, N-1)$. En cualquier otro caso, simplemente toma los espines que están ubicados a la derecha y a la izquierda de manera normal.

```
def condContornoH(j,N):
    if j==N-1:        #
        left = j-1    #
        right = 0     #
    elif j==0:        #
        left = N-1    # Periodicidad horizontal
        right = 1     #
    else:             #
        left = j-1    #
        right = j+1   #
    return left,right
```

Los valores `left` y `right` que devuelve esta función se usan como índices de la matriz `s`, como se mostrará más adelante.

- Bucle principal:

El bucle principal es similar al del problema obligatorio 2, solo que con algunos cambios. Para empezar, en cada instante temporal se elige un espín aleatorio p_1 que pertenece a cualquier fila menos a la superior y a las dos inferiores. Esto es así para no alterar el espín de primera fila ni de la última.

```
for w in range(t):

    # Se elijen dos partículas aleatorias vecinas [ p1=(i,j) ; p2=(u,v) ]
    i = np.random.randint(1,N-2)
    j = np.random.randint(0,N)
```

Luego la variable `flip` toma un valor aleatorio ± 1. El valor -1 hace que la pareja p_2 elegida sea vertical (se elije el espín que hay justo debajo de p_1), y el valor $+1$ hace que la pareja p_2 elegida sea horizontal (se elije el espín que hay justo a la derecha de p_1).

De esta forma, las coordenadas de cada espín son $p_1 = (i, j)$ y $p_2 = (u, v)$, siendo u y v función de i, j y `flip`.

```
    flip = np.random.choice([-1,1])
    if flip == -1:  #
        u = i + 1   # Vecinas verticales
        v = j       #
    else:               #
        u = i           #
        if j == N-1:    # Vecinas horizontales
            v = 0       #
        else:           #
            v = j + 1   #
```

Después se llama a la función `isingKawasaki()`, que en caso de que $p_1 = p_2$ no hace nada y hace que el bucle principal no avance, y en caso de que $p_1 \neq p_2$ evalúa el estado del sistema e intercambia la posición de p_1 y p_2 si procede, y hace que el bucle principal avance.

```
    # Una vez elegida la pareja, se decide si intercambiar los espines o no
    w = isingKawasaki(w,flip,s,i,j,u,v,N,T)
```

Finalmente se escribe el estado de la red en el fichero. Está configurado para que se escriba cada `skip` pasos Monte Carlo. También se añaden a la cola de sus respectivos vectores los resultados de la magnetización en la mitad superior, en la mitad inferior y la energía media por partícula en ese instante temporal. Más adelante se explicarán estas funciones.

```
    # Se guarda el estado de la red de espines y de otras magnitudes de interés
    if w%skip==0:
        fichero.write("\n")
        for i in range(N):
            fichero.write(' '.join(map(str, s[i])) + "\n")

        vMagnArriba.append(magnArriba(s,N))
        vMagnAbajo.append(magnAbajo(s,N))
        vEnergiaMPP.append(energiaMediaPorParticula(s,N))
```

- Intercambio de espines: función `isingKawasaki()`

Ya se ha visto cuándo se llama a la función `isingKawasaki()` en el bucle principal, y ahora se comentará su funcionamiento.

Lo primero que se hace es comprobar si los dos espines de la pareja son iguales, en cuyo caso no se realiza ningún cálculo (salga lo que salga el sistema quedaría igual) y se indica al bucle principal que no avance el tiempo (`return w-1`).

En caso de que los dos espines no sean iguales, se realiza el cálculo. Comienza determinando en qué posición están los espines que rodean a p_1 y a p_2, usando la función `condContornoH()` y asignando el valor de `left` o `right` a los correspondientes índices de la posición de los espines:

```
if s[i,j] != s[u,v]:
    up1,down1 = i-1,i+1
    up2,down2 = u-1,u+1
    left1,right1 = condContornoH(j,N)
    left2,right2 = condContornoH(v,N)
```

Una vez se conocen los espines de la pareja y los circundantes, se calcula la probabilidad p mediante la función `v_pE()` ó `h_pE()` según la pareja sea horizontal o vertical:

```
    if flip == -1:
        pE = v_pE(T,s,i,j,u,v,left1,left2,right1,right2,up1,down2)  # Caso pareja vertical
    else:
        pE = h_pE(T,s,i,j,u,v,left1,right2,up1,up2,down1,down2)  # Caso pareja horizontal
```

Finalmente, se compara p_E (o bien $p_E = 1$ si $p_E > 1$) con un número aleatorio n entre 0 y 1, y en caso de que p_E sea mayor, se intercambian los espines p_1 y p_2:

```
    if pE>1:      #
        p = 1     # Evaluación de p
    else:         #
        p = pE    #
    # Permutación de espines s1,s2 = s2,s1
    n = np.random.rand()  # Número aleatorio entre 0 y 1
    if n<p:
        s[i,j], s[u,v] = s[u,v], s[i,j]
    return w  # Los dos espines eran diferentes -> Avanzar w
```

- Cálculo de probabilidad: funciones `v_pE()` y `h_pE()`

En la función `isingKawasaki()` se usan las funciones `v_pE()` y `h_pE()` para calcular las probabilidades p. Ambas funciones funcionan exactamente igual, solo que en una se toman las posiciones correspondientes a una pareja vertical y en otra a una pareja horizontal.

Lo primero que se calcula es `deltaE`, es decir, la variación de energía del sistema supuesto el cambio. Esta expresión se ha deducido a partir de la que se usa en el obligatorio 2 para un espín, y el desarrollo de dicha deducción se puede encontrar en GitHub en la carpeta 'voluntario2\Resultados' junto con el resto del código. Tras calcular `deltaE`, el valor de p se calcula como $p = \exp(-\Delta E/T)$.

```
def v_pE(T,s,i,j,u,v,left1,left2,right1,right2,up1,down2):
    deltaE = 2*s[i,j]*(s[i,right1]+s[i,left1]+s[up1,j]
                       -s[u,right2]-s[u,left2]-s[down2,v]) # Pareja vertical
    return np.exp(-deltaE/T)

def h_pE(T,s,i,j,u,v,left1,right2,up1,up2,down1,down2):
    deltaE = 2*s[i,j]*(s[up1,j]+s[down1,j]+s[i,left1]
                       -s[up2,v]-s[down2,v]-s[u,right2]) # Pareja horizontal
    return np.exp(-deltaE/T)
```

• Magnetización del sistema: funciones `magnSistema()`, `magnArriba()` y `magnAbajo()`

Al final del bucle principal, se calculan las magnetizaciones en cada mitad del sistema llamando a las respectivas funciones. Dado que el funcionamiento es el mismo, se les da a las dos una explicación en conjunto, así como a `magnSistema()`.

