Magnesy i algorytm Metropolis

Weronika Smagór, Aleksandra Motor, Patryk Polczyk
Fizyka Techniczna III rok
Wydział Inżynierii Materiałowej i Fizyki Politechniki Krakowskiej

Luty 2021
Kraków

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 zadaniem jest wyjaśnienie zachowania termicznego w przypadku magnesów.

1.1 Model Isinga

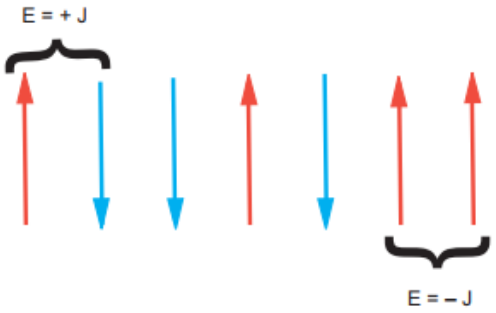
Jako nasz model bierzemy pod uwagę dipole magnetyczne zamocowane w miejscu na ogniwach łańcucha liniowego. Zakładamy, że cząstka w miejscu „i" ma spin si, który jest skierowany w górę albo w dół:

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

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:

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

Tutaj stała J nazywana jest energią wymiany i jest miarą siły interakcji spin-spin.



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.

Energia tego układu w stanie α_k jest wartością oczekiwaną sumy potencjału V nad spinami cząstek:

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

1.2 Mechanika statystyczna

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 $E_{\alpha j}$ 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 wymyślili 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.

Zgodnie z omówioną teorią oraz instrukcjami zaimplementowaliśmy program ising.py.

ising.py

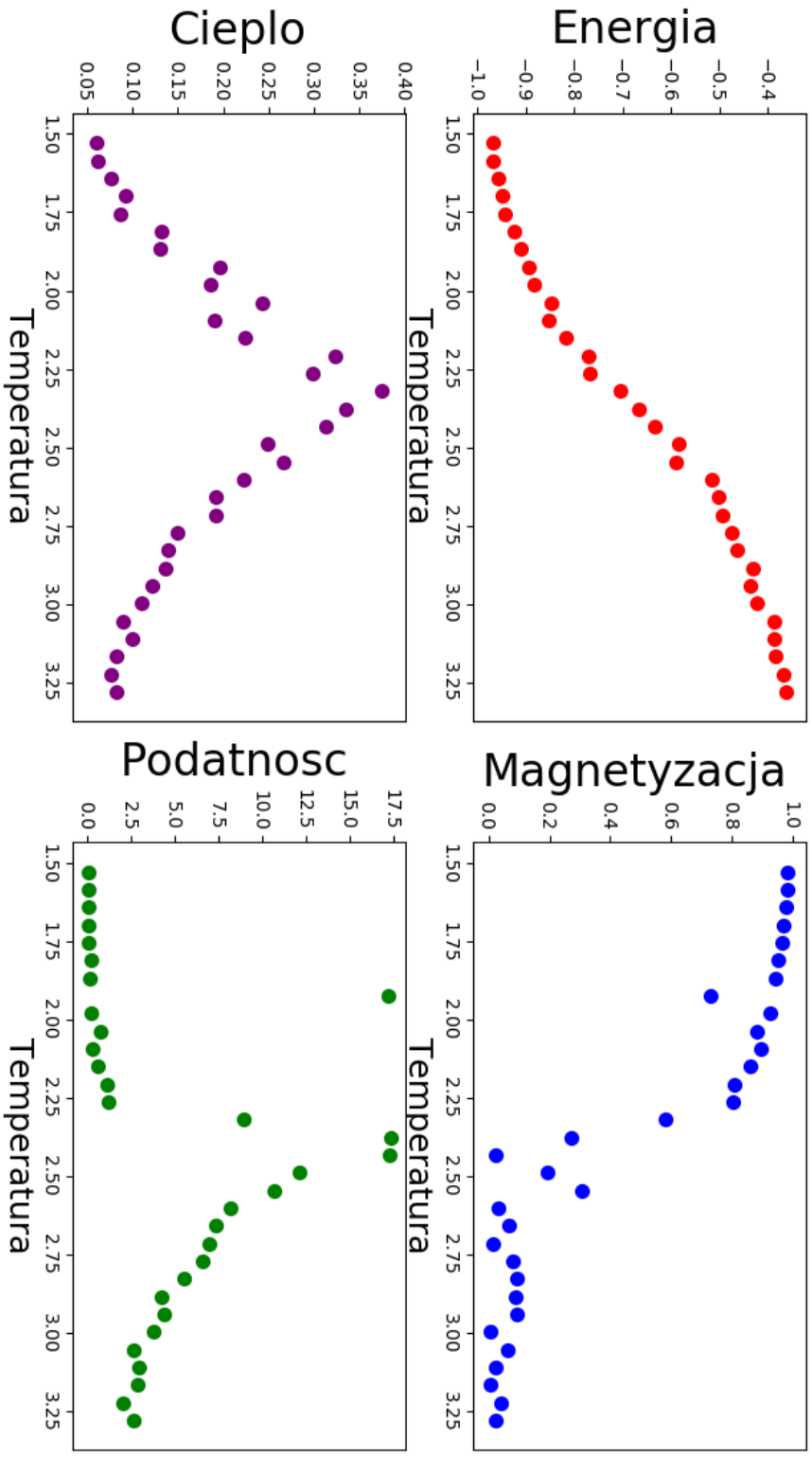
```python
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

def warPocz(N):
    """generator konfiguracji spinów dla warunków początkowych"""
    spin = 2 * np.random.randint(2, size=(N,N)) - 1
    return spin

def metropolis(spin, beta):
    """Algorytm Metropolisa"""
    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         #liczba punktow pomiarowych dla temperatury
    N       = 10         #wymiar siatki 10x10
    amPowt  = 1000       #ilosc powtorzen algorytmu Metropolisa
```

