Modelo de Ising Bidimensional

Félix Eduardo Rodríguez Lagonell
Mecánica estadística: PEC2
UNED

Resumen

En el presente trabajo se hace un estudio computacional del modelo de Ising bidimensional usando el algoritmo de Metrópolis bajo un campo magnético externo nulo. En particular se estudia la distribución espacial de espines al variar la temperatura y la correspondiente imanación

1. Introducción

Las simulaciones por ordenador juegan un papel muy importante en muchos campos de la física como la mecánica estadística. La razón de su importancia en la mecánica estadística radica en que muchos de sus modelos, incluso los que parecen más simples, son teóricamente intratables, por lo que es necesario recurrir a métodos computcionales que permitan encontrar buenas estimaciones.

De entre estos métodos, los denominados de Monte Carlo han demostrado ser muy potentes. Estos consisten en algoritmos que tratan de explorar el espacio de posibles soluciones de forma aleatoria pero controlada para, a partir de su comportamiento, obtener una solución en términos estadísticos.

Aplicaremos estos conceptos para hacer simulaciones del modelo de Ising, el cual es un modelo magnético en tanto en cuanto fue propuesto para explicar la transición de fase que existe en los materiales magnéticos (en particular materiales ferromagnéticos) al variar la temperatura. De hecho, fue el primer modelo que exhibía transición de fase y que fue resuelto de forma exacta por Lars Onsager [1, 2].

2. Modelo teórico

El punto de partida para un tratamiento cuántico del (anti)ferromagnetismo es el Hamiltoniano de Heisenberg

$$H = -J\sum_{i,j} s_i s_j - g\mu_B B\sum_i s_i,\ s_i = \pm 1 \qquad (1)$$

siendo g el factor de Landé, μ_B el magnetón de Bohr y J una constante de acoplamiento entre espines. El efecto del campo magnético B es el de favorecer que los espines se alineen en la dirección del campo para, de este modo, minimizar la energía. Nuestro estudio corresponde con $B = 0$ de tal forma que $H = -J\sum_{i,j} s_i s_j$.

Además, se sabe que el modelo de Ising sigue la distribución de Boltzmann, así el peso estadístico de los 2^N configuraciones de espín del sistema viene dada por la representación canónica

$$Z = \sum_{s_i} e^{-\beta H(s)}\ P(s) = \frac{e^{-H(s)}}{Z} \qquad (2)$$

con $\beta = (kT)^{-1}$. La imanación del sistema es

$$M = \frac{\partial log\,(Z)}{\partial \beta} = \sum_s P(s)\left(\sum_i s_i\right)_s \qquad (3)$$

donde el subíndice s indica que la suma sobre las variables de espín se realiza para cada configuración particular de los mismos. La energía del sistema es

$$E = \sum_s P(s)H(s) \tag{4}$$

La solución de Onsager muestra que el modelo presenta una transición de fase dada por

$$T_c = \frac{2J}{kln\left(\sqrt{2}+1\right)} \approx 2{,}269 \tag{5}$$

y una magnetización espontánea a temperaturas menores que T_c con valor

$$M = \left(1 - senh^{-4}\frac{2J}{kT}\right)^{\frac{1}{8}} \tag{6}$$

Por otro lado, se demuestra que la capacidad calorífica de la red cumple la relación

$$c_v = \frac{\beta^2}{N}\left(< E^2 > - < E >^2\right) \tag{7}$$

3. Procedimiento

3.1. Magnetización y transición de fase

Resulta una tarea complicada evaluar numéricamente el número de posibles estados para una red de NxN espines. Incluso resulta imposible para redes de tamaño modesto, por ejemplo N=20 tiene $2^{400} \sim 10^{120}$ configuraciones. Por esta razón haremos simulaciones de Monte Carlo para el estudio del modelo de Ising.

El algoritmo usado en este trabajo es el de Metrópolis [3], en el cual cambiamos el valor de un espín de la red elegido aleatoriamente si se cumplen ciertas condiciones. Este algortimo será implementado en Python y se encuentra en 6.3. El método seguido consiste en generar una red aleatoria bidemensional de tamaño NxN a baja temperatura con condiciones periódias de contorno, una vez que el sistema alcanza la estabilidad se aumenta la temperatura sucesivamente con el objetivo de estudiar la distribución de los espines y el comportamiento de la función de magnetización de la red. Este proceso es el resultado de tres posibles subprocesos que daría lugar a los mismos resultados:

1. Generar la red aleatoria a temperatura constante y esperar que alcance el equilibrio.
2. A partir del estado estable a bajas temperaturas evolucionar el sistema aumentando la tempertura.
3. A partir del estado estable a altas temperaturas evolucionar el sistema disminuyendo la tempertura.