El funcionamiento es el siguiente: se inicializa a cero una variable que se irá sumando. A continuación se crean dos bucles anidados, uno que recorre las filas y otro las columnas. En el interior de estos bucles se suma a la variable cada espín del sistema. Finalmente se divide entre el número de espines sobre el que se ha sumado para obtener el valor de la magnetización por partícula en ese dominio.

La diferencia entre las tres funciones está en cuál es el intervalo de filas sobre el que se suman los espines: en `magnSistema()` se suman todos los espines del sistema, en `magnArriba()` se suman sólo los de la mitad superior (`i=0` a `i=(N-1)/2`), y en `magnAbajo()` se suman sólo los de la mitad inferior (`i=1+(N-1)/2` a `i=(n-1)`).

```
def magnSistema(s,N):
    magn = 0
    for i in range(N):
        for j in range(N):
            magn += s[i,j]
    return magn

def magnArriba(s,N):
    magnUp = 0
    for i in range(int((N-1)/2)):
        for j in range(N):
            magnUp += s[i,j]
    return magnUp/(N**2/2)

def magnAbajo(s,N):
    magnDown = 0
    for i in range(int((N-1)/2),N):
        for j in range(N):
            magnDown += s[i,j]
    return magnDown/(N**2/2)
```

• Cálculo de la energía media por partícula: función `energiaMediaPorParticula()`

La tercera función que se llama al final del bucle principal es `energiaMediaPorParticula()`, que para un instante temporal calcula el valor promedio de la energía por partícula del sistema.

Para ello, se usa la expresión de la energía total del sistema

$$E = -\frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} s(i,j)\left[s(i,j-1)+s(i,j+1)+s(i+1,j)+s(i-1,j)\right] \tag{1}$$

```
def energiaMediaPorParticula(s,N):
    E = 0
    for i in range(N):
        for j in range(N):
            up,down = i-1,i+1
            left,right = condContornoH(j,N)
            E += s[i,j]*(s[up,j]+s[down,j]+s[i,left]+s[i,right])
```

Nótese que en la suma de la función no se incluye el producto por '$-0{,}5$', sino que como es común para todos los sumandos se realiza al final una sola vez para ahorrar cálculos. En este momento es también cuando se divide el valor resultante de la energía entre N^2 para obtener así el valor de la energía media por partícula:

```
    return -0.5*E/N**2
```

• Cálculo del calor específico: función `calcCalorEspecifico()`

El calor específico se calcula como

$$c_N = \frac{1}{N^2T^2}\left[\langle E^2\rangle - \langle E\rangle^2\right]. \tag{2}$$

En nuestro caso se usarán para calcular el promedio los valores de energía promedio por partícula, de modo que en el cálculo de c_N no se dividirá por N^2.

La primera parte de esta función consiste en calcular $\langle E^2 \rangle$ y $\langle E \rangle$ conocidos los valores de E. Para realizar estos promedios, se suman los correspondientes valores (E^2 para $\langle E^2 \rangle$ y E para $\langle E \rangle$) y luego se divide entre el número de muestras en instantes temporales que se tiene de E, es decir `len(E)`.

```
def calcCalorEspecifico(N,T,E):
    promECuad = 0
    promE = 0
    for t in range(len(E)):
        promECuad += E[t]**2
        promE += E[t]
    promECuad = promECuad/len(E)
    promE = promE/len(E)
```

Hecho esto, solo queda sustituir en la ecuación (2) sin dividir entre N^2:

```
    return (promECuad-promE**2)/(T)**2
```

De la misma forma que se calcula el calor específico mediante los promedios de E, en la función `calcSusceptMagnetica()` se calcula la susceptibilidad magnética de una región usando los promedios de su magnetización.

3. Resultados

Todos los ficheros de interés para cada apartado están dentro de `voluntario2\Resultados`. Si para cualquiera de las ejecuciones del programa se quiere conocer los valores de `N, T, pmc, skip`, el tiempo que ha tardado la ejecución, la magnetización del sistema o el calor específico c_N, estos se pueden encontrar en los ficheros *'ik_ data_ X.dat'* que hay en cada apartado. Cada apartado se divide a su vez en dos: uno en el que la magnetización del sistema es nula ($M = 0$) y otro en el que no ($M \neq 0$).

3.1. Apartado 1: Dinámica del modelo para varias temperaturas

A continuación se muestran fotogramas correspondientes a $t = 20000$ pmc y $t = 40000$ pmc para una red 10×10 a distintas temperaturas:

- **Ap. 1 - Magnetización nula ($M = 0$):**

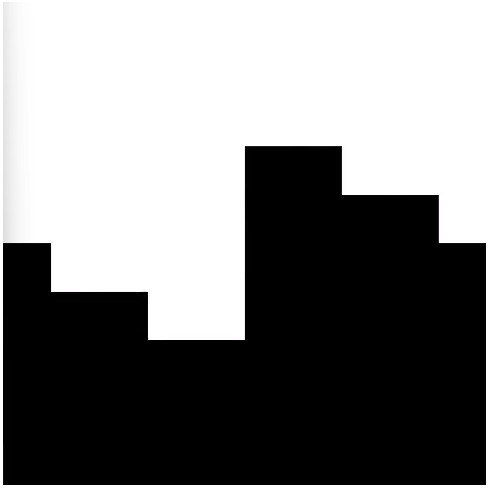


Figura 1: $T = 1$, 20000 pmc.

Figura 2: $T = 1$, 40000 pmc.

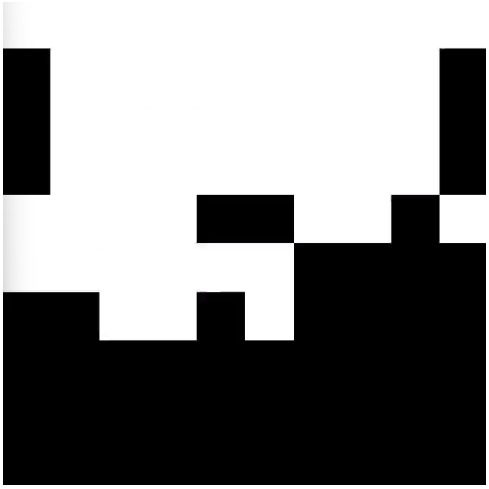


Figura 3: $T = 2$, 20000 pmc.

