自発的対称性の破れの条件

Shinya Koji

2020 年 1 月 31 日

1　背景

鉄などの磁石になれる強磁性体は，ある臨界温度 T_c 以下まで冷却していくと，外部磁場がゼロであっても内部のスピンの向きがある一方向に揃いはじめ絶対零度で完全に揃うことが知られている．これを自発的対称性の破れと呼んでいる．空間とは本来等方的であるため，特に外部磁場が無い状態では空間において特別な方向というのは存在しない．すなわち外部磁場が存在しなければ空間的対称性は常に保たれるはずである．しかし，強磁性体においては何らかの理由によりこの対称性が破れ，磁化が現れる．

本稿では，自発的対称性の破れの原因を強磁性体内部のスピン相互作用に求め，シミュレーション計算を用いてスピン間の相互作用が満たすべき条件を探り，それについて考察した．

イジング模型

本シミュレーションでは 2 次元イジング模型を用いた．
これをシュミレーションで表現するために要素が 0 または 1 である $n \times n$ の 2 次元配列を用意した．例えば，上向きスピンを 1，下向きスピンを 0 と対応付けすることもできる．しかし，今回は外部磁場がゼロであるため対応付けは特段重要ではない．

系のそれぞれのスピン σ_i $(i = 1, 2, \ldots, N)$ を

$$\sigma_i = \pm 1 \tag{1}$$

とする．また系のスピン変数の組を $\{\sigma\} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ と定義する．
このとき，ハミルトニアン $H(\{\sigma\})$ は $J > 0$ として

$$H(\{\sigma\}) = -J \sum_{<i,j>} \sigma_i \sigma_j - h \sum_i \sigma_i \tag{2}$$

で与えられる．ここに，h は外部磁場である．また，$< i, j >$ についての和は，系の最隣接スピンの組全てに対して重複なく和をとる．
$h = 0$ のとき，

$$H(\{\sigma\}) = -J \sum_{<i,j>} \sigma_i \sigma_j \tag{3}$$

となる．スピン i, j が同じ向きを向いているときエネルギーは小さくなる．容易に分かる通り，系のすべてのスピンがある同じ一方向を向いているとき系のエネルギーは最小となる．
これより，分配関数 Z は

$$Z = \sum_{\{\sigma\}} \exp(-\beta H\{\sigma\}) = \sum_{\{\sigma\}} \exp(\beta J \sum_{<i,j>} \sigma_i \sigma_j) \tag{4}$$

となる．

2　手法 1

以下の環境のもとでシミュレーションを実行した．

- OS：Ubuntu18.04
- CPU：メーカー不詳, 速度 1.5GHz 程度
- メモリ：12GB
- 言語：Python3.6

相互作用の規約について

目的は，T_c 以下の温度において自発的対称性の破れが観測される条件を見つけることであった．よって，$T > T_c$ において対称性が破れていない状態を初期状態とし，$T < T_c$ で温度を下げていくことに対応してエネルギーを下げていくような振舞いをさせた．(3) より，エネルギーを下げるのは周りのスピンの向きに自身の向きを揃えるような振舞いである．

以上のことから，考えられる相互作用の規約として以下のものを取り上げた．ここで，2 次元イジング模型では, あるスピンを囲むスピンは上下左右の 4 つであることに注意されたい．斜めのスピンは格子上で繋がっていないものとし，互いに相互作用を及ぼさないとした．

また，以下で s という量を用いた．あるスピン $\sigma_{i,t}$(t は遷移段階を示す変数) について，それと隣接するスピン j ($j = 1, 2, 3, 4$) に対して

$$s_{i,t} = \sum_j \sigma_{j,t} \tag{5}$$

と定義した．また，シミュレーション上の都合により (1) に変わって

$$\sigma_{i,t} = 1 \text{ or } 0 \tag{6}$$

とした．このようにスピン変数の値を変えても，本稿ではこれを用いて物理量を計算するわけではないため，結論には何ら影響ない．

規約 1 : ステイ

$$\begin{cases} s_{i,t} > 2 & \Rightarrow \sigma_{i,t+1} = 1 \\ s_{i,t} < 2 & \Rightarrow \sigma_{i,t+1} = 0 \\ s_{i,t} = 2 & \Rightarrow \sigma_{i,t+1} = \sigma_{i,t} \end{cases} \tag{7}$$

境界端 (2 次元系の 4 隅)，及び境界辺 (端を除く境界の 4 辺) の挙動もこれと同様に，周りのスピンに合わせるような振舞いをするようにした (詳細は以下のソースコードを参照されたい).