ising.py

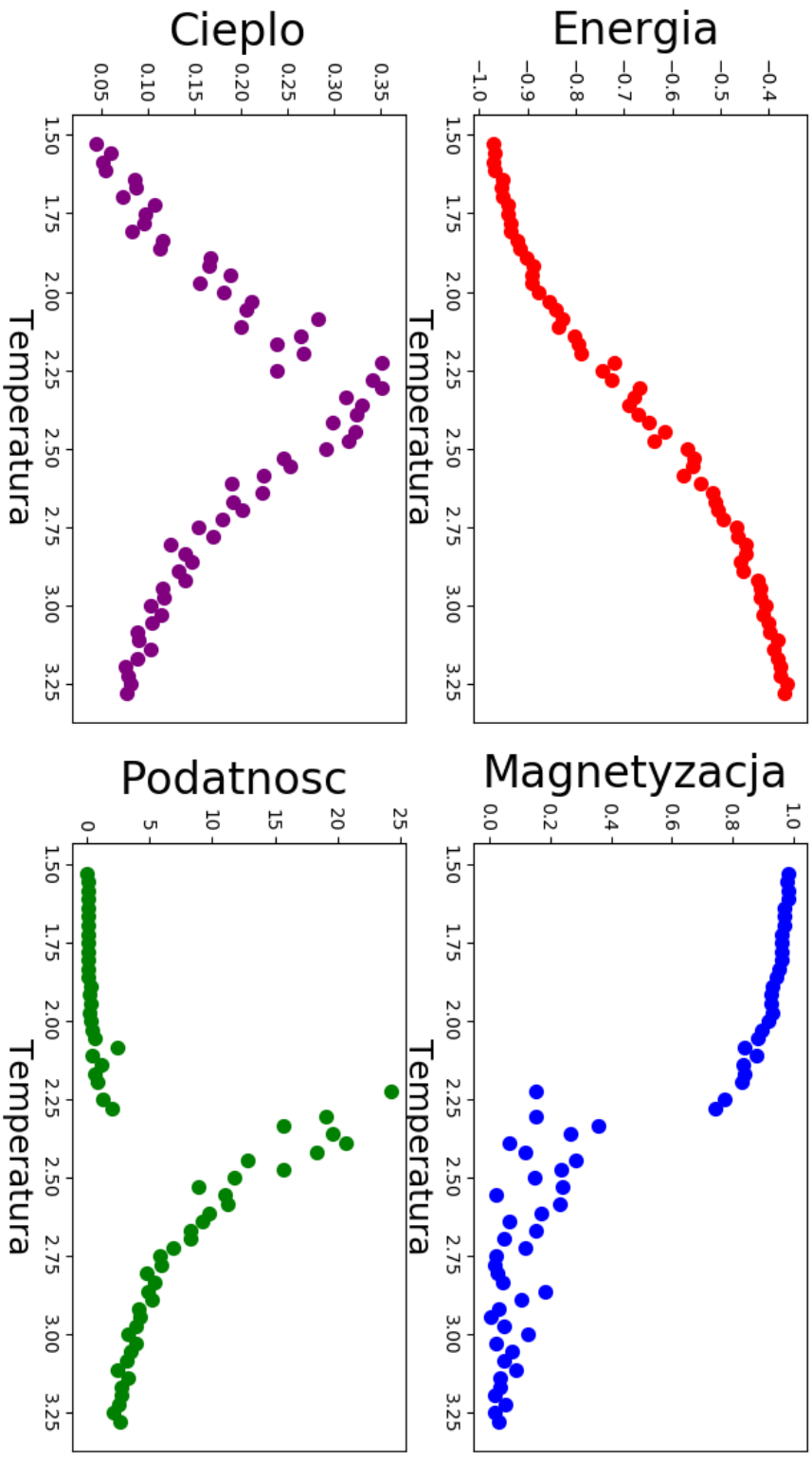
```
44
45      T = np.linspace(1.53, 3.28, nt);
46      E, M, C, X = np.zeros(nt), np.zeros(nt), np.zeros(nt), np.zeros(nt)
47      n1, n2 = 1.0/(amPowt*N*N), 1.0/(amPowt*amPowt*N*N)
48
49      for t in range(nt):
50          E1 = M1 = E2 = M2 = 0
51          spin = warPocz(N)
52          iT = 1.0/T[t]
53          iT2 = iT*iT
54
55          for i in range(amPowt):
56              metropolis(spin, iT)
57
58          for i in range(amPowt):
59              metropolis(spin, iT)
60              kalkEne = energia(spin)
61              kalkMag = magnetyzacja(spin)
62              E1 = E1 + kalkEne
63              M1 = M1 + kalkMag
64              E2 = E2 + kalkEne * kalkEne
65              M2 = M2 + kalkMag * kalkMag
66
67          E[t] = n1 * E1
68          M[t] = n1 * M1
69          C[t] = (n1 * E2 - n2 * E1 * E1) * iT2
70          X[t] = (n1 * M2 - n2 * M1 * M1) * iT
71
72  # wykresy
73      f = plt.figure(figsize=(18, 10));
74
75      sp = f.add_subplot(2, 2, 1);
76      plt.scatter(T, E, s=50, marker='o', color='red')
77      plt.xlabel("Temperatura", fontsize=20);
78      plt.ylabel("Energia", fontsize=24);
79      plt.axis('tight')
80
81      sp = f.add_subplot(2, 2, 2);
82      plt.scatter(T, abs(M), s=50, marker='o', color='Blue')
83      plt.xlabel("Temperatura", fontsize=20);