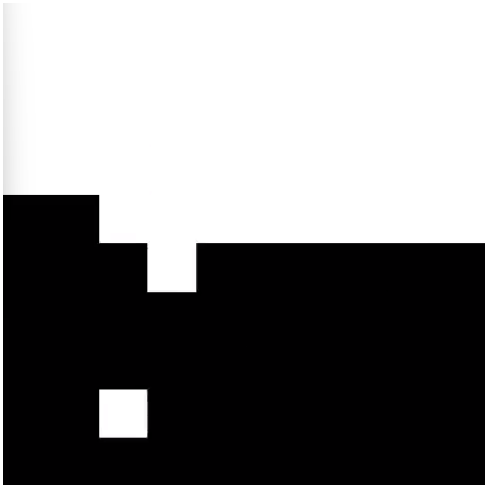


Figura 4: $T = 2$, 40000 pmc.

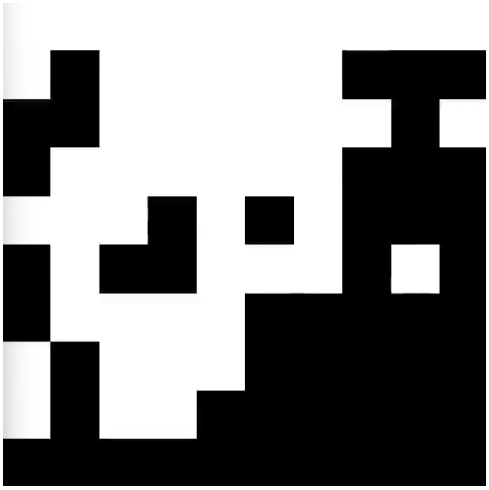


Figura 5: $T = 3$, 20000 pmc.

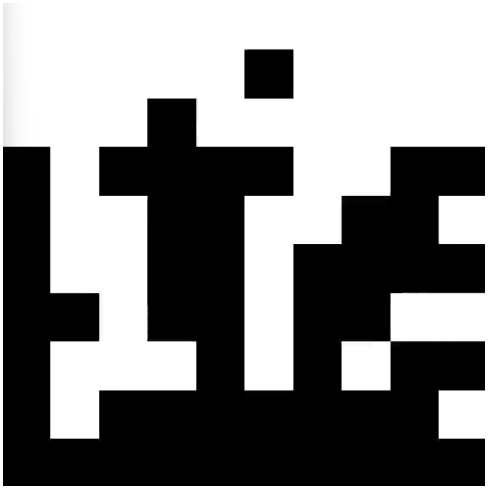


Figura 6: $T = 3$, 40000 pmc.

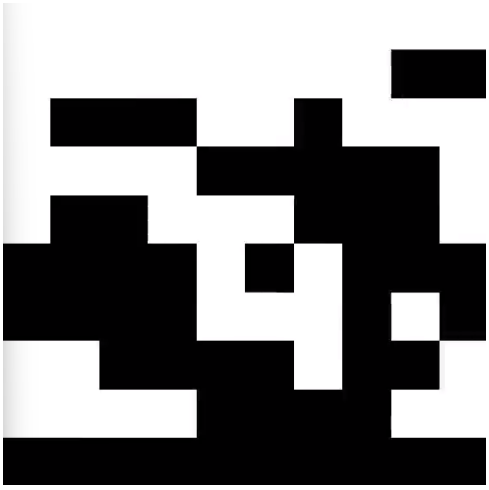


Figura 7: $T = 4$, 20000 pmc.

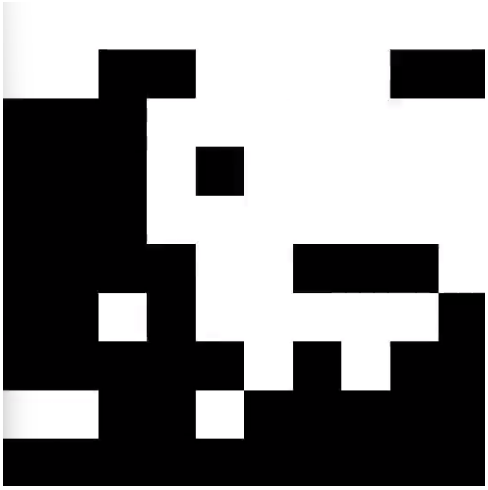


Figura 8: $T = 4$, 40000 pmc.

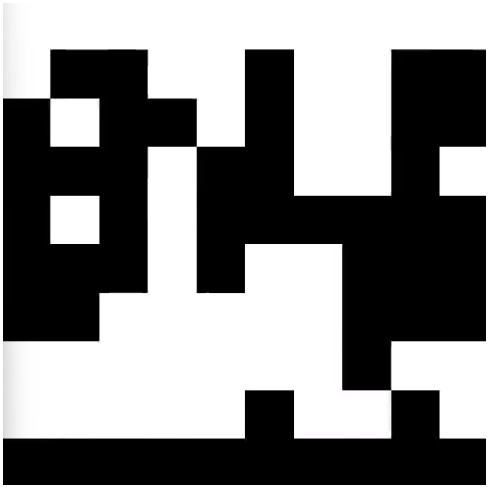


Figura 9: $T = 5$, 20000 pmc.

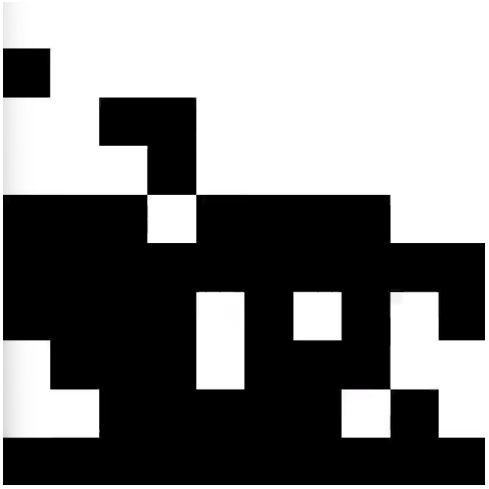


Figura 10: $T = 5$, 40000 pmc.

- **Ap. 1 - Magnetización aleatoria ($M \neq 0$):**

Figura 11: $T = 1$, 20000 pmc.

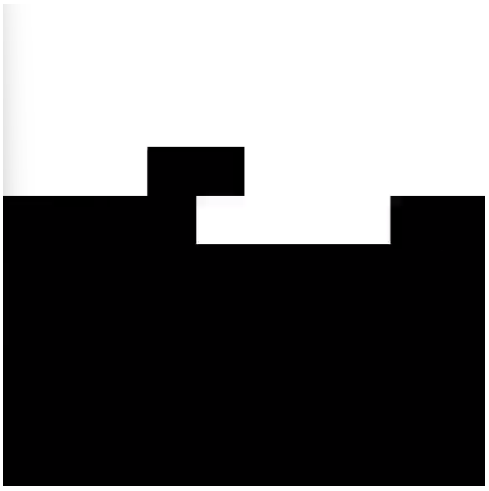


Figura 12: $T = 1$, 40000 pmc.

