

魔方陣

smat1957@gmail.com^{*1}

2025 年 4 月 26 日

^{*1} <https://altema.is.tohoku.ac.jp/QA4U3/>

第 1 章

魔方陣を量子アニーリングで解く

魔方陣は、 $N \times N$ の各セルに $1 \sim N^2$ の数値を一つずつ入れ、縦横斜めのどの列についても、その列の数値の総和が等しくなる様に、数値を配置するもの。

例えば 3×3 の魔方陣であれば、縦横斜めのどの列にある数値の総和も等しく 15 になる。

	2	9	4	→ 15
	7	5	3	→ 15
	6	1	8	→ 15
↙ 15	↓ 15	↓ 15	↓ 15	↘ 15

図 1.1 参考資料 [2] より

3×3 は、対称な形を 1 つと数えることにすると 1 通り、 4×4 では 880 通り、 5×5 では 275305224 通り、 6×6 では 17753889197660635632 通り存在することがわかっている。[2]

1.1 問題の構成

決定変数 q を各セル毎に $N \times N$ 個用意する。 $q_{i,j,n}$ は、 i 行 j 列目のセル内の $N \times N$ 個の決定変数。ここで、 $i, j \in \{1, 2, \dots, N\}$, $n \in \{1, 2, \dots, N \times N\}$ 。表は $N = 3$ の場合。

	1 列目 ($j = 1$)				2 列目 ($j = 2$)				3 列目 ($j = 3$)			
1 行目 ($i = 1$)	セル ($i = 1, j = 1$)				セル ($i = 1, j = 2$)				セル ($i = 1, j = 3$)			
	1	2	...	9	1	2	...	9	1	2	...	9
	q_{111}	q_{112}	...	q_{119}	q_{121}	q_{122}	...	q_{129}	q_{131}	q_{132}	...	q_{139}
2 行目 ($i = 2$)	セル ($i = 2, j = 1$)				セル ($i = 2, j = 2$)				セル ($i = 2, j = 3$)			
	1	2	...	9	1	2	...	9	1	2	...	9
	q_{211}	q_{212}	...	q_{219}	q_{221}	q_{222}	...	q_{229}	q_{231}	q_{232}	...	q_{239}
3 行目 ($i = 3$)	セル ($i = 3, j = 1$)				セル ($i = 3, j = 2$)				セル ($i = 3, j = 3$)			
	1	2	...	9	1	2	...	9	1	2	...	9
	q_{311}	q_{312}	...	q_{319}	q_{321}	q_{322}	...	q_{329}	q_{331}	q_{332}	...	q_{339}

この決定変数は0か1かの2値変数で、当該数値1～ $N \times N$ をそのセルに置く(1)か置かない(0)かを表すものとする。すると制約条件は、次の様に考えることができる。 $(M = N \times N)$ とする)

1. 各セルの中では1～9の中のどれか1つだけが1になる(セルに2つ以上の数値は入らない)
 $\rightarrow \sum_i q_i = 1 \text{ (for cell = 1} \sim 9)$

$$f_1 = \sum_i^N \sum_j^N \left(\sum_n^M q_{i,j,n} - 1 \right)^2$$

2. あるセルの数値と同じ数値は、他のセルには入らない

$$\rightarrow (q_i + q_{i+9} + q_{i+2 \times 9} + \dots + q_{i+8 \times 9}) = \sum_{cell=1}^9 q_{i+(cell-1) \times 9} = 1 \text{ (for } i = 1 \sim 9)$$

$$f_2 = \sum_n^M \left(\sum_i^N \sum_j^N q_{i,j,n} - 1 \right)^2$$

3. いずれの行方向の数値の和も同じ値 $S(= 15)$ になる

$$\begin{aligned} \rightarrow \sum_{cell=1}^3 (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \\ \sum_{cell=4}^6 (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \\ \sum_{cell=7}^9 (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \end{aligned}$$

