PROJEKT Wykorzystanie algorytmu Metropolisa w modelu Ising

Łukasz Siemieniec,Konrad Skutnik,Natalia Wójcik

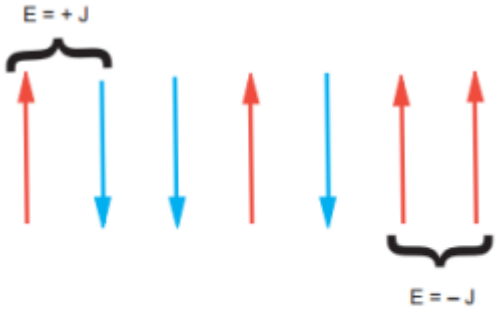
Styczeń 2022

Spis treści

1 Wstęp

Ferromagnetyki zawierają domeny o skończonej wielkości, w których spiny wszystkich atomów są skierowane w tym samym kierunku. Kiedy zewnętrzne pole magnetyczne jest przyłożone do tych materiałów, różne domeny są wyrównane i materiały stają się „namagnesowane". Jednak wraz ze wzrostem temperatury całkowity magnetyzm maleje, a w temperaturze Curie system przechodzi przez przejście fazowe, poza którym znika cała magnetyzacja. Naszym celem jest wyjaśnienie zachowania termicznego w przypadku magnesów.
Za model Isinga przyjmujemy N dipoli magnetycznych zamocowanych na ogniwach łańcucha liniowego. Energia interakcji między sąsiednimi parami to

$$s_i \equiv s_{z,i} = \pm\frac{1}{2}$$



Rysunek 1: Sieć 1-D N spinów użyta w modelu magnetyzmu Isinga. Energia interakcji między parami najbliższych sąsiadów E = ± J dla spinów wyrównanych i przeciwnych.

Ponieważ spin każdej cząstki może przyjąć jedną z dwóch wartości, istnieją 2N różne możliwe stany dla cząstek N w układzie. Zakładamy, że każdy dipol oddziałuje z zewnętrznym polem magnetycznym i jego najbliższym sąsiadem poprzez potencjał:

$$V_i = -Js_i \cdot s_{i+1} - g\mu_b s_i \cdot B$$

J - energia wymiany, miara siły interakcji spin-spin. Energia tego układu w stanie k jest wartością oczekiwaną sumy potencjału V nad spinami cząstek:

$$E_{\alpha_k} = \left\langle \alpha_k \middle| \sum_i V_i \middle| \alpha_k \right\rangle = -J \sum_{i=1}^{N-1} s_i s_{i+1} - B\mu_b \sum_{i=1}^{N} s_i$$

Mechanika statystyczna zaczyna się od elementarnych interakcji między cząstkami układu i konstruuje makroskopowe właściwości termodynamiczne, takie jak określone ciepło. Podstawowym założeniem jest, że wszystkie konfiguracje systemu zgodne z ograniczeniami są możliwe. Kiedy mówimy, że obiekt ma temperaturę T, mamy na myśli, że atomy obiektu są w równowadze termodynamicznej tak, że każdy atom ma średnią energię proporcjonalną do T. Chociaż może to być stan równowagi, jest to stan dynamiczny, w którym energia obiektu zmienia się, gdy wymienia on energię z otoczeniem. Energia Ej w kanonicznym zespole nie jest stała. Prawdopodobieństwa Ej podane przez rozkład Boltzmanna:

$$P(E_{\alpha j}, T) = \frac{e^{\frac{-E_{\alpha j}}{k_B T}}}{Z(T)}, \quad Z(T) = \sum_{\alpha_j} e^{\frac{-E_{\alpha j}}{k_B T}}$$

2 Algorytm Metropolis

Rosenbluth, Teller i Teller stworzyli algorytm usprawniający obliczanie średnich metodą Monte Carlo. Algorytm losowo zmienia poszczególne spiny tak, że średnie prawdopodobieństwo wystąpienia konfiguracji jest zgodne z rozkładem Boltzmana. Algorytm Metropolis jest połączeniem techniki redukcji wariancji i techniki odrzucenia von Neumanna. Algorytm będziemy realizować w kilku krokach. Rozpoczynamy od ustalonej temperatury i początkowej konfiguracji spinu, następnie zastosujemy algorytm aż do osiągnięcia równowagi termicznej. Dalsze stosowanie algorytmu generuje fluktuacje statystyczne dotyczące równowagi, z których wyprowadzimy wielkości termodynamiczne, takie jak namagnesowanie. Następuje zmienia temperatury, a cały proces powtarza się. Dokładność zredukowanych zależności temperaturowych dostarczy przekonujących dowodów na trafność algorytmu. Ponieważ możliwe konfiguracje 2N cząstek N mogą być bardzo dużą liczbą, potrzebny czas komputera może być bardzo długi.