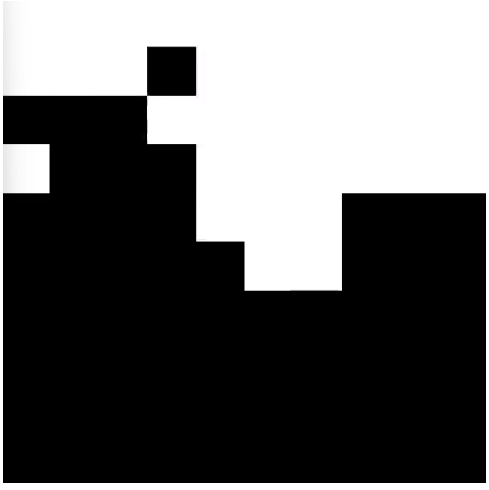


Figura 13: $T = 2$, 20000 pmc.

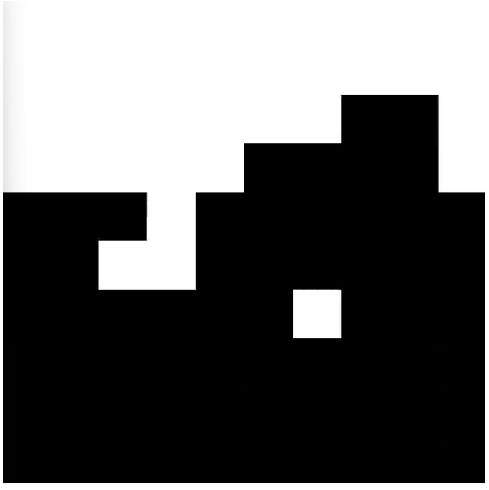


Figura 14: $T = 2$, 40000 pmc.

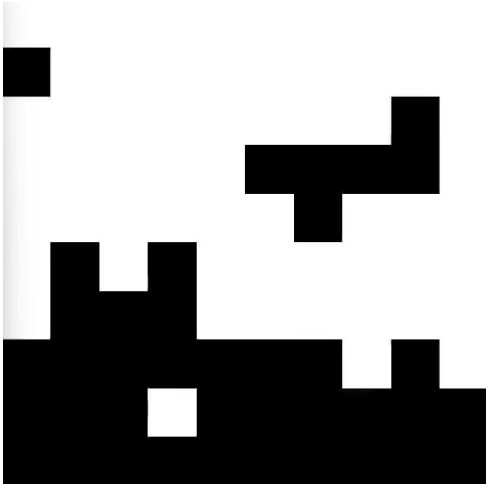


Figura 15: $T = 3$, 20000 pmc.

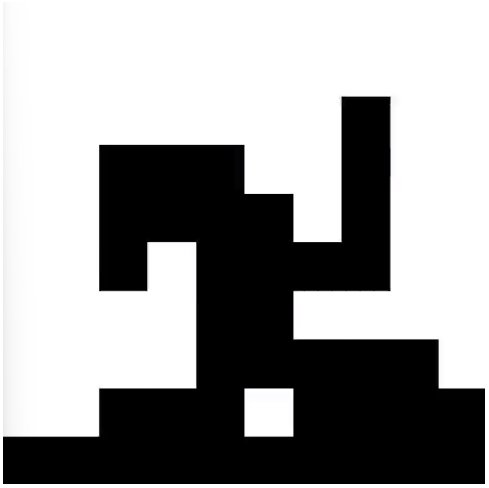


Figura 16: $T = 3$, 40000 pmc.

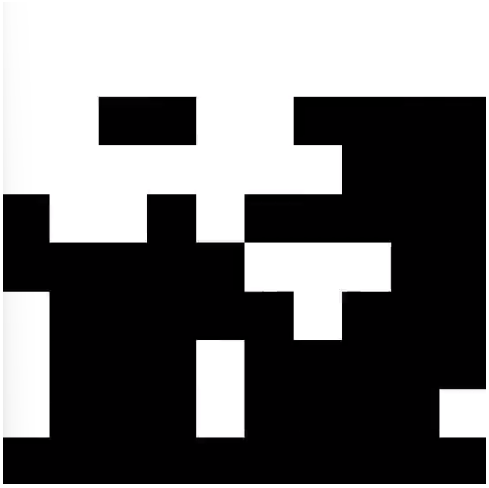


Figura 17: $T = 4$, 20000 pmc.

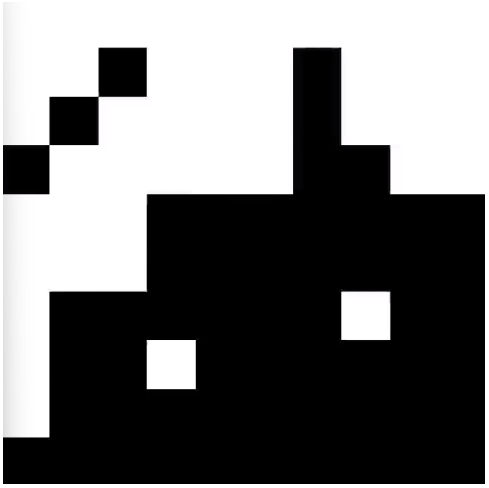


Figura 18: $T = 4$, 40000 pmc.

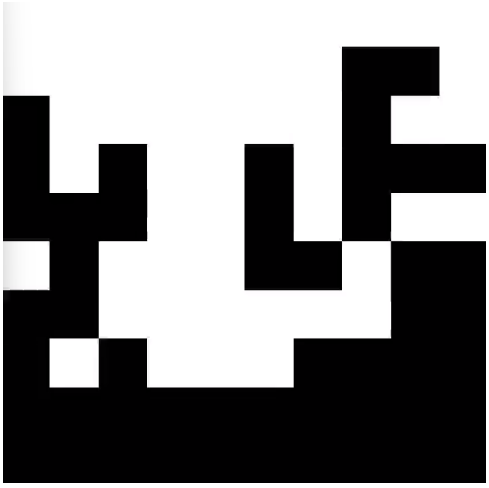


Figura 19: $T = 5$, 20000 pmc.