規約 2 : リバース

$$\begin{cases} s_{i,t} > 2 & \Rightarrow \sigma_{i,t+1} = 1 \\ s_{i,t} < 2 & \Rightarrow \sigma_{i,t+1} = 0 \\ s_{i,t} = 2 & \Rightarrow \sigma_{i,t+1} = \overline{\sigma}_{i,t} \end{cases} \tag{8}$$

ここで

$$\overline{\sigma}_{i,t} \quad = \begin{cases} 1, & \text{if} \quad \sigma_{i,t} = 0 \\ 0, & \text{if} \quad \sigma_{i,t} = 1 \end{cases} \tag{9}$$

とした．境界端 (2 次元系の 4 隅)，及び境界辺 (端を除く境界の 4 辺) の挙動もこれと同様に，周りのスピンに合わせるような振舞いをするようにした．

規約 3 : ランダム

$$\begin{cases} s_{i,t} > 2 & \Rightarrow \sigma_{i,t+1} = 1 \\ s_{i,t} < 2 & \Rightarrow \sigma_{i,t+1} = 0 \\ s_{i,t} = 2 & \Rightarrow \sigma_{i,t+1} = 1 \text{ or } 0 \end{cases} \tag{10}$$

$s_{i,t} = 2$ のとき，$\sigma_{i,t+1}$ が 1 または 0 となる確率はともに 50% とした.
境界端 (2 次元系の 4 隅)，及び境界辺 (端を除く境界の 4 辺) の挙動もこれと同様に，周りのスピンに合わせるような振舞いをするようにした.