84      plt.ylabel("Magnetyzacja", fontsize=24);
85      plt.axis('tight')
86
87      sp = f.add_subplot(2, 2, 3);
88      plt.scatter(T, C, s=50, marker='o', color='purple')
89      plt.xlabel("Temperatura", fontsize=20);
90      plt.ylabel("Cieplo", fontsize=24);
91      plt.axis('tight')
92
93      sp = f.add_subplot(2, 2, 4);
94      plt.scatter(T, X, s=50, marker='o', color='green')
95      plt.xlabel("Temperatura", fontsize=20);
96      plt.ylabel("Podatnosc",fontsize=24);
97      plt.axis('tight')
98      plt.show()
```

3 Wyniki

W celu porównania dokładności przeprowadziliśmy symulację dla 32 oraz 64 punktów pomiarowych dla temperatury.



Rysunek 2: Wyniki otrzymane za pomocą programu ising.py dla 32 punktow pomiarowych



Rysunek 3: Wyniki otrzymane za pomocą programu ising.py dla 64 punktów pomiarowych

4 Podsumowanie

Osiągnęliśmy cel naszego projektu wyjaśniając zachowanie termiczne w przypadku magnesów oraz zapoznając się z algorytmem Metropolis. Zaimplementowany przez nas program ising.py został dołączony do raportu, natomiast wyniki jego działania zaprezentowane zostały na rysunkach 2 oraz 3.

Literatura

[1] A Survey of Computational Physics: Introductory Computational Science, Rubin H. Landau, José Páez, Cristian C. Bordeianu, Princeton University Press 2011.

[2] `https://pl.wikipedia.org/wiki/Ferromagnetyzm`

[3] `https://pl.wikipedia.org/wiki/Model_Isinga`