

数独

smat1957@gmail.com^{*1}

2025 年 4 月 28 日

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

第 1 章

数独を量子アニーリングで解く

数独は $M \times M$ のブロックを、行方向に M 列分、列方向に M 行分並べた、全 $M \times M = M^2$ ブロック、従って $M^2 \times M^2$ 個のセルからなる盤面で、どの行および列についても、また $M \times M$ の各ブロックの中においても、同じ数値が 2 個以上現れてはならないという制約の下、各セルに $1 \sim M^2$ までの数値を一つずつ入れて盤面を埋めていくクイズ。

1.1 問題の構成

決定変数 q を各セル毎に $M \times M = M^2$ 個用意する。 $q_{i,j,n}$ は、 i 行 j 列目のセル内の $M \times M = M^2$ 個の決定変数。

ここで、 $i, j, n \in \{1, 2, \dots, M \times M = M^2\}$ 。

下の表は 3×3 のブロック 1 個の例を表している。このブロックが横に 3 行、縦に 3 列、全部で 9 ブロックが並んでいる盤面がよく知られた数独問題になる。

	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、数値 $1 \sim N$ をそのセルに置く (1) か置かない (0) かを表している。

制約条件は、次の様に考えることができる。 $(M \times M = M^2)$ を N と書く事にする)

1. 各セルの中では $q_1 \sim q_N$ の内でどれか 1 つだけが 1 になる (セルに 2 つ以上の数値は入らない)

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

2. 同一の行 (列) にあるセルの数値と同じ数値は、同じ行 (列) の他のセルには入らない (第 1 項が行、第 2 項が列)

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

3. いずれのブロック $(M \times M)$ においても、その中のセルの数値は重複しない

$$f_3 = \sum_{\text{ブロック先頭セルの } i_0, j_0} \sum_n^N \left(\sum_x^M \sum_y^M q_{i_0+x-1, j_0+y-1, n} - 1 \right)^2$$

4. いずれの行 (列) 方向の数値の和も同じ値 $S (= 1 + 2 + \dots + N)$ になる (第 1 項が行、第 2 項が列)

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

5. いずれのブロック $(M \times M)$ においても、その中のセルの数値の和は同じ値 $S (= 1 + 2 + \dots + N)$ になる

$$f_5 = \sum_{\text{ブロック先頭セルの } i_0, j_0} \left(\sum_x^M \sum_y^M \sum_n^N n \cdot q_{i_0+x-1, j_0+y-1, n} - S \right)^2$$

6. 予め数値 $X \in \{1, \dots, N\}$ が決められている I 行 J 列目のセルがある

$$f_6 = \sum_{\text{既定のセル } I, J} \left(\sum_n^N n \cdot q_{I, J, n} - X \right)^2$$