ソースコード

今回用いた規約 1 の Python コードを以下に示す．規約 2, 規約 3 についても周りのスピンが半々のときの振舞いの部分を変更すればよい.

```
import numpy as np
path = './data_s.dat'

n = 100  # 粒子数はn^2

mg = [0 for i in range(10)]

# 初期状態
mg[0] = np.random.randint(0, 2, (n, n))
mg[1] = np.random.randint(0, 2, (n, n))

n1 = 0
n0 = 0
for i in np.arange(0, n):
    for j in np.arange(0, n):
        mg[1][i][j] = mg[0][i][j]
        if mg[0][i][j] == 1:
            n1 += 1
        elif mg[0][i][j] == 0:
            n0 += 1
text = str(n1 / (n * n)) + "␣" + str(n0 / (n * n)) + "\n"
with open(path, mode='a') as f:
    f.write(text)
print(mg[0])
print("\n")

for k in range(20):
    for m in [0,1]:
        n1 = 0
        n0 = 0
        # 内部の振舞い
        for i in np.arange(1, n - 1):
```

```
    for j in np.arange(1, n - 1):
        s = mg[m][i][j - 1] + mg[m][i][j + 1] + mg[m][i - 1][j] + mg[m][i
            + 1][j]
        if s > 2:
            mg[(m + 1) % 2][i][j] = 1
            n1 += 1
        elif s < 2:
            mg[(m + 1) % 2][i][j] = 0
            n0 += 1
        elif s == 2:
            if mg[m][i][j] == 1:
                mg[(m + 1) % 2][i][j] = 1
                n1 += 1
            elif mg[m][i][j] == 0:
                mg[(m + 1) % 2][i][j] = 0
                n0 += 1
# 境界端の振舞い
# 左上
s = mg[m][1][0] + mg[m][0][1]
if s == 2:
    mg[(m + 1) % 2][0][0] = 1
    n1 += 1
elif s == 0:
    mg[(m + 1) % 2][0][0] = 0
    n0 += 1
elif s == 1:
    if mg[m][0][0] == 1:
        mg[(m + 1) % 2][0][0] = 1
        n1 += 1
    elif mg[m][0][0] == 0:
        mg[(m + 1) % 2][0][0] = 0
        n0 += 1
# 右上
s = mg[m][0][n -2] + mg[m][1][n - 1]
if s == 2:
    mg[(m + 1) % 2][0][n -1] = 1
    n1 += 1
elif s == 0:
    mg[(m + 1) % 2][0][n - 1] = 0
    n0 += 1
elif s == 1:
    if mg[m][0][n - 1] == 1:
        mg[(m + 1) % 2][0][n - 1] = 1
        n1 += 1
    elif mg[m][0][n - 1] == 0:
        mg[(m + 1) % 2][0][n - 1] = 0
        n0 += 1
# 左下
```

```
    s = mg[m][n - 2][0] + mg[m][n - 1][1]
    if s == 2:
        mg[(m + 1) % 2][n - 1][0] = 1
        n1 += 1
    elif s == 0:
        mg[(m + 1) % 2][n - 1][0] = 0
        n0 += 1
    elif s == 1:
        if mg[m][n - 1][0] == 1:
            mg[(m + 1) % 2][n - 1][0] = 1
            n1 += 1
        elif mg[m][n - 1][0] == 0:
            mg[(m + 1) % 2][n - 1][0] = 0
            n0 += 1
    # 右下
    s = mg[m][n - 1][n - 2] + mg[m][n - 2][n - 1]
    if s == 2:
        mg[(m + 1) % 2][n - 1][n - 1] = 1
        n1 += 1
    elif s == 0:
        mg[(m + 1) % 2][n - 1][n - 1] = 0
        n0 += 1
    elif s == 1:
        if mg[m][n - 1][n - 1] == 1:
            mg[(m + 1) % 2][n - 1][n - 1] = 1
            n1 += 1
        elif mg[m][n - 1][n - 1] == 0:
            mg[(m + 1) % 2][n - 1][n - 1] = 0
            n0 += 1
    # 境界辺の振舞い
    for i in [0, n - 1]:
        if i == 0:
            for j in np.arange(1, n - 1):
                s = mg[m][i][j - 1] + mg[m][i][j + 1] + mg[m][i + 1][j]
                if s >= 2:
                    mg[(m + 1) % 2][i][j] = 1
                    n1 += 1
                elif s < 2:
                    mg[(m + 1) % 2][i][j] = 0
                    n0 += 1

                s = mg[m][j - 1][i] + mg[m][j + 1][i] + mg[m][j][i + 1]
                if s >= 2:
                    mg[(m + 1) % 2][j][i] = 1
                    n1 += 1
                elif s < 2:
                    mg[(m + 1) % 2][j][i] = 0
                    n0 += 1
```

```
    elif i == n - 1:
        for j in np.arange(1, n - 1):
            s = mg[m][i][j - 1] + mg[m][i][j + 1] + mg[m][i - 1][j]
            if s >= 2:
                mg[(m + 1) % 2][i][j] = 1
                n1 += 1
            elif s < 2:
                mg[(m + 1) % 2][i][j] = 0
                n0 += 1

            s = mg[m][j - 1][i] + mg[m][j + 1][i] + mg[m][j][i - 1]
            if s >= 2:
                mg[(m + 1) % 2][j][i] = 1
                n1 += 1
            elif s < 2:
                mg[(m + 1) % 2][j][i] = 0
                n0 += 1

# 書き込み
text = str(n1 / (n * n)) + "␣" + str(n0 / (n * n)) + "\n"
with open(path, mode='a') as f:
    f.write(text)

print(mg[m])
print("\n")
```

3 結果 1

図 1，図 2, 図 3 はそれぞれ規約 1，規約 2, 規約 3 のもとでのシュミレーションの結果である．
縦軸 ratio はスピン 1,0 の全体に対する割合を示す．横軸 t は，遷移段階を示す．
自身を周りのスピンの向きに合わせ，エネルギーを小さくする振舞いを系全体で繰り返せば，いずれすべてのスピンの向きが揃い，対称性が破れそうだという直感に反して，規約 1，規約 2 のもとでは割合はほぼ半々のままであった．つまり，規約 1, 規約 2 の相互作用のもとでは自発的対称性の破れは起こらず，エネルギーは十分に小さくならない．
一方，規約 3 では若干対称性が破れた．これは $s_{i,t} = 2$ の場合のゆらぎが重要であることを示している．
しかし，破れは十分ではなく，図 3 の横軸のスケールからわかるように，かなり遅いペースで起きた．