$$f_3 = \sum_i^N \left(\sum_j^N \sum_n^M n \cdot q_{i,j,n} - S \right)^2$$

4. いずれの列方向の数値の和も同じ値 $S(= 15)$ になる

$$\begin{aligned} \rightarrow \sum_{cell=1,4,7} (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \\ \sum_{cell=2,5,8} (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \\ \sum_{cell=3,6,9} (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \end{aligned}$$

$$f_4 = \sum_j^N \left(\sum_i^N \sum_n^M n \cdot q_{i,j,n} - S \right)^2$$

5. 右下がりの対角要素の和と右上がりの対角要素の和も同じ値 $S(= 15)$ になる

$$\begin{aligned} \rightarrow \sum_{cell=1,5,9} (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \\ \sum_{cell=3,5,7} (\sum_{i=1}^9 i \times q_{i+(cell-1) \times 9}) &= 15 \end{aligned}$$

$$f_5 = \left(\sum_d^N \sum_n^M n \cdot q_{d,d,n} - S \right)^2 + \left(\sum_d^N \sum_n^M n \cdot q_{d,N-d+1,n} - S \right)^2$$

1.2 式の展開と実装

- 式を展開する上で留意する点は次の2点だけ
 - (1) 0か1の何れかの値しかとらない二値変数の場合 $q^2 = q$ が成り立つ
 - (2) 定数は最小化に関係ないので無視できる
- 展開した制約式に現れる \sum を、そのまま for 文の繰り返しの移せば QUBO を生成できる
- QUBO ができたら、それを量子コンピュータのシミュレータである、SASampler() あるいは SQAASampler() の第1引数に渡してあげると、計算結果の sampleset を受け取ることができる

- 数式上で N 個の数値を $\sum_{i=1}^N$ の様に扱っていても、プログラム上の始まりの値は0なので、全部で N 個の数値を for 文で繰り返すとすると、終わりの値は $N - 1$ になる
- また、魔方陣の盤面に置く数値は0～9の10個ではなくて、1～9の9個である事にも注意してプログラムする必要がある

1.2.1 class

```
from openjij import SASampler, SQAASampler
from collections import defaultdict, Counter
import numpy as np

class MagicCircle:
    def __init__(self, N=3):
        self.N = N          # self.N = 3
        self.M = N * N      # self.M = 9
        self.S = N * (N**2 + 1) // 2  # self.S = 15
        self.idx = {}
        k = 0
        for i in range(self.N):
            for j in range(self.N):
                for n in range(self.M):
                    self.idx[(i,j,n)] = k
                    k += 1
        samplers = [SASampler(), SQAASampler()]
        self.sampler = samplers[0]

    def get_param(self):
        return self.N, self.M, self.S, self.idx
```

1.2.2 制約： f_1

各セルの中では1～9の中のどれか1つだけが1になる（セルに2つ以上の数値は入らない）

$$\begin{aligned}
 f_1 &= \sum_i^N \sum_j^N \left(\sum_n^M q_{i,j,n} - 1 \right)^2 \\
 &= \sum_i \sum_j \left(\sum_{n_1} \sum_{n_2} q_{i,j,n_1} q_{i,j,n_2} - 2 \sum_n q_{i,j,n} \right)
 \end{aligned}$$

```
def sub1(self, i, j, L, Q):
    N, M, _, idx = self.get_param()
    for n1 in range(M):
        Q[(idx[(i, j, n1)], idx[(i, j, n1)])] -= 2.0 * L
    for n2 in range(M):
        Q[(idx[(i, j, n1)], idx[(i, j, n2)])] += 1.0 * L
```

```
def f1(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        for j in range(N):
            self.sub1(i, j, L, Q)
    return Q
```