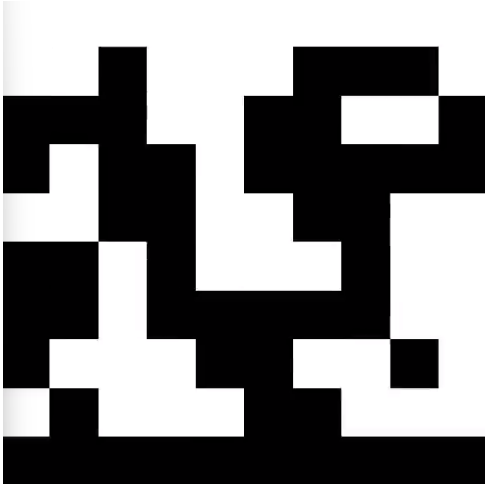


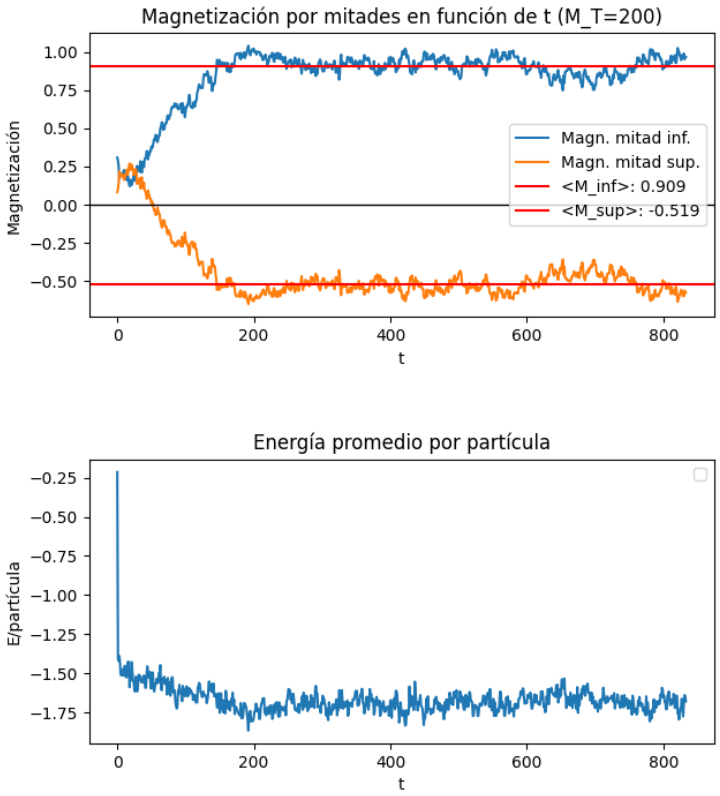
Figura 20: $T = 5$, 40000 pmc.

3.2. Apartado 2: Curva de magnetización por partícula en cada dominio

En este apartado se calculará la magnetización por partícula en cada dominio y se representará gráficamente en función de la temperatura. Esto se hará para diferentes valores de N.

Para esta representación gráfica se usa el programa `plot.py` que hay en cada carpeta dentro de los resultados. El resto de los datos de interés se encuentra también en esas carpetas. En cada una de las ejecuciones se muestran los valores promedio de la magnetización de cada región en gráficos como el de la derecha.

Para el promedio de la magnetización se tienen en cuenta los datos a partir de la mitad de la ejecución del programa, para promediar sobre estados más cercanos al equilibrio.
También se muestran los valores de la magnetización en los archivos '`ik_info_T.dat`'.



(Este gráfico en concreto corresponde con una red 32×32 con $M = 200$ a temperatura $T = 2$.)

- **Ap. 2 - Magnetización nula ($M = 0$):**

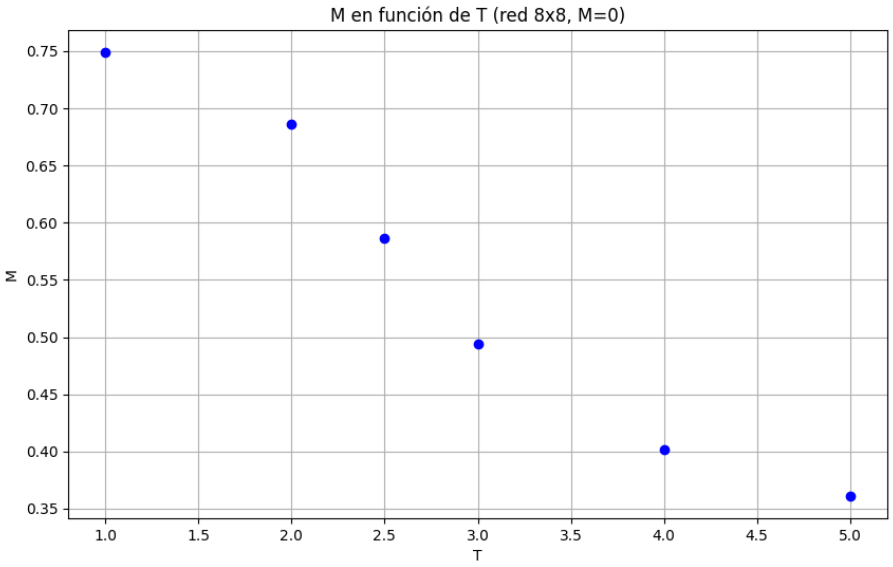


Figura 21: Curva de magnetización promedio por partícula en cada región en función de la temperatura para una red 8x8.

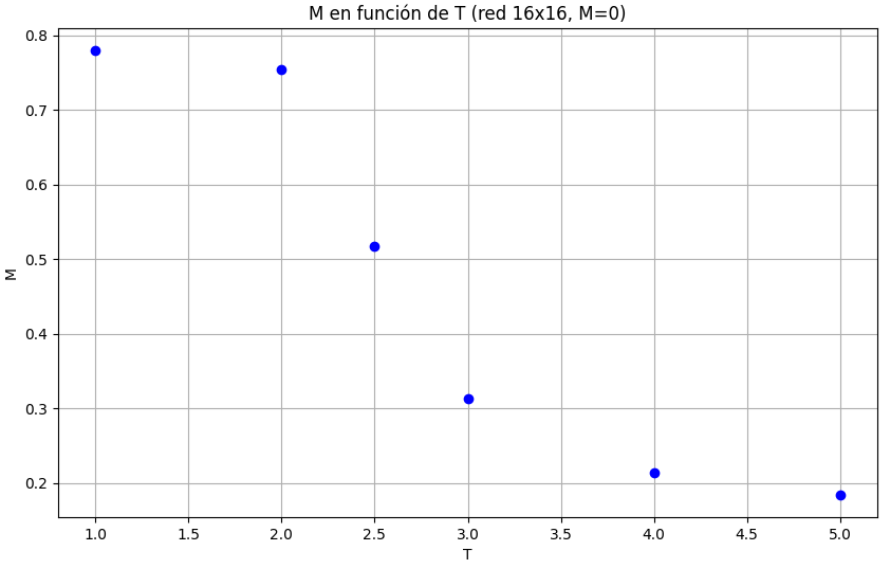


Figura 22: Curva de magnetización promedio por partícula en cada región en función de la temperatura para una red 16×16.