3 Przedstawienie kodu projektu

```python
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt


def warpocz(n):
    spin = 2 * np.random.randint(2, size=(n, n)) - 1
    return spin


def metropolis(spin, beta):
    for i in range(n):
        for j in range(n):
            a = np.random.randint(0, n)
            b = np.random.randint(0, n)
            s = spin[a, b]
            nb = spin[(a + 1) % n, b] + spin[a, (b + 1) % n] + spin[(a - 1) % n, b] + spin[a, (b - 1) % n]
            cost = 2 * s * nb
            if cost < 0:
                s *= -1
            elif rand() < np.exp(-cost * beta):
                s *= -1
```

```
            spin[a, b] = s
    return spin


def energia(spin):
    energia = 0
    for i in range(len(spin)):
        for j in range(len(spin)):
            s = spin[i, j]
            nb = spin[(i + 1) % n, j] + spin[i, (j + 1) % n] + spin[(i - 1) % n, j] + spin[i, (j - 1) % n]
            energia += -nb * s
    return energia / 4


def magnetyzacja(spin):
    mag = np.sum(spin)
    return mag


if __name__ == '__main__':
    # parametry symulacji
    nt = 32  # temperatura
```

```
    n = 10  # wymiar siatki 10x10
    amPowt = 1000  # ilosc powtorzen

    T = np.linspace(1.53, 3.28, nt)
    E, M, C, X = np.zeros(nt), np.zeros(nt), np.zeros(nt), np.zeros(nt)
    n1, n2 = 1.0 / (amPowt * n * n), 1.0 / (amPowt * amPowt * n * n)

    for t in range(nt):
        E1 = M1 = E2 = M2 = 0
        spin = warpocz(n)
        iT = 1.0 / T[t]
        iT2 = iT * iT

        for i in range(amPowt):
            metropolis(spin, iT)

        for i in range(amPowt):
            metropolis(spin, iT)
            kalkEne = energia(spin)
            kalkMag = magnetyzacja(spin)
            E1 = E1 + kalkEne
```

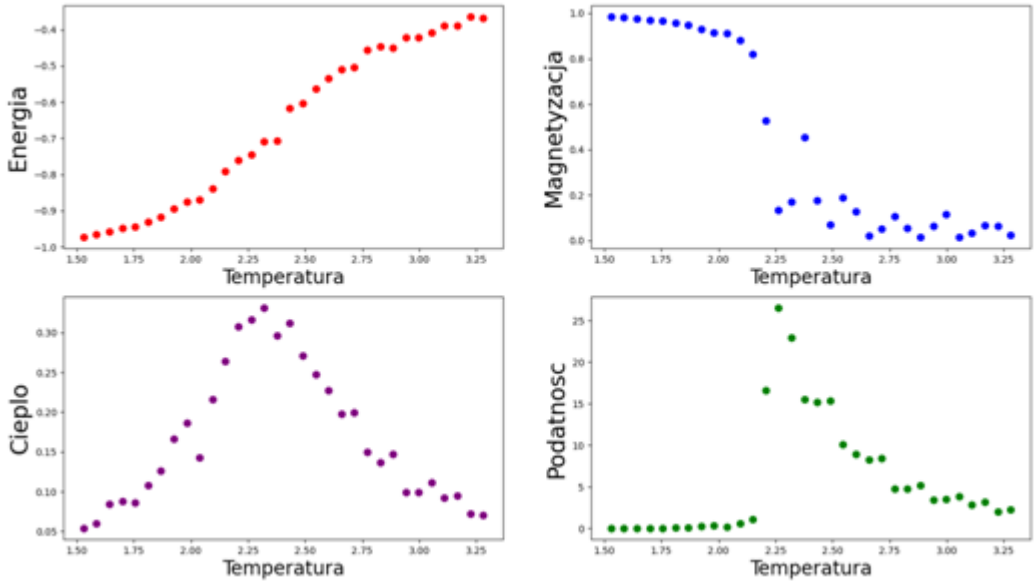
.

```
            M1 = M1 + kalkMag
            E2 = E2 + kalkEne * kalkEne
            M2 = M2 + kalkMag * kalkMag

        E[t] = n1 * E1
        M[t] = n1 * M1
        C[t] = (n1 * E2 - n2 * E1 * E1) * iT2
        X[t] = (n1 * M2 - n2 * M1 * M1) * iT

    # wykresy
    f = plt.figure(figsize=(18, 10))

    sp1 = f.add_subplot(2, 2, 1)
    plt.scatter(T, E, s=50, marker='o', color='red')
    plt.xlabel("Temperatura", fontsize=20)
    plt.ylabel("Energia", fontsize=24)
    plt.axis('tight')

    sp2 = f.add_subplot(2, 2, 2)
    plt.scatter(T, abs(M), s=50, marker='o',
color='Blue')
    plt.xlabel("Temperatura", fontsize=20)
    plt.ylabel("Magnetyzacja", fontsize=24)
```

```
    plt.axis('tight')

    sp3 = f.add_subplot(2, 2, 3)
    plt.scatter(T, C, s=50, marker='o', color='purple')
    plt.xlabel("Temperatura", fontsize=20)
    plt.ylabel("Ciepło", fontsize=24)
    plt.axis('tight')

    sp4 = f.add_subplot(2, 2, 4)
    plt.scatter(T, X, s=50, marker='o', color='green')
    plt.xlabel("Temperatura", fontsize=20)
    plt.ylabel("Podatnosc", fontsize=24)
    plt.axis('tight')
    plt.show()

```

.

4 Pokazanie otrzymanych wykresów za pomocą programu



5 Podsumowanie

Uzyskaliśmy cel naszego projektu wyjaśniając zachowanie termiczne w przypadku magnesów oraz zapoznając się z algorytmem Metropolis. Uzyskane wykresy są takie jakie powinnismy otrzymać.

6 Bibliografia

Na podstawie pliku MonteCarlo2.pdf