1.1.1 式の展開

$$\begin{aligned}
f_1 &= \sum_i^N \sum_j^N \left(\sum_n^N q_{i,j,n} - 1 \right)^2 \\
&= \sum_i^N \sum_j^N \left(\sum_{n_1}^N \sum_{n_2}^N q_{i,j,n_1} q_{i,j,n_2} - 2 \sum_n^N q_{i,j,n} \right) \\
f_2 &= \sum_i^N \sum_n^N \left(\sum_j^N q_{i,j,n} - 1 \right)^2 + \sum_j^N \sum_n^N \left(\sum_i^N q_{i,j,n} - 1 \right)^2 \\
&= \sum_i^N \sum_n^N \left(\sum_{j_1}^N q_{i,j_1,n} \sum_{j_2}^N q_{i,j_2,n} - 2 \sum_j^N q_{i,j,n} \right) + \sum_j^N \sum_n^N \left(\sum_{i_1}^N q_{i_1,j,n} \sum_{i_2}^N q_{i_2,j,n} - 2 \sum_i^N q_{i,j,n} \right) \\
f_3 &= \sum_{\text{ブロック先頭セルの } i_0, j_0}^N \sum_n^N \left(\sum_x^M \sum_y^M q_{i_0+x-1, j_0+y-1, n} - 1 \right)^2 \\
&= \sum_{\text{ブロック先頭セルの } i_0, j_0}^N \sum_n^N \left(\sum_{x_1}^M \sum_{y_1}^M q_{i_0+x_1-1, j_0+y_1-1, n} \sum_{x_2}^M \sum_{y_2}^M q_{i_0+x_2-1, j_0+y_2-1, n} \right. \\
&\quad \left. - 2 \sum_x^M \sum_y^M q_{i_0+x-1, j_0+y-1, n} \right) \\
f_4 &= \sum_i^N \left(\sum_j^N \sum_n^N n \cdot q_{i,j,n} - S \right)^2 + \sum_j^N \left(\sum_i^N \sum_n^N n \cdot q_{i,j,n} - S \right)^2 \\
&= \sum_i^N \left(\sum_{j_1}^N \sum_{n_1}^N n_1 q_{i,j_1,n_1} \sum_{j_2}^N \sum_{n_2}^N n_2 q_{i,j_2,n_2} - 2S \sum_j^N \sum_n^N n q_{i,j,n} \right) \\
&\quad + \sum_j^N \left(\sum_{i_1}^N \sum_{n_1}^N n_1 q_{i_1,j,n_1} \sum_{i_2}^N \sum_{n_2}^N n_2 q_{i_2,j,n_2} - 2S \sum_i^N \sum_n^N n q_{i,j,n} \right) \\
f_5 &= \sum_{\text{ブロック先頭セルの } i_0, j_0}^N \left(\sum_x^M \sum_y^M \sum_n^N n \cdot q_{i_0+x-1, j_0+y-1, n} - S \right)^2 \\
&= \sum_{\text{ブロック先頭セルの } i_0, j_0}^N \left(\sum_{x_1}^M \sum_{y_1}^M \sum_{n_1}^N n_1 q_{i_0+x_1-1, j_0+y_1-1, n_1} \sum_{x_2}^M \sum_{y_2}^M \sum_{n_2}^N n_2 q_{i_0+x_2-1, j_0+y_2-1, n_2} \right. \\
&\quad \left. - 2S \sum_x^M \sum_y^M \sum_n^N n q_{i_0+x-1, j_0+y-1, n} \right) \\
f_6 &= \sum_{\text{既定のセル } I, J} \left(\sum_n^N n \cdot q_{I,J,n} - X \right)^2 \\
&= \sum_{\text{既定セルの } I, J} \left(\sum_{n_1}^N n_1 q_{I,J,n_1} \sum_{n_2}^N n_2 q_{I,J,n_2} - 2X \sum_n^N n q_{I,J,n} \right)
\end{aligned}$$

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 文の繰り返しに移せば 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 SQASampler() 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 文で繰り返すとなると、終わりの値は $N - 1$ になる
- また、盤面に置く数値は $0 \sim N$ の $N + 1$ 個ではなくて、 $1 \sim N$ の N 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, SQASampler
from collections import defaultdict, Counter
import numpy as np

class NumberPlace:
    def __init__(self, M=2, FileN='data.txt'):
        self.M = M
        self.N = M * M
        S = 0
        for i in range(1, self.N+1):
            S += i
        self.S = S
        with open(FileN, 'r') as f:
            self.required = f.read().splitlines()
        self.idx = {}
        k = 0
        for i in range(self.N):
            for j in range(self.N):
                for n in range(self.N):
                    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 block_ij(self):
        i0j0 = []
        for i in range(self.M):
            i0j0.append(self.M*i)
```

```
return i0j0
```

ブロック先頭セルの*i*₀, *j*₀は、

M=2なら [(0,0),(0,2)],
[(2,0),(2,2)]

M=3なら [(0,0),(0,3),(0,6)],
[(3,0),(3,3),(3,6)],
[(6,0),(6,3),(6,6)]

M=2なら [(*M**0,*M**0),(*M**0,*M**1)],
[(*M**1,*M**0),(*M**1,*M**1)]

M=3なら [(*M**0,*M**0),(*M**0,*M**1),(*M**0,*M**2)],
[(*M**1,*M**0),(*M**1,*M**1),(*M**1,*M**2)],
[(*M**2,*M**0),(*M**2,*M**1),(*M**2,*M**2)]

1.2.2 制約：*f*₁

各セルの中では *q*₁ ~ *q*_{*N*} の内でどれか 1 つだけが 1 になる（セルに 2 つ以上の数値は入らない）

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

$$= \sum_i^N \sum_j^N \left(\sum_{n_1}^N \sum_{n_2}^N q_{i,j,n_1} q_{i,j,n_2} - 2 \sum_n^N q_{i,j,n} \right)$$

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

同一の行（列）にあるセルの数値と同じ数値は、同じ行（列）の他のセルには入らない（第 1 項が行、第 2 項が列）

$$\begin{aligned}
f_2 &= \sum_i^N \sum_n^N \left(\sum_j^N q_{i,j,n} - 1 \right)^2 + \sum_j^N \sum_n^N \left(\sum_i^N q_{i,j,n} - 1 \right)^2 \\
&= \sum_i^N \sum_n^N \left(\sum_{j_1}^N q_{i,j_1,n} \sum_{j_2}^N q_{i,j_2,n} - 2 \sum_j^N q_{i,j,n} \right) + \sum_j^N \sum_n^N \left(\sum_{i_1}^N q_{i_1,j,n} \sum_{i_2}^N q_{i_2,j,n} - 2 \sum_i^N q_{i,j,n} \right)
\end{aligned}$$