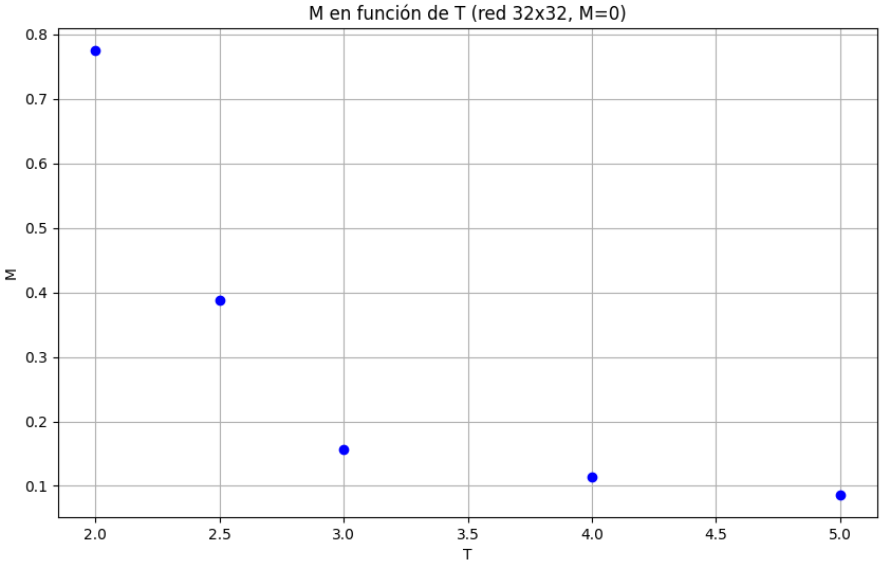


Figura 23: Curva de magnetización promedio por partícula en cada región en función de la temperatura para una red 32×32.

- **Ap. 2 - Magnetización no nula ($M \neq 0$):**

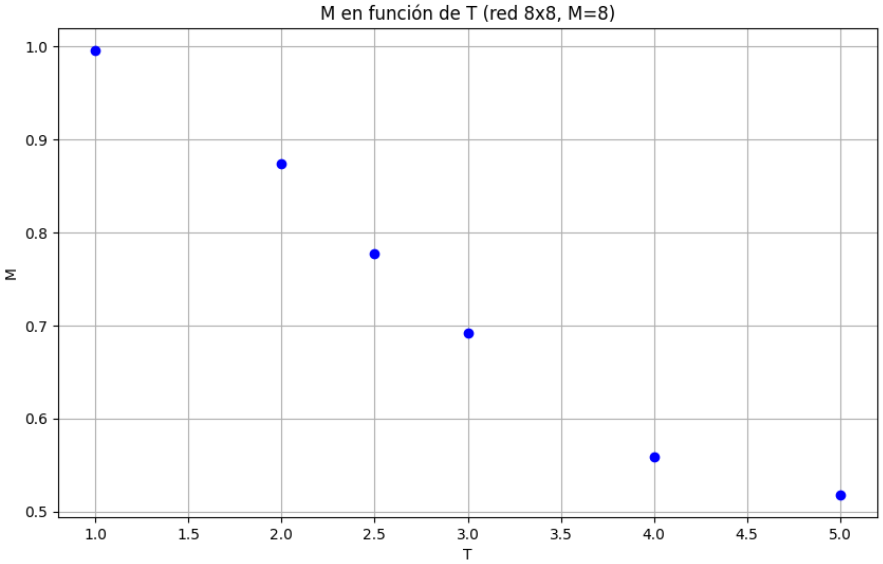


Figura 24: Curva de magnetización promedio por partícula en cada región en función de la temperatura para una red 8x8.

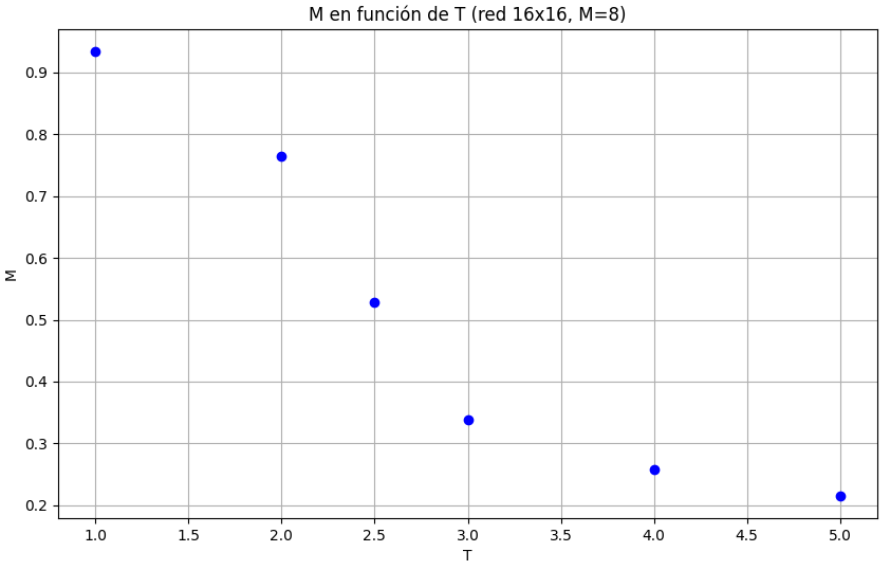


Figura 25: Curva de magnetización promedio por partícula en cada región en función de la temperatura para una red 16×16.

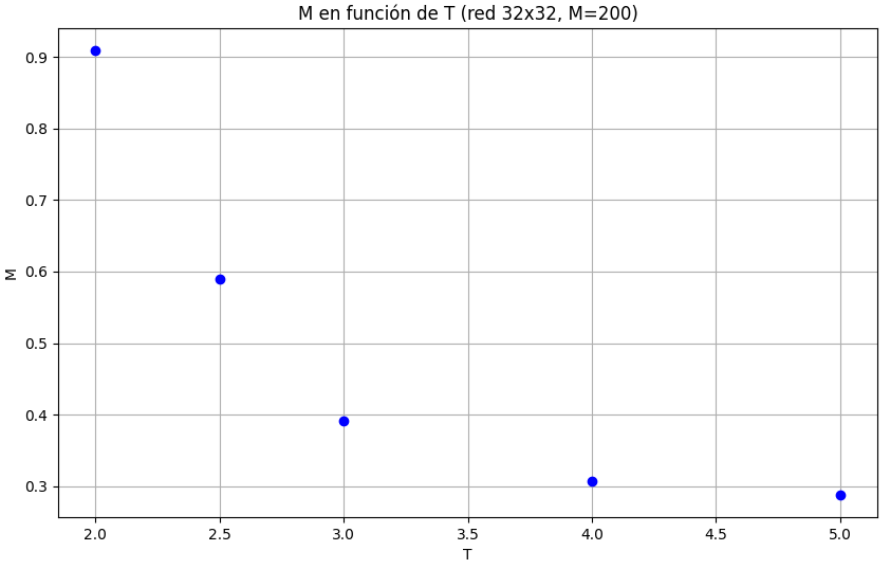


Figura 26: Curva de magnetización promedio por partícula en cada región en función de la temperatura para una red 32×32.