1.2.3 制約： f_2

あるセルの数値と同じ数値は、他のセルには入らない

$$f_2 = \sum_n \left(\sum_i \sum_j q_{i,j,n} - 1 \right)^2$$

$$= \sum_n \left(\sum_{i_1} \sum_{j_1} q_{i_1,j_1,n} \sum_{i_2} \sum_{j_2} q_{i_2,j_2,n} - 2 \sum_i \sum_j q_{i,j,n} \right)$$

```
def sub2(self, n, L, Q):
    N, _, _, idx = self.get_param()
    for i1 in range(N):
        for j1 in range(N):
            Q[(idx[(i1, j1, n)], idx[(i1, j1, n)]] -= 2.0 * L
        for i2 in range(N):
            for j2 in range(N):
                Q[(idx[(i1, j1, n)], idx[(i2, j2, n)]] += 1.0 * L

def f2(self, L, Q):
    _, M, _, _ = self.get_param()
    for n in range(M):
        self.sub2(n, L, Q)
    return Q
```

1.2.4 制約： f_3

いずれの行方向の数値の和も同じ値 S になる

$$f_3 = \sum_i \left(\sum_j \sum_n n \cdot q_{i,j,n} - S \right)^2$$

$$= \sum_i \left(\sum_{j_1} \sum_{n_1} n_1 \cdot q_{i,j_1,n_1} \sum_{j_2} \sum_{n_2} n_2 \cdot q_{i,j_2,n_2} - 2 \cdot S \sum_j \sum_n n \cdot q_{i,j,n} \right)$$

```
def sub3(self, i, L, Q):
    N, M, S, idx = self.get_param()
    for j1 in range(N):
```

```

        for n1 in range(M):
            Q[(idx[(i, j1, n1)], idx[(i, j1, n1)])] -= 2.0 * (n1+1) * S * L
            for j2 in range(N):
                for n2 in range(M):
                    Q[(idx[(i, j1, n1)], idx[(i, j2, n2)])] += (n1+1) * (n2+1) * L

def f3(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        self.sub3(i, L, Q)
    return Q

```

1.2.5 制約： f_4

いずれの列方向の数値の和も同じ値 S になる

$$\begin{aligned}
 f_4 &= \sum_j \left(\sum_i \sum_n n \cdot q_{i,j,n} - S \right)^2 \\
 &= \sum_j \left(\sum_{i_1} \sum_{n_1} n_1 \cdot q_{i_1,j,n_1} \sum_{i_2} \sum_{n_2} n_2 \cdot q_{i_2,j,n_2} - 2 \cdot S \sum_i \sum_n n \cdot q_{i,j,n} \right)
 \end{aligned}$$

```

def sub4(self, j, L, Q):
    N, M, S, idx = self.get_param()
    for i1 in range(N):
        for n1 in range(M):
            Q[(idx[(i1, j, n1)], idx[(i1, j, n1)])] -= 2.0 * (n1+1) * S * L
            for i2 in range(N):
                for n2 in range(M):
                    Q[(idx[(i1, j, n1)], idx[(i2, j, n2)])] += (n1+1) * (n2+1) * L

def f4(self, L, Q):
    N, _, _, _ = self.get_param()
    for j in range(N):
        self.sub4(j, L, Q)
    return Q

```

1.2.6 制約： f_5

右下がりの対角要素の和と右上がりの対角要素の和も同じ値 S になる

$$\begin{aligned}
f_5 &= \left(\sum_d^N \sum_n^M n \cdot q_{d,d,n} - S \right)^2 + \left(\sum_d^N \sum_n^M n \cdot q_{d,N-d+1,n} - S \right)^2 \\
&= \left(\sum_{d_1} \sum_{n_1} n_1 \cdot q_{d_1,d_1,n_1} \sum_{d_2} \sum_{n_2} n_2 \cdot q_{d_2,d_2,n_2} - 2 \cdot S \sum_d \sum_n n \cdot q_{d,d,n} \right) \\
&\quad + \left(\sum_{d_1} \sum_{n_1} n_1 \cdot q_{d_1,N-d_1+1,n_1} \sum_{d_2} \sum_{n_2} n_2 \cdot q_{d_2,N-d_2+1,n_2} - 2 \cdot S \sum_d \sum_n n \cdot q_{d,N-d+1,n} \right)
\end{aligned}$$

```

def f5(self, L, Q):
    N, M, S, idx = self.get_param()
    for d1 in range(N):
        for n1 in range(M):
            Q[(idx[(d1, d1, n1)], idx[(d1, d1, n1)])] -= 2.0*(n1+1)*S*L
            Q[(idx[(d1, N-d1-1, n1)], idx[(d1, N-d1-1, n1)])] -= 2.0*(n1 + 1)*S*L
            for d2 in range(N):
                for n2 in range(M):
                    Q[(idx[(d1, d1, n1)], idx[(d2, d2, n2)])] += (n1+1)*(n2+1)*L
                    Q[(idx[(d1, N-d1-1, n1)], idx[(d2, N-d2-1, n2)])] += (n1+1)*(n2+1)*L
    return Q