```

def sub2R(self, i, n, L, Q):
    N, _, _, idx = self.get_param()
    for j1 in range(N):
        Q[(idx[(i, j1, n)], idx[(i, j1, n)])] -= 2.0 * L
        for j2 in range(N):
            Q[(idx[(i, j1, n)], idx[(i, j2, n)])] += 1.0 * L

def sub2C(self, j, n, L, Q):
    N, _, _, idx = self.get_param()
    for i1 in range(N):
        Q[(idx[(i1, j, n)], idx[(i1, j, n)])] -= 2.0 * L
        for i2 in range(N):
            Q[(idx[(i1, j, n)], idx[(i2, j, n)])] += 1.0 * L

def f2(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        for n in range(N):
            self.sub2R(i, n, L, Q)
    for j in range(N):
        for n in range(N):
            self.sub2C(j, n, L, Q)
    return Q

```

1.2.4 制約： f_3

いずれのブロック（ $M \times M$ ）においても、その中のセルの数値は重複しない

$$\begin{aligned}
f_3 &= \sum_{\text{ブロック先頭セルの } i_0, j_0} \sum_n^N \left(\sum_x^M \sum_y^M q_{i_0+x-1, j_0+y-1, n} - 1 \right)^2 \\
&= \sum_{\text{ブロック先頭セルの } i_0, j_0} \sum_n^N \left(\sum_{x_1}^M \sum_{y_1}^M q_{i_0+x_1-1, j_0+y_1-1, n} \sum_{x_2}^M \sum_{y_2}^M q_{i_0+x_2-1, j_0+y_2-1, n} \right. \\
&\quad \left. - 2 \sum_x^M \sum_y^M q_{i_0+x-1, j_0+y-1, n} \right)
\end{aligned}$$

```

def sub3(self, i0, j0, n, L, Q):
    N, M, _, idx = self.get_param()
    for x1 in range(M):
        for y1 in range(M):

```

```

        Q[(idx[(i0 + x1, j0 + y1, n)], idx[(i0 + x1, j0 + y1, n)])] -= 2.0 * L
        for x2 in range(M):
            for y2 in range(M):
                Q[(idx[(i0 + x1, j0 + y1, n)], idx[(i0 + x2, j0 + y2, n)])] +=
1.0 * L

def f3(self, L, Q):
    N, _, _, idx = self.get_param()
    i0j0 = self.block_ij()
    for i0 in i0j0:
        for j0 in i0j0:
            for n in range(N):
                self.sub3(i0, j0, n, L, Q)
    return Q

```

1.2.5 制約： f_4

いずれの行（列）方向の数値の和も同じ値 $S(= 1 + 2 + \dots + N)$ になる（第1項が行、第2項が列）

$$\begin{aligned}
 f_4 &= \sum_i \left(\sum_j \sum_n n \cdot q_{i,j,n} - S \right)^2 + \sum_j \left(\sum_i \sum_n n \cdot q_{i,j,n} - S \right)^2 \\
 &= \sum_i \left(\sum_{j_1} \sum_{n_1} n_1 q_{i,j_1,n_1} \sum_{j_2} \sum_{n_2} n_2 q_{i,j_2,n_2} - 2S \sum_j \sum_n n q_{i,j,n} \right) \\
 &\quad + \sum_j \left(\sum_{i_1} \sum_{n_1} n_1 q_{i_1,j,n_1} \sum_{i_2} \sum_{n_2} n_2 q_{i_2,j,n_2} - 2S \sum_i \sum_n n q_{i,j,n} \right)
 \end{aligned}$$