3.3. Apartado 4: Energía media por partícula en función de T

En cada simulación se obtiene una gráfica en la que se aprecia cómo la energía promedio por partícula es una curva que, para temperaturas bajas, cae hasta estabilizarse en torno a un valor promedio. Para temperaturas más altas, la curva presenta oscilaciones cada vez mayores y en general la energía promedio es mayor (menos negativa).

A continuación se representa gráficamente la energía media por partícula en función de la temperatura. Se hará esta representación para varios tamaños de red.

- **Ap. 4 - Magnetización nula ($M = 0$):**

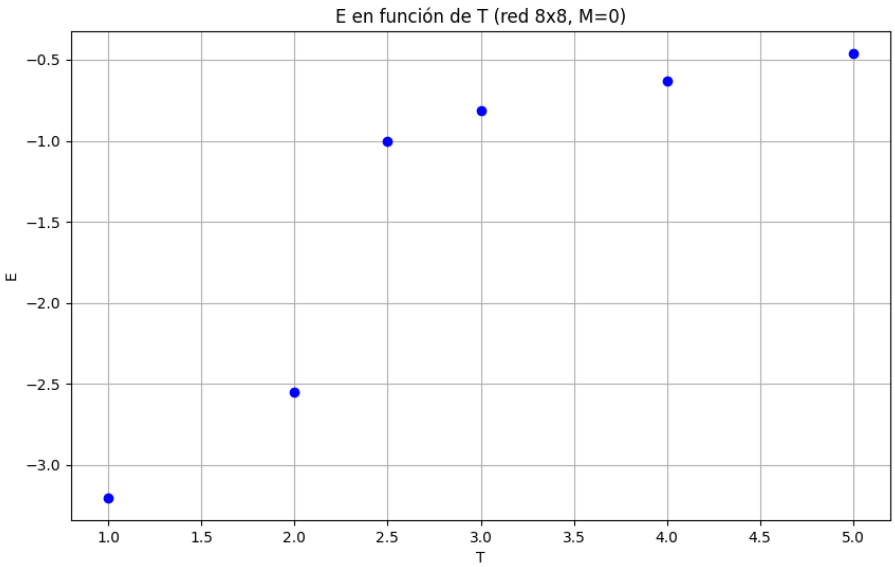


Figura 27: Energía en función de T para una red 8×8 y $M = 0$.

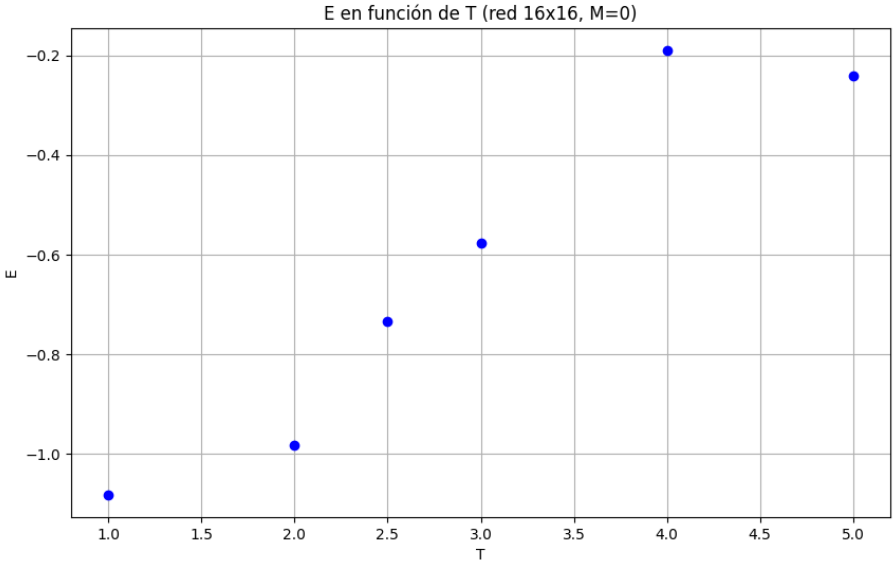


Figura 28: Energía en función de T para una red 16×16 y $M = 0$.

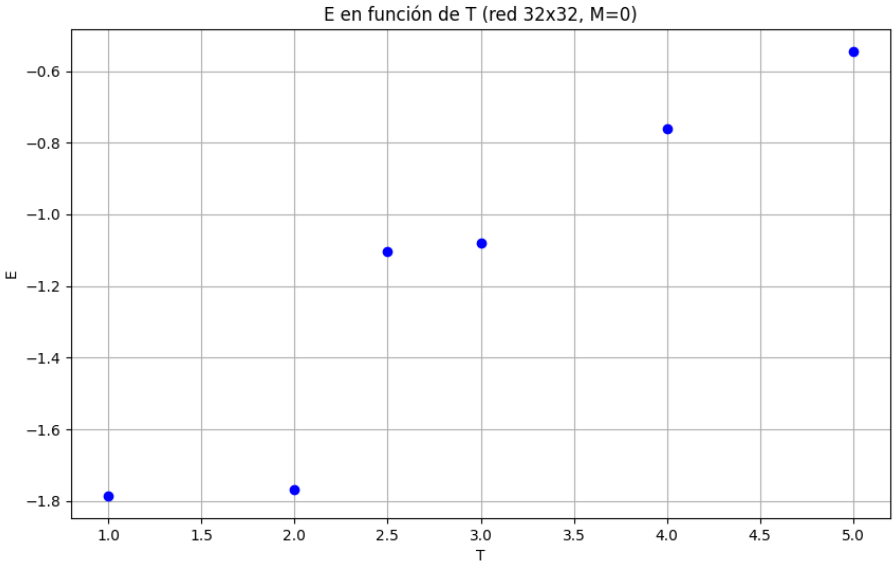


Figura 29: Energía en función de T para una red 32×32 y $M = 0$.