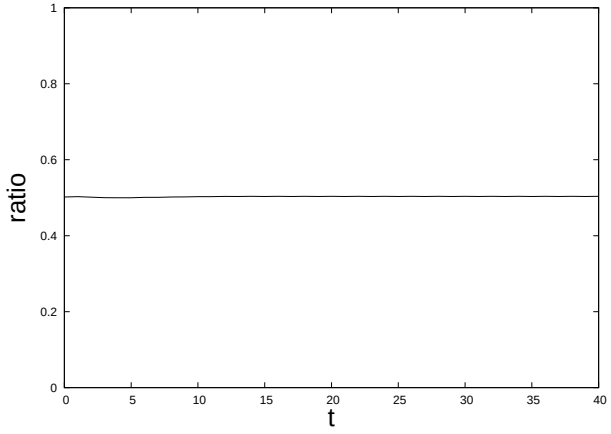

図１　規約１：ステイ

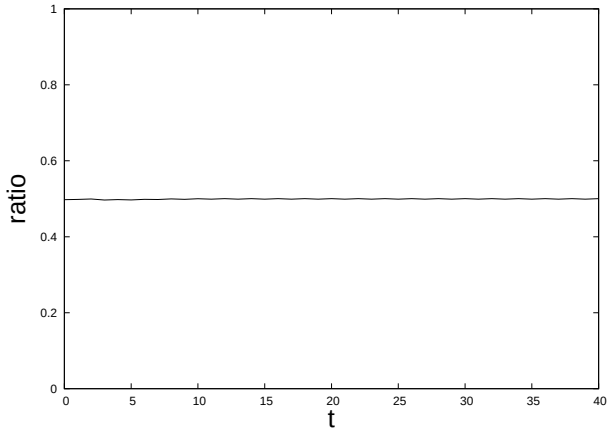

図２　規約２：リバース

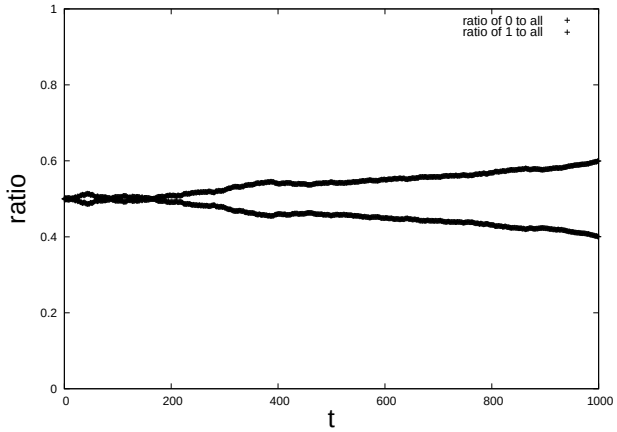

図３　規約３：ランダム

4　手法 2

規約 1, 規約 2, 規約 3 では直感に反して自発的対称性の破れが十分に起こらなかったため，平均場 m を導入した．
シミュレーションでは, これを

$$m_t = a\frac{N_{1,t} - N_{0,t}}{N} \tag{11}$$

で定義した．ここで $N_{1,t}, N_{0,t}$ はそれぞれ t におけるスピン 1,0 の総数である．また，a は平均場の重みである．本稿では常に $a = 1$ としたが，コードでは変更が用意にできるようにした．$-a < m_t < a$ である．

平均場 m を導入したので (5) を改めて

$$s_{i,t} = \sum_j \sigma_{j,t} + m_t \tag{12}$$

とした．
以上を踏まえて，規約 1, 規約 2, 規約 3 はそのままにシミュレーションを実行した．但し，平均場を導入したことに付随して，境界辺の振舞いを少し変更した．詳細はコードを見比べて確認されたい．