```

def sub4R(self, i, L, Q):
    N, _, S, idx = self.get_param()
    for j1 in range(N):
        for n1 in range(N):
            Q[(idx[(i, j1, n1)], idx[(i, j1, n1)])] -= 2.0 * (n1+1) * S * L
        for j2 in range(N):
            for n2 in range(N):
                Q[(idx[(i, j1, n1)], idx[(i, j2, n2)])] += (n1+1) * (n2+1) * L

def sub4C(self, j, L, Q):
    N, _, S, idx = self.get_param()
    for i1 in range(N):
        for n1 in range(N):
            Q[(idx[(i1, j, n1)], idx[(i1, j, n1)])] -= 2.0 * (n1+1) * S * L
        for i2 in range(N):
            for n2 in range(N):
                Q[(idx[(i1, j, n1)], idx[(i2, j, n2)])] += (n1+1) * (n2+1) * L

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

```



```

        self.sub4R(i, L, Q)
    for j in range(N):
        self.sub4C(j, L, Q)
    return Q

```

1.2.6 制約： f_5

いずれのブロック ($M \times M$) においても、その中のセルの数値の和は同じ値 $S (= 1 + 2 + \dots + N)$ になる

$$\begin{aligned}
 f_5 &= \sum_{\text{ブロック先頭セルの } i_0, j_0} \left(\sum_x^M \sum_y^M \sum_n^N n \cdot q_{i_0+x-1, j_0+y-1, n} - S \right)^2 \\
 &= \sum_{\text{ブロック先頭セルの } i_0, j_0} \left(\sum_{x_1}^M \sum_{y_1}^M \sum_{n_1}^N n_1 q_{i_0+x_1-1, j_0+y_1-1, n_1} \sum_{x_2}^M \sum_{y_2}^M \sum_{n_2}^N n_2 q_{i_0+x_2-1, j_0+y_2-1, n_2} \right. \\
 &\quad \left. - 2S \sum_x^M \sum_y^M \sum_n^N n q_{i_0+x-1, j_0+y-1, n} \right)
 \end{aligned}$$

```

def sub5(self, i0, j0, L, Q):
    N, M, S, idx = self.get_param()
    for x1 in range(M):
        for y1 in range(M):
            for n1 in range(N):
                Q[(idx[(i0+x1, j0+y1, n1)], idx[(i0+x1, j0+y1, n1)])] -= 2.0 * (n1
+1) * S * L
                for x2 in range(M):
                    for y2 in range(M):
                        for n2 in range(N):
                            Q[(idx[(i0+x1, j0+y1, n1)], idx[(i0+x2, j0+y2, n2)])]
+= (n1+1) * (n2+1) * L

def f5(self, L, Q):
    i0j0 = self.block_ij()
    for i0 in i0j0:
        for j0 in i0j0:
            self.sub5(i0, j0, L, Q)
    return Q

```

1.2.7 制約： f_6

予め数値 $X \in \{1, \dots, N\}$ が決められている I 行 J 列目のセルがある

$$\begin{aligned}
f_6 &= \sum_{\text{既定のセル } I, J} \left(\sum_n^N n \cdot q_{I, J, n} - X \right)^2 \\
&= \sum_{\text{既定セルの } I, J} \left(\sum_{n_1}^N n_1 q_{I, J, n_1} \sum_{n_2}^N n_2 q_{I, J, n_2} - 2X \sum_n^N n q_{I, J, n} \right)
\end{aligned}$$

```

def f6(self, I, J, X, L, Q):
    N, _, _, idx = self.get_param()
    for n1 in range(N):
        Q[(idx[(I, J, n1)], idx[(I, J, n1)])] -= 2 * (n1+1) * X * L
        for n2 in range(N):
            Q[(idx[(I, J, n1)], idx[(I, J, n2)])] += (n1+1) * (n2+1) * L
    return Q

```

1.2.8 評価関数： f

$$f = \lambda_1 \cdot f_1 + \lambda_2 \cdot (f_2 + f_3) + \lambda_3 \cdot (f_4 + f_5) + \lambda_4 \cdot \left(\sum_{\text{既定}} f_6 \right)$$