3.2. Calor específico

Nos centraremos en calcular el calor específico para redes pequeñas (N=2,4,6) mediante análisis directo de las posibles energías y sus pesos estadísticos, pues no conllevan excesivo tiempo de computación. El objetivo es determinar la curva característica de esta variable y estudiar la tendencia a medida que aumentamos el número de espines. El algoritmo propuesto lo podemos ver en 6.1 y 6.2

4. Resultados

4.1. Magnetización y transición de fase

El modelo implementado predice satisfactoriamente una transición de fase entorno a la temperatura crítica T_c. En la figura 1 tenemos la evolución de una red de espines de tamaño 100x100. La imagen 1 de dicha figura corresponde al sistema en una situación metaestable a la temperatura $T_0 = 1$. El sistema de espines evoluciona al aumentar la temperatura en intervalos de $\triangle T = 0{,}2$ hasta la imagen 12 a temperatura $T_F = 3{,}8$ pasando por la temperatura crítica de transición de fase en la imagen 7. Aunque la transición de fase es apreciable, la presencia de estados metaestables, es decir, regiones estables de magnetización opuesta a las regiones colindantes (islas), difiere drásticamente de la solución analítica pues la magnetización (figura 2) debería ser máxima (todos los espines con la misma orientación) a bajas temperaturas.

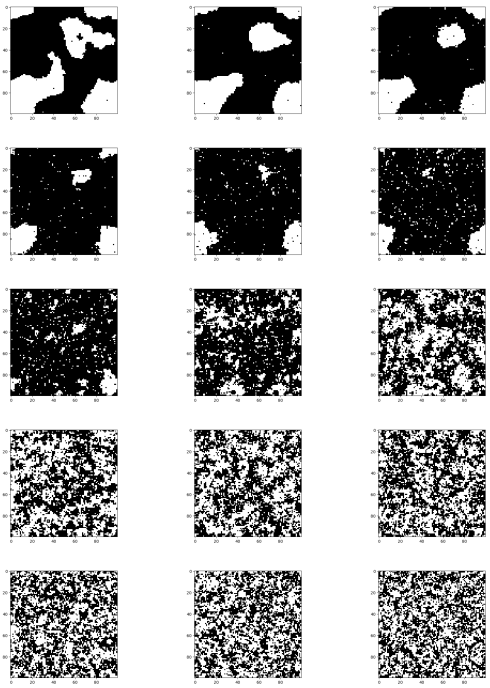

Figura 1: Distribución espines 100x100 a medida que aumenta la temperatura desde $T_1 = 1$ hasta $T_{12} = 3{,}8$ en intervalos de $\triangle T = 0{,}2$. Se aprecia la transición de fase entre la imagen 7 y la 8.

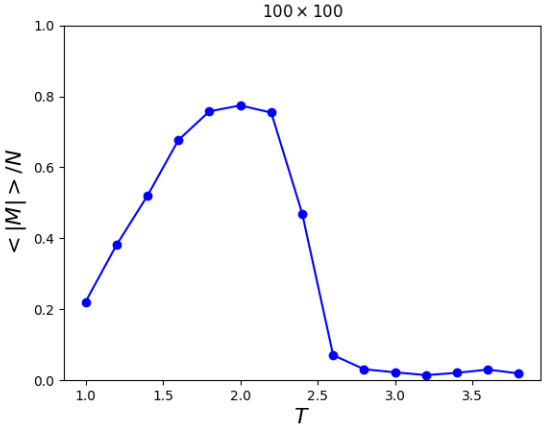

Figura 2: Magnetización espines 100x100 en función de la temperatura.

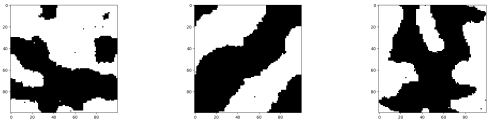

Figura 3: Distintas configuraciones metaestables 100x100

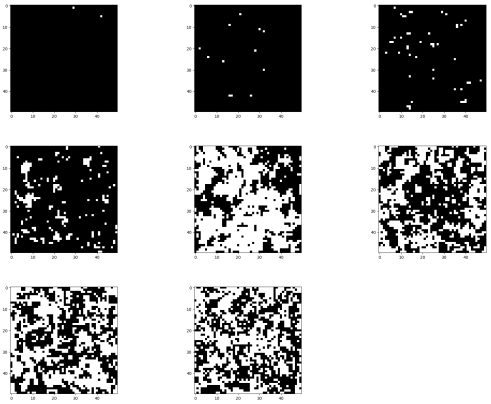

Figura 4: Distribución espines 50x50 a medida que aumenta la temperatura desde $T_1 = 1$ hasta $T_8 = 3{,}6$ en intervalos de $\triangle T = 0{,}4$. Se aprecia la transición de fase entre la imagen 4 a $T_4 = 2{,}2$ y la imagen 5 a $T_5 = 2{,}4$.