- **Ap. 4 - Magnetización no nula ($M \neq 0$):**

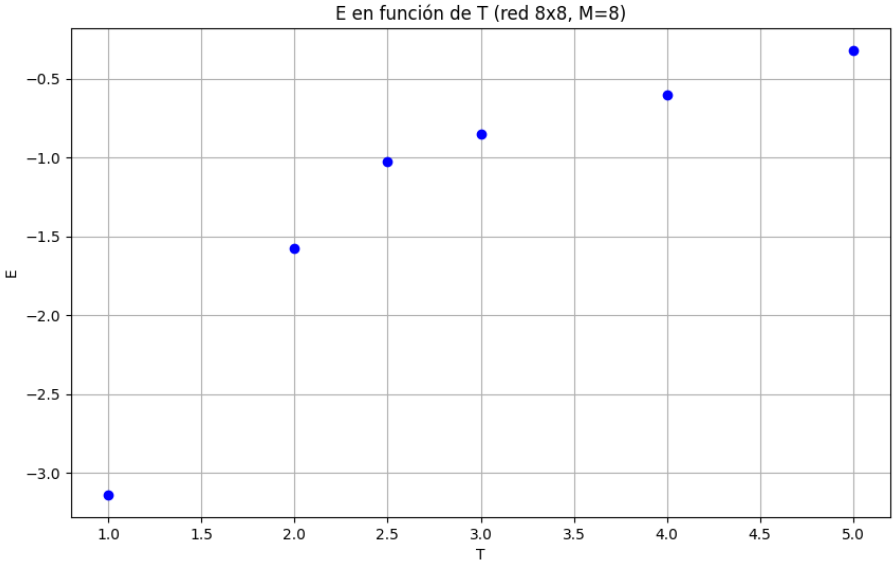


Figura 30: Energía en función de T para una red 8×8 y $M = 8$.

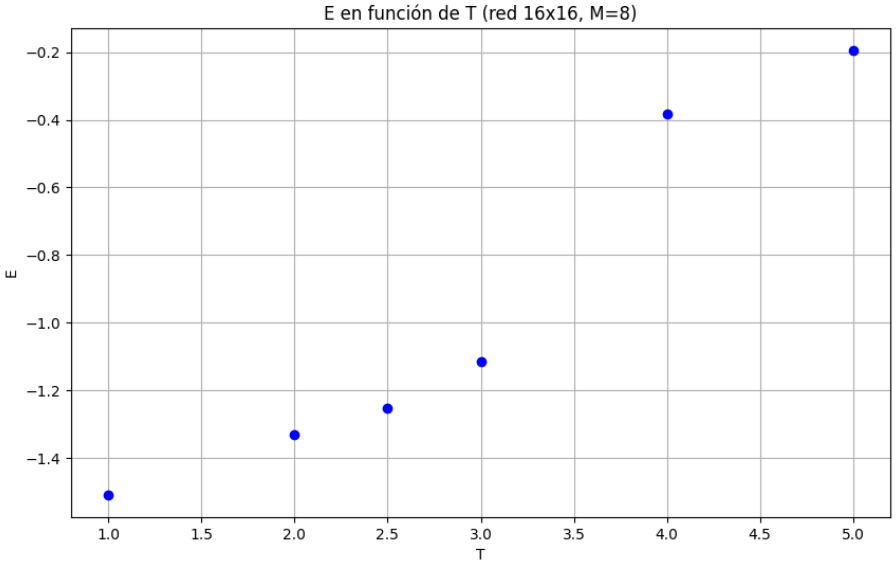


Figura 31: Energía en función de T para una red 16×16 y $M = 8$.

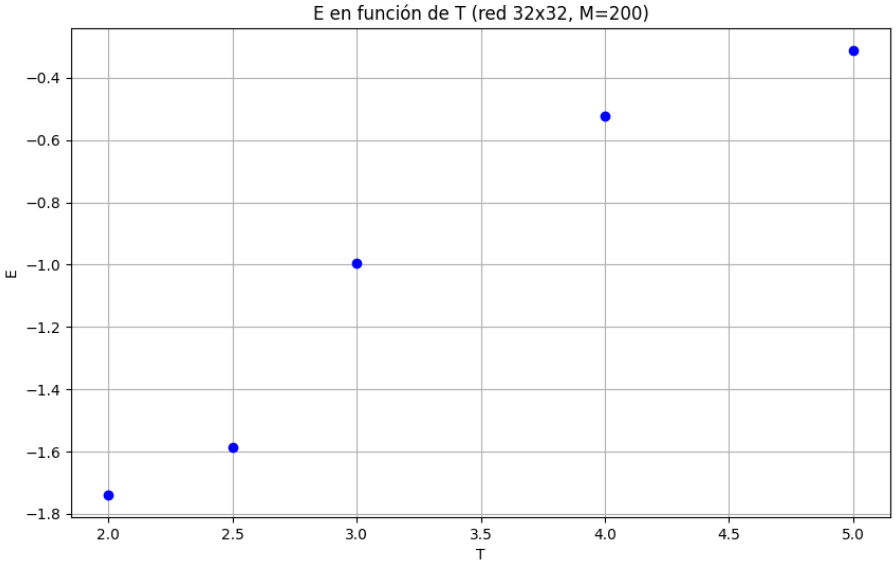


Figura 32: Energía en función de T para una red 32×32 y $M = 200$.

3.4. Apartado 5: Temperatura crítica

Para cada N se han realizado varias simulaciones a diferente temperatura y luego se han comparado los valores de susceptibilidad magnética. Se ha observado que el máximo se encuentra entre $T = 2{,}2$ y $T = 2{,}3$, por lo que se han hecho varias simulaciones en torno a esas temperaturas. Se estima la temperatura crítica T_c como aquella para la que la susceptibilidad magnética es máxima.

A continuación se recogen en tablas los valores que se han obtenido para dichos máximos.

- **Ap. 5 - Magnetización nula ($M = 0$):**

Tamaño de red	16×16	32×32	48×48
T_c	2.25	2.24	2.27

- **Ap. 5 - Magnetización no nula ($M \neq 0$):**

Tamaño de red	16×16	32×32	48×48
Magnetización	$M = 8$	$M = 200$	$M = 8$
T_c	2.23	2.22	2.27

Al comparar con los gráficos del apartado 2 se aprecia que, en efecto, la temperatura crítica se encuentra entre $T = 2$ y $T = 3$ aproximadamente.

En las tablas se observa que la temperatura crítica disminuye un poco y luego aumenta. Se han realizado varias tomas de datos y en todas se ha obtenido este comportamiento (disminución y luego aumento rápido). Tampoco se ha obtenido un comportamiento lineal al calcular la temperatura crítica usando el calor específico.

Referencias

[1] M. E. J. Newman, G. T. Barkema. *"Monte Carlo Methods in Statistical Physics"*.