```

def f(self, lagrange1=1.0, lagrange2=1.0, lagrange3=1.0, lagrange4=1.0):
    Q = defaultdict(lambda: 0)
    _ = self.f1(lagrange1, Q)
    _ = self.f2(lagrange2, Q)
    _ = self.f3(lagrange2, Q)
    if 0.0 < lagrange3:
        _ = self.f4(lagrange3, Q)
        _ = self.f5(lagrange3, Q)
    for a in self.required:
        IJX = a.split(',')
        _ = self.f6(int(IJX[0]), int(IJX[1]), int(IJX[2]), lagrange4, 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, _, _, 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):
                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, debug=False):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(N*N, N)
    # 各セルに数値は1つ?
    for i in range(N*N):
        s = 0
        for n in range(N):
            s += b[i][n]
        if s != 1:
            if debug:
                print(f'!: セルの中の数値が1つでない{i:3d}:{b[i]}')
            return False
    # 各ブロックに重複する数値はない?
    i0j0 = self.block_ij()
    for i in i0j0:
        for j in i0j0:
            for n in range(N):
                s = 0
                for x in range(M):
                    for y in range(M):
                        bidx = (i+x)*N + j+y
                        s += b[bidx][n]
                if s != 1:
                    if debug:
                        print(f'!: ブロック内で数値が重複')
                    return False
    #
    for n in range(N):
        # 各行に重複する数値はない?
        for i in range(N):
            s = 0
            for j in range(N):
```

```

        bidx = i * N + j
        s += b[bidx][n]
    if s != 1:
        if debug:
            print(f'!: 行で数値が重複')
        return False
    # 各列に重複する数値はない?
    for j in range(N):
        s = 0
        for i in range(N):
            bidx = i * N + j
            s += b[bidx][n]
        if s != 1:
            if debug:
                print(f'!: 列で数値が重複')
            return False

    #
    return True

def check2(self, a, debug=False):
    N, M, S, _ = self.get_param()
    b = np.array(a).reshape(N, N)
    # 既定値は正しい?
    for a in self.required:
        IJX = a.split(',')
        if b[int(IJX[0])][int(IJX[1])] != int(IJX[2]):
            if debug:
                print(f'!: 既定値が違う: ({IJX[0]},{IJX[1]}){IJX[2]} != {b[int(IJX[0])][int(IJX[1])]}')
            return False
    # 各行の数値の和はS?
    for i in range(N):
        s = 0
        for j in range(N):
            s += b[i][j]
        if s != S:
            if debug:
                print(f'!: 行の総和={s}!={S}')
            return False
    # 各列の数値の和はS?
    for j in range(N):
        s = 0
        for i in range(N):
            s += b[i][j]
        if s != S:
            if debug:
                print(f'!: 列の総和={s}!={S}')
            return False
    # 各ブロックの数値の和はS?
    i0j0 = self.block_ij()
    for i in i0j0:
        for j in i0j0:
            s = 0
            for x in range(M):

```

```

        for y in range(M):
            #print(i+x, j+y)
            s += b[i+x][j+y]
        if s != S:
            if debug:
                print(f'!!: ブロック内の総和={s}!={S}')
            return False
    #
    return True

def decode(self, a):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(N**2, N)
    #print(b)
    mat = []
    for v in b:
        num = 0
        for i, u in enumerate(v):
            if u==1:
                num = i+1
        mat.append(num)
    return mat

def print_shape(self):
    for i in range(self.N):
        print(f'{i}: ', end='\t')
        for j in range(self.N):
            for a in self.required:
                IJX = a.split(',')
                if i==int(IJX[0]) and j==int(IJX[1]):
                    print(int(IJX[2]), end=' ')
                    break
            else:
                print('_', end=' ')
        print()

```

1.2.9 main

```

if __name__ == '__main__':
    KiteiF = 'data9alt.txt'
    M = 3
    sudoku = NumberPlace(M, KiteiF)
    sudoku.print_shape()
    #lagrange1 = 40.0          # 数値に重複なし
    #lagrange2 = 5.4          # 行、列、ブロック、で重複なし
    #lagrange3 = 0.0          # 和はS
    #lagrange4 = 5.1          # 既定セル
    lagrange1 = 1.0           # 数値に重複なし
    lagrange2 = lagrange1 * 0.653 # 行、列、ブロック、で重複なし 0.018
    lagrange3 = lagrange1 * 0.0   # 和はS
    lagrange4 = lagrange1 * 0.05  # 既定セル

```

```

Q = sudoku.f(lagrange1, lagrange2, lagrange3, lagrange4)
num_reads = 100
sampleset = sudoku.solve(Q, num_reads)
ans = sudoku.result(sampleset)
print(*ans, sep='\n')
#
debug = True
for sample in sampleset.record['sample']:
    if sudoku.check1(sample, debug):
        if debug: print('check1 Passed!')
        a = sudoku.decode(sample)
        if sudoku.check2(a, debug):
            if debug: print('check2 Passed!')
            print(np.array(a).reshape(M*M, M*M))
            print()

```

1.3 実行結果

→ 4x4 の場合

既定セルの値：data4.txt