```

1.2.7 評価関数：f

$$f = \lambda_1 \cdot f_1 + \lambda_2 \cdot f_2 + \lambda_3 \cdot (f_3 + f_4 + f_5)$$

```

def f(self, lagrange1=1.0, lagrange2=1.0, lagrange3=1.0):
    Q = defaultdict(lambda: 0)
    _ = self.f1(lagrange1, Q)
    _ = self.f2(lagrange2, Q)
    _ = self.f3(lagrange3, Q)
    _ = self.f4(lagrange3, Q)
    _ = self.f5(lagrange3, Q)
    return Q

def solv(self, Q, num_reads=1):
    sampleset = self.sampler.sample_qubo(Q, num_reads=num_reads)
    return sampleset

def result(self, sampleset):
    N, M, S, idx = self.get_param()
    result = [i for i in sampleset.first[0].values()]
    ans = [[None] * N for _ in range(N)]
    for i in range(N):
        for j in range(N):
            for n in range(N**2):
                if result[idx[(i, j, n)]] == 1:
                    ans[i][j] = n+1

```

```
return ans
```

出力結果のチェック

出力された結果を、ふるいにかける仕掛け

```
def evaluate(self, sampleset, prn=True):
    # Extract sample solutions, energies, and sort them by frequency
    samples = sampleset.record['sample']
    energies = sampleset.record['energy']
    # Combine solutions and corresponding energies
    sample_data = [(tuple(sample), energy) for sample, energy in zip(samples,
        energies)]
    # Sort the results by appearance frequency and then energy
    sample_frequency = Counter(sample for sample, _ in sample_data)
    # Print sorted results by frequency and include energy
    if prn:
        print("\nSorted samples by frequency and energy:")
        for solution, freq in sample_frequency.most_common():
            energy = next(energy for sample, energy in sample_data if sample ==
                solution)
            print(f"Sample: {solution}, Frequency: {freq}, Energy: {energy:+.2f}")
    return sample_data, sample_frequency

def check1(self, a):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(M, M)
    for i in range(M):
        s = 0
        for j in range(M):
            s += b[i][j]
        if s!=1:
            return False
    for j in range(M):
        s = 0
        for i in range(M):
            s += b[i][j]
        if s!=1:
            return False
    return True

def check2(self, a):
    N, M, S, _ = self.get_param()
    b = np.array(a).reshape(N, N)
    for i in range(N):
        s = 0
        for j in range(N):
            s += b[i][j]
        if s!= S:
            return False
    #
    for j in range(N):
        s = 0
```



```

        for i in range(N):
            s += b[i][j]
        if s != S:
            return False
#
s = 0
for i in range(N):
    for j in range(N):
        if i==j:
            s += b[i][j]
if s!=S:
    return False
#
s = 0
for i in range(N):
    k = N-i-1
    s += b[i][k]
if s!=S:
    return False
#
return True

def decode(self, a):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(M, M)
    mat = []
    for i in range(M):
        for j in range(M):
            if b[i][j]==1:
                mat.append(j+1)
    return mat

```

1.2.8 main

```

if __name__ == '__main__':
    N = 3
    mc = MagicCircle(N)
    lagrange1 = 10.0
    lagrange2 = 10.0
    lagrange3 = 1.0
    Q = mc.f(lagrange1, lagrange2, lagrange3)
    num_reads = 1000
    sampleset = mc.solve(Q, num_reads)
    #ans = mc.result(sampleset)
    #print(*ans, sep='\n')
    #
    for sample in sampleset.record['sample']:
        if mc.check1(sample):
            a = mc.decode(sample)
            if mc.check2(a):
                print(np.array(a).reshape(N, N))