ソースコード

以下に平均場を導入した Python コードを示す.

```
import numpy as np

# 書き込みデータのパス
path = './data_mf_s.dat'

n = 100  # 全粒子数はn^2

# 初期状態を定義
mg = [0 for i in range(2)]
mg[0] = np.random.randint(0, 2, (n, n))
mg[1] = np.random.randint(0, 2, (n, n))
n1 = 0
n0 = 0
for i in np.arange(0, n):
    for j in np.arange(0, n):
        mg[1][i][j] = mg[0][i][j]
        if mg[0][i][j] == 1:
            n1 += 1
        elif mg[0][i][j] == 0:
            n0 += 1
text = str(n1 / (n * n)) + " " + str(n0 / (n * n)) + "\n"
with open(path, mode='a') as f:
    f.write(text)

# 状態をターミナルへ出力
```

```
print(mg[0])
print("\n")

# 平均場の重み
a = 1

# 系の遷移
for k in range(5):
    for m in [0,1]:
        # 平均場の導入
        mf = a * (n1 - n0) / (n * n)

        n1 = 0
        n0 = 0
        # 内部の振舞い
        for i in np.arange(1, n - 1):
            for j in np.arange(1, n - 1):
                s = mg[m][i][j - 1] + mg[m][i][j + 1] + mg[m][i - 1][j] + mg[m][i
                    + 1][j] + mf
                if s > 2:
                    mg[(m + 1) % 2][i][j] = 1
                    n1 += 1
                elif s < 2:
                    mg[(m + 1) % 2][i][j] = 0
                    n0 += 1
                elif s == 2:
                    if mg[m][i][j] == 1:
                        mg[(m + 1) % 2][i][j] = 1
                        n1 += 1
                    elif mg[m][i][j] == 0:
                        mg[(m + 1) % 2][i][j] = 0
                        n0 += 1
        # 境界端の振舞い
        # 左上
        s = mg[m][1][0] + mg[m][0][1] + mf
        if s > 1:
            mg[(m + 1) % 2][0][0] = 1
            n1 += 1
        elif s < 1:
            mg[(m + 1) % 2][0][0] = 0
            n0 += 1
        elif s == 1:
            if mg[m][0][0] == 1:
                mg[(m + 1) % 2][0][0] = 1
                n1 += 1
            elif mg[m][0][0] == 0:
                mg[(m + 1) % 2][0][0] = 0
                n0 += 1
```

```
# 右上
s = mg[m][0][n -2] + mg[m][1][n - 1] + mf
if s > 1:
    mg[(m + 1) % 2][0][n -1] = 1
    n1 += 1
elif s < 1:
    mg[(m + 1) % 2][0][n - 1] = 0
    n0 += 1
elif s == 1:
    if mg[m][0][n - 1] == 1:
        mg[(m + 1) % 2][0][n - 1] = 1
        n1 += 1
    elif mg[m][0][n - 1] == 0:
        mg[(m + 1) % 2][0][n - 1] = 0
        n0 += 1
# 左下
s = mg[m][n - 2][0] + mg[m][n - 1][1] + mf
if s > 1:
    mg[(m + 1) % 2][n - 1][0] = 1
    n1 += 1
elif s < 1:
    mg[(m + 1) % 2][n - 1][0] = 0
    n0 += 1
elif s == 1:
    if mg[m][n - 1][0] == 1:
        mg[(m + 1) % 2][n - 1][0] = 1
        n1 += 1
    elif mg[m][n - 1][0] == 0:
        mg[(m + 1) % 2][n - 1][0] = 0
        n0 += 1
# 右下
s = mg[m][n - 1][n - 2] + mg[m][n - 2][n - 1] + mf
if s > 1:
    mg[(m + 1) % 2][n - 1][n - 1] = 1
    n1 += 1
elif s < 1:
    mg[(m + 1) % 2][n - 1][n - 1] = 0
    n0 += 1
elif s == 1:
    if mg[m][n - 1][n - 1] == 1:
        mg[(m + 1) % 2][n - 1][n - 1] = 1
        n1 += 1
    elif mg[m][n - 1][n - 1] == 0:
        mg[(m + 1) % 2][n - 1][n - 1] = 0
        n0 += 1
# 境界辺の振舞い
for i in [0, n - 1]:
    if i == 0:
```

```
        for j in np.arange(1, n - 1):
            s = mg[m][i][j - 1] + mg[m][i][j + 1] + mg[m][i + 1][j] + mf
            if s >= 1.5:
                mg[(m + 1) % 2][i][j] = 1
                n1 += 1
            elif s < 1.5:
                mg[(m + 1) % 2][i][j] = 0
                n0 += 1

            s = mg[m][j - 1][i] + mg[m][j + 1][i] + mg[m][j][i + 1] + mf
            if s >= 1.5:
                mg[(m + 1) % 2][j][i] = 1
                n1 += 1
            elif s < 1.5:
                mg[(m + 1) % 2][j][i] = 0
                n0 += 1

    elif i == n - 1:
        for j in np.arange(1, n - 1):
            s = mg[m][i][j - 1] + mg[m][i][j + 1] + mg[m][i - 1][j] + mf
            if s >= 1.5:
                mg[(m + 1) % 2][i][j] = 1
                n1 += 1
            elif s < 1.5:
                mg[(m + 1) % 2][i][j] = 0
                n0 += 1

            s = mg[m][j - 1][i] + mg[m][j + 1][i] + mg[m][j][i - 1] + mf
            if s >= 1.5:
                mg[(m + 1) % 2][j][i] = 1
                n1 += 1
            elif s < 1.5:
                mg[(m + 1) % 2][j][i] = 0
                n0 += 1

# 書き込み
text = str(n1 / (n * n)) + "␣" + str(n0 / (n * n)) + "\n"
with open(path, mode='a') as f:
    f.write(text)

# 状態をターミナルへ出力
print(mg[(m + 1) % 2])
print("\n")
```