```

0, 1, 1
1, 0, 2
3, 0, 4
3, 3, 1

```

期待したのは次の状態

```

[3, 1, 4, 2]
[2, 4, 1, 3]
[1, 2, 3, 4]
[4, 3, 2, 1]

```

実行結果は次の通り。惜しいが正解ではない

```

0: _ 1 _ _
1: 2 _ _ _
2: _ _ _ _
3: 4 _ _ 1

```

```

[3, 1, 4, 2]
[2, 4, 1, 4]
[1, 2, 4, 3]
[4, 3, 2, 1]

```

何度かやっていると、辛うじて期待するものが出ることもあるが、、

```

[[3 1 4 2]
 [2 4 1 3]
 [1 3 2 4]
 [4 2 3 1]]

```

これで解けていると言えるのか？

(lagrange1 ~ lagrange4 の値のバランスがデリケートだ)

→ 9x9 の場合

一度も解に至っていない

```

0: 8 _ _ 2 _ _ _ 7 _
1: _ _ 1 _ 3 _ 5 _ _
2: _ 9 _ _ _ 6 _ _ _
3: _ _ _ 5 _ _ _ 2 _
4: 7 _ _ _ 8 _ _ _ 1
5: _ 6 _ _ _ 2 _ _ _
6: _ _ _ 7 _ _ _ 4 _
7: _ _ 3 _ 1 _ 8 _ _
8: _ 2 _ _ _ 9 _ _ 6

```

```

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

```

!: 列で数値が重複

!: セルの中の数値が1つでない 1: [1 0 0 0 1 0 0 0 0]

!: セルの中の数値が1つでない 42: [0 1 0 0 0 1 0 0 0]

!: セルの中の数値が1つでない 5: [0 1 0 0 0 0 1 0 0]

!: 列で数値が重複

!: 行で数値が重複

!: 行で数値が重複

check1 Passed!

!: 既定値が違う: (0, 0) 8!=6

!: 列で数値が重複

!: セルの中の数値が1つでない 0: [0 1 0 1 0 0 0 0 0]

!: 行で数値が重複
 !: セルの中の数値が1つでない 25:[0 0 0 1 1 0 0 0 0]
 !: セルの中の数値が1つでない 20:[0 0 0 1 0 0 0 1 0]
 !: セルの中の数値が1つでない 31:[0 0 0 0 1 0 1 0 0]
 !: セルの中の数値が1つでない 1:[1 0 1 0 0 0 0 0 0]
 !: セルの中の数値が1つでない 39:[1 0 0 0 0 0 0 1 0]
 !: 行で数値が重複
 !: 行で数値が重複
 !: セルの中の数値が1つでない 4:[0 0 0 0 1 0 0 1 0]
 !: セルの中の数値が1つでない 71:[0 0 0 1 0 0 0 1 0]
 !: セルの中の数値が1つでない 52:[0 0 0 0 1 0 0 0 1]
 !: 行で数値が重複
 !: セルの中の数値が1つでない 22:[0 1 0 0 0 0 1 0 0]
 !: 列で数値が重複
 !: 列で数値が重複
 !: セルの中の数値が1つでない 36:[1 0 0 0 1 0 0 0 0]
 !: セルの中の数値が1つでない 24:[1 0 0 0 0 0 1 0 0]
 !: セルの中の数値が1つでない 20:[0 0 0 1 1 0 0 0 0]
 !: セルの中の数値が1つでない 71:[0 0 0 0 0 0 1 1 0]
 !: 行で数値が重複

1.4 プログラムの全体

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

class NumberPlace:
    def __init__(self, M=2, FileN='data.txt'):
        self.M = M
        self.N = M * M
        S = 0
        for i in range(1, self.N+1):
            S += i
        self.S = S
        with open(FileN, 'r') as f:
            self.required = f.read().splitlines()
        self.idx = {}
        k = 0
        for i in range(self.N):
            for j in range(self.N):
                for n in range(self.N):
                    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 block_ij(self):
    i0j0 = []
    for i in range(self.M):
        i0j0.append(self.M*i)
    return i0j0

def sub1(self, i, j, L, Q):
    N, _, _, idx = self.get_param()
    for n1 in range(N):
        Q[(idx[(i, j, n1)], idx[(i, j, n1)])] -= 2.0 * L
        for n2 in range(N):
            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 sub2R(self, i, n, L, Q):
    N, _, _, idx = self.get_param()
    for j1 in range(N):
        Q[(idx[(i, j1, n)], idx[(i, j1, n)])] -= 2.0 * L
        for j2 in range(N):
            Q[(idx[(i, j1, n)], idx[(i, j2, n)])] += 1.0 * L

def sub2C(self, j, n, L, Q):
    N, _, _, idx = self.get_param()
    for i1 in range(N):
        Q[(idx[(i1, j, n)], idx[(i1, j, n)])] -= 2.0 * L
        for i2 in range(N):
            Q[(idx[(i1, j, n)], idx[(i2, j, n)])] += 1.0 * L

def f2(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        for n in range(N):
            self.sub2R(i, n, L, Q)
    for j in range(N):
        for n in range(N):
            self.sub2C(j, n, L, Q)
    return Q

def sub3(self, i0, j0, n, L, Q):
    N, M, _, idx = self.get_param()
    for x1 in range(M):
        for y1 in range(M):
            Q[(idx[(i0 + x1, j0 + y1, n)], idx[(i0 + x1, j0 + y1, n)])] -= 2.0