```

```
| print()
```

1.3 実行結果

複数出力されているのは、対称な形をチェックしていないため

```
[[2 9 4]
 [7 5 3]
 [6 1 8]]
```

```
[[6 1 8]
 [7 5 3]
 [2 9 4]]
```

```
[[2 7 6]
 [9 5 1]
 [4 3 8]]
```

```
[[8 1 6]
 [3 5 7]
 [4 9 2]]
```

```
[[2 9 4]
 [7 5 3]
 [6 1 8]]
```

```
[[6 7 2]
 [1 5 9]
 [8 3 4]]
```

プロセスは終了コード 0 で終了しました

1.4 プログラムの全体

```
from openjij import SASampler, SQA_Sampler
from collections import defaultdict, Counter
import numpy as np

class MagicCircle:
    def __init__(self, N=3):
        self.N = N      # self.N = 3
```

```

self.M = N * N      # self.M = 9
self.S = N * (N**2 + 1) // 2  # self.S = 15
self.idx = {}
k = 0
for i in range(self.N):
    for j in range(self.N):
        for n in range(self.M):
            self.idx[(i,j,n)] = k
            k += 1
samplers = [SASampler(), SQASampler()]
self.sampler = samplers[0]

def get_param(self):
    return self.N, self.M, self.S, self.idx

def sub1(self, i, j, L, Q):
    N, M, _, idx = self.get_param()
    for n1 in range(M):
        Q[(idx[(i, j, n1)], idx[(i, j, n1)])] -= 2.0 * L
        for n2 in range(M):
            Q[(idx[(i, j, n1)], idx[(i, j, n2)])] += 1.0 * L

def f1(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        for j in range(N):
            self.sub1(i, j, L, Q)
    return Q

def sub2(self, n, L, Q):
    N, _, _, idx = self.get_param()
    for i1 in range(N):
        for j1 in range(N):
            Q[(idx[(i1, j1, n)], idx[(i1, j1, n)])] -= 2.0 * L
            for i2 in range(N):
                for j2 in range(N):
                    Q[(idx[(i1, j1, n)], idx[(i2, j2, n)])] += 1.0 * L

def f2(self, L, Q):
    _, M, _, _ = self.get_param()
    for n in range(M):
        self.sub2(n, L, Q)
    return Q

def sub3(self, i, L, Q):
    N, M, S, idx = self.get_param()
    for j1 in range(N):
        for n1 in range(M):
            Q[(idx[(i, j1, n1)], idx[(i, j1, n1)])] -= 2.0 * (n1+1) * S * L
            for j2 in range(N):
                for n2 in range(M):
                    Q[(idx[(i, j1, n1)], idx[(i, j2, n2)])] += (n1+1) * (n2+1)
* L