En la figura 3 presentamos distintas configuraciones de estos estados metaestables. El origen de esta situación radica en un tiempo insuficiente (insuficientes iteraciones del algoritmo) desde la configuración inicial aleatoria de espines (inicialización del algoritmo) hasta el estado de equilibrio a $T = 1$. Este razonamiento se hace patente con una red de 50x50 espines en el que sí se llevan a cabo suficientes iteraciones para llegar a la situación inicial estable con todos los espines orientados. Los resultados se muestran en la figura 4 y 5 donde podemos ver como el sistema parte inicialmente desde una configuración estable con magnetización máxima.

A su vez, a modo de comparación y para veri-

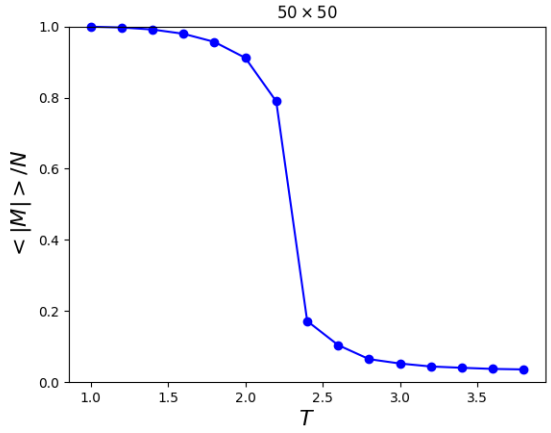

Figura 5: Magnetización espines 50x50 en función de la temperatura.

ficar que el algoritmo ofrece resultados certeros en correspondencia con la ecuación 6, hemos calculado la magnetización de redes pequeñas de espines. Los resultados se muestran en la figura 6. Para bajas temperaturas el sistema se mantiene magnetizado espontáneamente hasta que se acerca a la temperatura $T_c = 2{,}269$, a partir de la cual el sistema se desordena completamente resultando en un sistema cuya magnetización es nula. Comparando con las redes 50x50 y 100x100, la tendencia al aumentar el número de espines se refleja en una pendiente mayor cerca de dicho valor de temperatura crítica.

4.2. Calor específico

Tal como mencionamos en el capítulo anterior, calculamos el calor específico de varias redes pequeñas mediante la ecuación 7. Los resultados los podemos ver en la figura 7. Como es de esperar la función es creciente al aumentar la temperatura hasta llegar a un máximo absoluto dado por T_c a partir del cual decrece rápidamente cuánto mayor sea el número de espines de la red.