5　結果 2

図 4 は平均場を導入した規約 1 の相互作用のもとでの系の振舞いである．規約 2, 規約 3 においても同様の結果となった．

図 3 のゆらぎの影響のみでは非常に遅くて不十分だった対称性の破れが，図 4 では平均場を付与したことにより，非常に綺麗にそして迅速に系全体がスピン 0 の状態に遷移したことが読み取れる．自発的対称性の破れを引き起こすためには最隣接スピンとの相互作用のみならず，系全体との相互作用である平均場を導入することが重要であることがわかった．

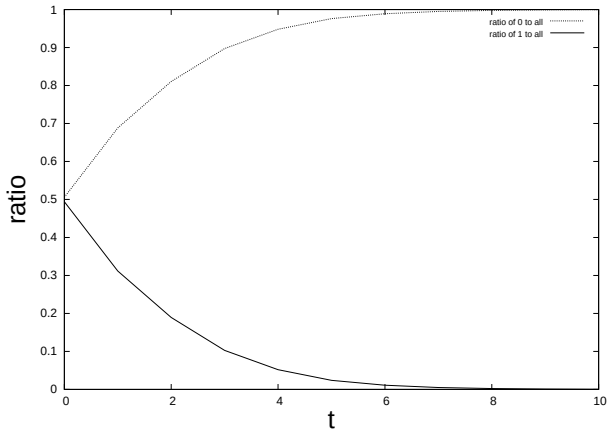

図 4　平均場を加えた規約 1 のもとでの振舞い

6　考察

はじめ，規約 1, 規約 2, 規約 3 はどれも直感的には正しそうで，対称性の破れを引き起こすには十分だろうと考えていた．しかし，実際にシミュレーションしてみると，規約 1, 規約 2 では対称性は維持されたままで，スピンの数はそれぞれほとんど常に半々であった．規約 3 ではかろうじてバランスを崩すことができたが，不十分で，しかも遷移が遅かった．

これの原因は系の状態をターミナルで見た時にすぐに読み取れた．系のいたるところに以下のような振動構造が見られたのである．

表 1　振動構造

0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1

この構造は非常に厄介で，スピンの遷移が一斉に起こる今回のシミュレーションでは構造中のそれぞれのスピンが 0 と 1 を振動し，永遠にそのバランスが崩れない．これに気がついたのが結果 1 の段階で，他に上手い規約が思いつかなかったため，振動構造を崩すために平均場を導入せざるを得なかった．

平均場を導入した結果は，図 4 でわかるように成功であった．ここで面白いのは，図 4 の $t = 0$ を見てみる

とスピンのバランスがほんの僅かに崩れているということである．これからわかることは規約 3 の結果でもそうだったように，確率的なゆらぎ，スピンの割合のゆらぎが，自発的対称性の破れにおいては重要だということである．

7　結論

本稿では，数値シミュレーションによって，強磁性体がもつ自発的対称性の破れを実現する条件を探った．そのモデルとして 2 次元イジング模型を扱った．
その結果，単に最隣接スピン間の相互作用により，エネルギーを小さくしようとするのみでは不十分であり，いわゆる「多数決」では自発的対称性の破れは起こらないことがわかった．これは直感に反した結果であった．この原因は，系の遷移が同時に起こる場合，系の至るところに現れる振動構造が永遠にスピンのバランスを崩さないためであった．

そこで，最隣接スピン間の相互作用に加えて平均場を導入した．そして，それを踏まえて系のエネルギーを小さくするように挙動させた．その結果，振動構造を崩すことに成功し，系はあっという間に自発的対称性の破れを実現した．
ここからわかることは，自発的対称性の破れを引き起こすためには最隣接スピンとの相互作用のみならず，系全体との相互作用である平均場も重要であるということである．また，確率的なゆらぎ，スピンの割合のゆらぎというランダムな要素も重要である．

最後に本稿の問題点を列挙する．

- 系の遷移が同時に起こる．
- 初期状態の影響を考察できていない．
- 1 次元，3 次元について実験できていない．
- 平均場の重み a について $a = 1$ の場合しか実験できていない．

これらは，時間的に都合がつかなかったこともあるが，1 つめに関しては，短い計算時間でかつスピンがバラバラに遷移していく上手いアルゴリズムを実装できなかったことによる．この辺に関してはまたの機会に試してみたい．

参考文献

なし