```

```

def f3(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        self.sub3(i, L, Q)
    return Q

def sub4(self, j, L, Q):
    N, M, S, idx = self.get_param()
    for i1 in range(N):
        for n1 in range(M):
            Q[(idx[(i1, j, n1)], idx[(i1, j, n1)])] -= 2.0 * (n1+1) * S * L
        for i2 in range(N):
            for n2 in range(M):
                Q[(idx[(i1, j, n1)], idx[(i2, j, n2)])] += (n1+1) * (n2+1)
    * L

def f4(self, L, Q):
    N, _, _, _ = self.get_param()
    for j in range(N):
        self.sub4(j, L, Q)
    return Q

def f5(self, L, Q):
    N, M, S, idx = self.get_param()
    for d1 in range(N):
        for n1 in range(M):
            Q[(idx[(d1, d1, n1)], idx[(d1, d1, n1)])] -= 2.0 * (n1+1) * S * L
            Q[(idx[(d1, N-d1-1, n1)], idx[(d1, N-d1-1, n1)])] -= 2.0 * (n1 + 1)
    * S * L
        for d2 in range(N):
            for n2 in range(M):
                Q[(idx[(d1, d1, n1)], idx[(d2, d2, n2)])] += (n1+1) * (n2
+1) * L
                Q[(idx[(d1, N-d1-1, n1)], idx[(d2, N-d2-1, n2)])] += (n1 +
1) * (n2 + 1) * L
    return Q

def f(self, lagrange1=1.0, lagrange2=1.0, lagrange3=1.0):
    Q = defaultdict(lambda: 0)
    _ = self.f1(lagrange1, Q)
    _ = self.f2(lagrange2, Q)
    _ = self.f3(lagrange3, Q)
    _ = self.f4(lagrange3, Q)
    _ = self.f5(lagrange3, Q)
    return Q

def solv(self, Q, num_reads=1):
    sampleset = self.sampler.sample_qubo(Q, num_reads=num_reads)
    return sampleset

def result(self, sampleset):
    N, M, S, idx = self.get_param()
    result = [i for i in sampleset.first[0].values()]
    ans = [[None] * N for _ in range(N)]

```

```

        for i in range(N):
            for j in range(N):
                for n in range(N**2):
                    if result[idx[(i,j,n)]] == 1:
                        ans[i][j] = n+1
            return ans

def evaluate(self, sampleset, prn=True):
    # Extract sample solutions, energies, and sort them by frequency
    samples = sampleset.record['sample']
    energies = sampleset.record['energy']
    # Combine solutions and corresponding energies
    sample_data = [(tuple(sample), energy) for sample, energy in zip(samples,
energies)]
    # Sort the results by appearance frequency and then energy
    sample_frequency = Counter(sample for sample, _ in sample_data)
    # Print sorted results by frequency and include energy
    if prn:
        print("\nSorted samples by frequency and energy:")
        for solution, freq in sample_frequency.most_common():
            energy = next(energy for sample, energy in sample_data if sample ==
solution)
            print(f"Sample: {solution}, Frequency: {freq}, Energy: {energy:+.2f
}")
    return sample_data, sample_frequency

def check1(self, a):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(M, M)
    for i in range(M):
        s = 0
        for j in range(M):
            s += b[i][j]
        if s!=1:
            return False
    for j in range(M):
        s = 0
        for i in range(M):
            s += b[i][j]
        if s!=1:
            return False
    return True

def check2(self, a):
    N, M, S, _ = self.get_param()
    b = np.array(a).reshape(N, N)
    for i in range(N):
        s = 0
        for j in range(N):
            s += b[i][j]
        if s!= S:
            return False
    #
    for j in range(N):

```

```

        s = 0
        for i in range(N):
            s += b[i][j]
        if s != S:
            return False

#
s = 0
for i in range(N):
    for j in range(N):
        if i==j:
            s += b[i][j]
if s!=S:
    return False

#
s = 0
for i in range(N):
    k = N-i-1
    s += b[i][k]
if s!=S:
    return False

#
return True

def decode(self, a):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(M, M)
    mat = []
    for i in range(M):
        for j in range(M):
            if b[i][j]==1:
                mat.append(j+1)
    return mat

if __name__ == '__main__':
    N = 3
    mc = MagicCircle(N)
    lagrange1 = 10.0
    lagrange2 = 10.0
    lagrange3 = 1.0
    Q = mc.f(lagrange1, lagrange2, lagrange3)
    num_reads = 1000
    sampleset = mc.solve(Q, num_reads)
    #ans = mc.result(sampleset)
    #print(*ans, sep='\n')
    #
    for sample in sampleset.record['sample']:
        if mc.check1(sample):
            a = mc.decode(sample)
            if mc.check2(a):
                print(np.array(a).reshape(N, N))
                print()

```

プログラム 1.1 魔方陣

参考文献

- [1] 西森秀稔、大関真之, 量子アニーリングの基礎, 共立出版
- [2] https://zenn.dev/luna_moonlight/articles/38de858bdc855f