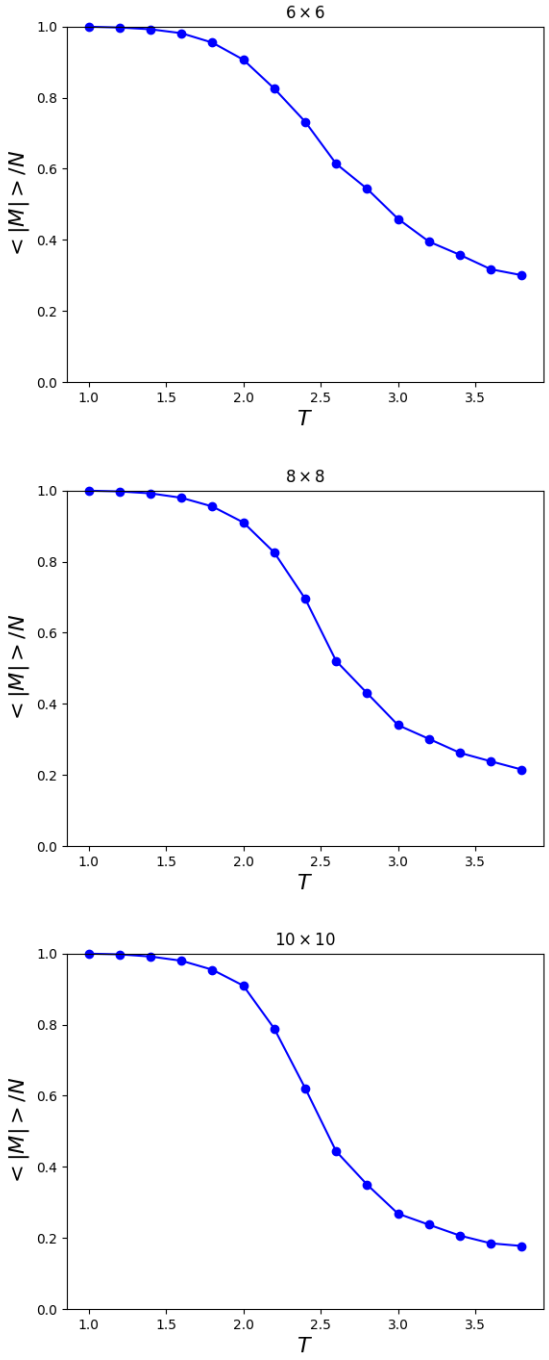

Figura 6: Magnetización en función de la temperatura 6x6, 8x8, 10x10

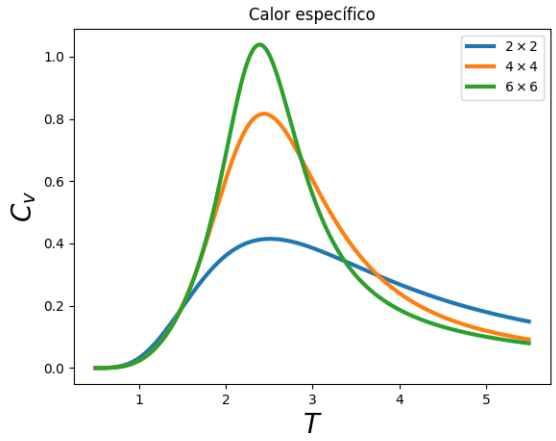

Figura 7: Calor específico en función de la temperatura para redes 2x2, 4x4, 6x6

5. Conclusión

El estudio del modelo de Ising usando el algoritmo de Metrópolis implementado en Python ha cumplido las expectativas para los comportamientos sujetos a estudio, a saber, la transición de fase y la ruptura de simetría espontánea de la magnetización por debajo del valor de temperatura crítica, que coincide con la solución analítica. Los resultados de magnetización y calor específico cumplieron con las estimaciones a través de los varios métodos utilizados, mientras que el resultado de la magnetización espontánea no fue satisfecha para redes mayores a 100x100. Se concluyó que esta desviación fue causada por una potencia informática insuficiente: el modelo no tuvo tiempo suficiente para alcanzar un estado verdaderamente estable. Aún así, una mayor potencia de cálculo junto con una posible mejora en el diseño e implementación en Python (o bien implementar otros algoritmos de Monte Carlo, por ejemplo el algoritmo de agrupamiento de Wolff) puede subsanar este inconveniente sin ningún tipo de problemas.

6. Apéndice

6.1. Algoritmo energías y pesos estadísticos

```
def gray_flip(t, N):
    k = t[0]
    if k > N: return t, k
    t[k - 1] = t[k]
    t[k] = k + 1
    if k != 1: t[0] = 1
    return t, k
L = 2 # Tamaño de arista. No poner L>6
N = L * L # Tablero de espines
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) %N,
            (i // L) * L + (i - 1) % L, (i - L) %N)
                for i in range(N)}
S = [-1] * N
E = -2 * N
dos = {}
dos[E] = 1
tau = list(range(1, N + 2))
for i in range(1, 2 ** N):
    tau, k = gray_flip(tau, N)
    h = sum(S[n] for n in nbr[k - 1])
    E += 2 * h * S[k - 1]
    S[k - 1] *= -1
    if E in dos: dos[E] += 1
    else: dos[E] = 1
for E in sorted(dos.keys()):
    print(E, dos[E])
#
# El resultado es la energía y el número de estados con dicha energía.
# Podemos guardar el resultado en un fichero .txt para luego leerlo con el siguiente algoritmo.
# Debemos guardar el .txt con el nombre "data_dos_Lx.txt." donde x será el tamaño de la arista L=x.
```