```

* L

```

        for x2 in range(M):
            for y2 in range(M):
                Q[(idx[(i0 + x1, j0 + y1, n)], idx[(i0 + x2, j0 + y2, n)])]
+= 1.0 * L

def f3(self, L, Q):
    N, _, _, idx = self.get_param()
    i0j0 = self.block_ij()
    for i0 in i0j0:
        for j0 in i0j0:
            for n in range(N):
                self.sub3(i0, j0, n, L, Q)
    return Q

def sub4R(self, i, L, Q):
    N, _, S, idx = self.get_param()
    for j1 in range(N):
        for n1 in range(N):
            Q[(idx[(i, j1, n1)], idx[(i, j1, n1)])] -= 2.0 * (n1+1) * S * L
        for j2 in range(N):
            for n2 in range(N):
                Q[(idx[(i, j1, n1)], idx[(i, j2, n2)])] += (n1+1) * (n2+1)
* L

def sub4C(self, j, L, Q):
    N, _, S, idx = self.get_param()
    for i1 in range(N):
        for n1 in range(N):
            Q[(idx[(i1, j, n1)], idx[(i1, j, n1)])] -= 2.0 * (n1+1) * S * L
        for i2 in range(N):
            for n2 in range(N):
                Q[(idx[(i1, j, n1)], idx[(i2, j, n2)])] += (n1+1) * (n2+1)
* L

def f4(self, L, Q):
    N, _, _, _ = self.get_param()
    for i in range(N):
        self.sub4R(i, L, Q)
    for j in range(N):
        self.sub4C(j, L, Q)
    return Q

def sub5(self, i0, j0, L, Q):
    N, M, S, idx = self.get_param()
    for x1 in range(M):
        for y1 in range(M):
            for n1 in range(N):
                Q[(idx[(i0+x1, j0+y1, n1)], idx[(i0+x1, j0+y1, n1)])] -= 2.0 *
(n1+1) * S * L
            for x2 in range(M):
                for y2 in range(M):
                    for n2 in range(N):
                        Q[(idx[(i0+x1, j0+y1, n1)], idx[(i0+x2, j0+y2, n2
)]] += (n1+1) * (n2+1) * L

```

```

def f5(self, L, Q):
    i0j0 = self.block_ij()
    for i0 in i0j0:
        for j0 in i0j0:
            self.sub5(i0, j0, L, Q)
    return Q

def f6(self, I, J, X, L, Q):
    N, _, _, idx = self.get_param()
    for n1 in range(N):
        Q[(idx[(I, J, n1)], idx[(I, J, n1)])] -= 2 * (n1+1) * X * L
        for n2 in range(N):
            Q[(idx[(I, J, n1)], idx[(I, J, n2)])] += (n1+1) * (n2+1) * L
    return Q

def f(self, lagrange1=1.0, lagrange2=1.0, lagrange3=1.0, lagrange4=1.0):
    Q = defaultdict(lambda: 0)
    _ = self.f1(lagrange1, Q)
    _ = self.f2(lagrange2, Q)
    _ = self.f3(lagrange2, Q)
    if 0.0 < lagrange3:
        _ = self.f4(lagrange3, Q)
        _ = self.f5(lagrange3, Q)
    for a in self.required:
        IJX = a.split(',')
        _ = self.f6(int(IJX[0]), int(IJX[1]), int(IJX[2]), lagrange4, 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, _, _, 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):
                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, debug=False):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(N*N, N)
    # 各セルに数値は1つ?
    for i in range(N*N):
        s = 0
        for n in range(N):
            s += b[i][n]
        if s != 1:
            if debug:
                print(f'!: セルの中の数値が1つでない{i:3d}:{b[i]}')
            return False
    # 各ブロックに重複する数値はない?
    i0j0 = self.block_ij()
    for i in i0j0:
        for j in i0j0:
            for n in range(N):
                s = 0
                for x in range(M):
                    for y in range(M):
                        bidx = (i+x)*N + j+y
                        s += b[bidx][n]
                if s != 1:
                    if debug:
                        print(f'!: ブロック内で数値が重複')
                    return False
    #
    for n in range(N):
        # 各行に重複する数値はない?
        for i in range(N):
            s = 0
            for j in range(N):
                bidx = i * N + j
                s += b[bidx][n]
            if s != 1:
                if debug:
                    print(f'!: 行で数値が重複')
                return False
        # 各列に重複する数値はない?
        for j in range(N):
            s = 0
            for i in range(N):
                bidx = i * N + j
                s += b[bidx][n]
            if s != 1:

```

```

        if debug:
            print(f'!: 列で数値が重複')
        return False

#
return True

def check2(self, a, debug=False):
    N, M, S, _ = self.get_param()
    b = np.array(a).reshape(N, N)
    # 既定値は正しい?
    for a in self.required:
        IJX = a.split(',')
        if b[int(IJX[0])][int(IJX[1])] != int(IJX[2]):
            if debug:
                print(f'!: 既定値が違う: ({IJX[0]}, {IJX[1]}) {IJX[2]} != {b[int(IJX[0])][int(IJX[1])]}')
            return False
    # 各行の数値の和はS?
    for i in range(N):
        s = 0
        for j in range(N):
            s += b[i][j]
        if s != S:
            if debug:
                print(f'!: 行の総和={s}!={S}')
            return False
    # 各列の数値の和はS?
    for j in range(N):
        s = 0
        for i in range(N):
            s += b[i][j]
        if s != S:
            if debug:
                print(f'!: 列の総和={s}!={S}')
            return False
    # 各ブロックの数値の和はS?
    i0j0 = self.block_ij()
    for i in i0j0:
        for j in i0j0:
            s = 0
            for x in range(M):
                for y in range(M):
                    #print(i+x, j+y)
                    s += b[i+x][j+y]
            if s != S:
                if debug:
                    print(f'!: ブロック内の総和={s}!={S}')
                return False
    #
    return True

def decode(self, a):
    N, M, _, _ = self.get_param()
    b = np.array(a).reshape(N**2, N)

```

```

        #print(b)
        mat = []
        for v in b:
            num = 0
            for i, u in enumerate(v):
                if u==1:
                    num = i+1
            mat.append(num)
        return mat

def print_shape(self):
    for i in range(self.N):
        print(f' {i}: ', end='\t')
        for j in range(self.N):
            for a in self.required:
                IJX = a.split(',')
                if i==int(IJX[0]) and j==int(IJX[1]):
                    print(int(IJX[2]), end=' ')
                    break
            else:
                print('_', end=' ')
        print()

if __name__ == '__main__':
    KiteiF = 'data9alt.txt'
    M = 3
    sudoku = NumberPlace(M, KiteiF)
    sudoku.print_shape()
    #lagrange1 = 40.0      # 数値に重複なし
    #lagrange2 = 5.4      # 行、列、ブロック、で重複なし
    #lagrange3 = 0.0      # 和はS
    #lagrange4 = 5.1      # 既定セル
    lagrange1 = 1.0       # 数値に重複なし
    lagrange2 = lagrange1 * 0.653      # 行、列、ブロック、で重複なし 0.018
    lagrange3 = lagrange1 * 0.0        # 和はS
    lagrange4 = lagrange1 * 0.05       # 既定セル
    Q = sudoku.f(lagrange1, lagrange2, lagrange3, lagrange4)
    num_reads = 100
    sampleset = sudoku.solv(Q, num_reads)
    ans = sudoku.result(sampleset)
    print(*ans, sep='\n')
    #
    debug = True
    for sample in sampleset.record['sample']:
        if sudoku.check1(sample, debug):
            if debug: print('check1 Passed!')
            a = sudoku.decode(sample)
            if sudoku.check2(a, debug):
                if debug: print('check2 Passed!')
                print(np.array(a).reshape(M*M, M*M))
                print()

```

プログラム 1.1 数独

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://amplify.fixstars.com/ja/techresources/application/sudoku/>
- [3] <https://zenn.dev/airev/articles/airev-quantum-02>