6.2. Algoritmo calor específico

```
import math, os
import matplotlib.pyplot as plt
# Calcularemos cv para 2x2, 4x4, 6x6 espines.
A = [2, 4, 6]
```

```
for L in A:
    N = L * L # Tablero de espines
    # Cargamos el fichero .txt con los resultados de energía
    filename = 'data_dos_L %i.txt' % L
    if os.path.isfile(filename):
        dos = {}
        f = open(filename, 'r')
        for line in f:
            E, N_E = line.split()
            dos[int(E)] = int(N_E)
        f.close()
    else:
        exit('input file missing')
    list_T = [0.5 + 0.01 * i for i in range(500)]
    list_cv = []
    for T in list_T:
        Z = 0.0
        E_av = 0.0
        M_av = 0.0
        E2_av = 0.0
        for E in dos.keys():
            weight = math.exp(- E / T) * dos[E]
            Z += weight
            E_av += weight * E
            E2_av += weight * E ** 2
        E2_av /= Z
        E_av /= Z
        cv = (E2_av - E_av ** 2) / N / T ** 2
        list_cv.append(cv)
    plt.hold(True)
    plt.title('Calor específico')
    plt.xlabel('$T$', fontsize=20)
    plt.ylabel('$C_v$', fontsize=20)
    plt.plot(list_T,list_cv,lw=3,label='$ %i\\times %i$' % (L, L))
    plt.legend()
    plt.savefig('plot.png')
```

6.3. Algoritmo magnetización y transición fase

```
import random, math, os, pylab
import numpy as np
import matplotlib.pyplot as plt
# Guardaremos los resultados en
# una carpeta llamada "snapshots"
output_dir = 'snapshots'
if not os.path.exists(output_dir): os.makedirs(output_dir)
# Tamaño de arista de la red.
L = 100
N = L * L # Tablero de espines
nbr = {i : ((i // L) * L + (i + 1) % L, (i + L) %N,
            (i // L) * L + (i - 1) % L, (i - L) % N) \
               for i in range(N)}
# Número de iteraciones.
# Podemos aumentar este valor
# a mdedida que disminuimos L
nsteps = 100*N
# Temperaturas
list_T = [1.0 + 0.2 * i for i in range(15)]
list_av_m = []
# Espines en rango aleatorio.
S = [random.choice([1, -1]) for k in range(N)]
A = [S[i*L:i*L+L] for i in range(L)]
X, Y = np.meshgrid(range(L), range(L))
X, Y = X+1, Y+1
fxy = A*(X/X)*(Y/Y)
plt.imshow(fxy,"binary")
plt.savefig(output_dir + '/' + 'snap_ %06i_T1.png' % 0)
M = sum(S)
for T in list_T:
    print('T =', T)
    beta = 1.0 / T
    M_tot = 0.0
    n_measures = 0
    for step in range(nsteps):
        k = random.randint(0, N - 1)
        delta_E = 2.0 * S[k] * sum(S[nn] for nn in nbr[k])
        if random.uniform(0.0, 1.0) < math.exp(-beta * delta_E):
            S[k] *= -1
            M += 2 * S[k]
        if step % N == 0 and step > nsteps / 2:
            M_tot += abs(M)
            n_measures += 1
    list_av_m.append(abs(M_tot) / float(n_measures * N))
```

```
        A = [S[i*L:i*L+L] for i in range(L)]
        X, Y = np.meshgrid(range(L), range(L))
        X, Y = X+1, Y+1
        fxy = A*(X/X)*(Y/Y)
        plt.imshow(fxy,"binary")
        plt.savefig(output_dir + '/' +
'snap_ %06i_T %.1f.png' % (nsteps,T))
    plt.close()
    pylab.title('$ %i\\times %i$' % (L, L))
    pylab.xlabel('$T$', fontsize=16)
    pylab.ylabel('$<|M|>/N$', fontsize=16)
    pylab.plot(list_T, list_av_m, 'bo-',
clip_on=False)
    pylab.ylim(0.0, 1.0)
    pylab.savefig(output_dir + '/' +
'magnetization_L %i.png' % L)
    print("End")
```

Referencias

[1] Statistical Mechanics, K. Huang, 1963

[2] Computational Physics, S. E. Koonin, 1986

[3] The Journal of Chemical Physics 21, Metropolis-Rosenbuth-Teller, 1953

[3] Mecánica Estadística, Abalo-Pacheco-Sánchez, UNED, 2008

[4] Física del Estado Sólido, Teoría y Métodos Numéricos, Francisco Domínguez-Adame, 2001

[6] Monte Carlo Methods in Statistical Physics, Newman-Barkema, 2001

[7] A guide to Monte Carlo Simulations in Statistical Physics, Landau-Binder, 2003