

情報科学実験 3・演習 3
ゼミレポート課題

2023 年 12 月 30 日 (土)

6321120

横溝 尚也

1 課題 1

1.1 課題内容

デザインが与えられたとき、BIBD の条件を満たしているかどうかを確認するプログラム

1.2 アルゴリズムについて

BIBD であるか判定するために以下の条件を満たすかプログラムで実装した。

- 入力されたブロックのブロックサイズがすべて一定であること
- 入力されたブロック集合に含まれる任意の要素 x, y に対して、ブロックに x, y が含まれるブロック数が一定であること
- 最終的に求めたパラメータが $v > k \geq 2$ を満たすこと

入力方式に関してはファイルの中身を変えることなく標準入力によってできるだけ簡易的に入力を行い、入力した集合族に関してプログラムが正常に動く、また不正な入力に関してはエラー内容によって適切に除外できるようにする。

会合数の計算には入力されたデザインの結合行列 M を計算し、 $M \times M^T$ の以下の性質を利用して会合数の計算を行った。

行列 M の要素 $m_{i,j}$ に対して

$$\sum_{h=1}^b m_{i,h} m_{j,h} = \begin{cases} r & (i = j) \\ \lambda & (i \neq j) \end{cases}$$

が成立。これより $M \times M^T$ の非対角成分を抽出して上記の条件を満たすか判定する。

1.3 プログラムの内容

以下がソースコードである。説明の便宜上、行数を振ってある

プログラム 1 check_BIBD.py

```
1 #課題 (1)
2 #部分集合族を入力した際にそのデザインがであるかの判定 BIBD
3 import numpy as np
4
5 #すべてのブロックサイズが一定であるか判定
6 def check_blocksize(subset_family):
7     block_size = len(subset_family[0])
8     for i in range(len(subset_family)):
9         if block_size != len(subset_family[i]):
10             return False ,block_size
11     return True ,block_size
12
13
```

```

14 #点集合の要素 X と要素数 v を計算
15 def cal_pointnum(subset_family):
16     unique_points = [] #リストの初期化
17
18     for subset in subset_family:
19         for point in subset:
20             if point not in unique_points:
21                 unique_points.append(point)
22
23     return unique_points, len(unique_points)
24
25 #結合行列の作成
26 def make_incidenceMatrix(subset_family):
27     X, v = cal_pointnum(subset_family) #点集合と点の数を格納
28     b = len(subset_family) #ブロック数
29     incidence_matrix = [[0] * b for _ in range(v)] #結合行列の初期化
30
31     #結合行列にデザインの情報を格納
32     for point_index in range(len(X)):
33         for block_index in range(len(subset_family)):
34             if(X[point_index] in subset_family[block_index]):
35                 incidence_matrix[point_index][block_index] = 1
36
37     #print(np.array(incidence_matrix))
38     return np.array(incidence_matrix)
39
40
41 #会合数  $\lambda$  が常に一定であるか判定
42 def cal_numberOfMeeting(subset_family):
43     M = make_incidenceMatrix(subset_family) #結合行列を M とする
44     processed_M = M @ M.T #M * M 転置を計算
45     #print(processed_M)
46
47     #MM_t[i][j]がi=ならば j,r,i!=ならば  $\lambda$  となるか判定 j
48     r = processed_M[1][1] #出現回数の初期化 r
49     meetings = processed_M[1][2] #会合数  $\lambda$  の初期化
50     check_bool = [True, True] #出現回数と会合数の判定の初期化
51
52     for i in range(len(processed_M)):
53         for j in range(len(processed_M[i])):
54             if i == j: # 一致しているか判定 r
55                 if r != processed_M[i][j]:
56                     check_bool[0] = False
57             else:
58                 if meetings != processed_M[i][j]:
59                     check_bool[1] = False

```

```

60         return check_bool, r, meetings
61
62 #入力を受け付ける処理
63 def input_array():
64     print("次元配列を入力してください。2")
65     print("要素をスペースで区切り、行ごとにを押してください。Enter")
66     print("最後にをつけて','を押すと入力終了します Enter.")
67
68     input_lines = []
69     while True:
70         try:
71             line = input()
72             if line.endswith(',') :
73                 line = line.rstrip(',')
74                 input_lines.append(line)
75                 break
76             input_lines.append(line)
77         except EOFError:
78             break
79
80     # 入力されたテキストを 3 次元配列に変換
81     two_dimensional_array = []
82     for line in input_lines:
83         row = [int(x) for x in line.split()]
84         two_dimensional_array.append(row)
85
86     return two_dimensional_array
87
88
89 #メイン関数
90 def check_BIBD(subset_family):
91     all_parameter = [0] * 5 #パラメータの初期化
92
93     #ブロックサイズについて
94     isBlocksize = check_blocksize(subset_family)[0]
95     if not isBlocksize:
96         return False, 1
97     #ブロックサイズの格納
98     all_parameter[3] = check_blocksize(subset_family)[1]
99
100    #点集合について
101    all_parameter[0] = cal_pointnum(subset_family)[1]
102    X = cal_pointnum(subset_family)[0]
103
104    #会合数について
105    isMeetings = cal_numberOfMeeting(subset_family)[0][1]

```

```

106     if not isMeetings:
107         return False, 2
108     all_parameter[4] = cal_numberOfMeeting(subset_family)[2]
109     all_parameter[2] = cal_numberOfMeeting(subset_family)[1]
110
111     #ブロック数計算
112     b = len(subset_family)
113     all_parameter[1] = b
114     #v>k>=2 の確認
115     if not (all_parameter[0] > all_parameter[3] >= 0):
116         return False, 3
117
118     return True, all_parameter[0], all_parameter[1], all_parameter[2], all_parameter
        [3], all_parameter[4]
119
120 def make_output(result):
121     if result[0]:
122         print("入力されたデザインは BIBD□の条件を満たしています。以下がパラメータです")
123         print("点の個数:" ,result[1],"ブロック数:" ,result[2],
124             "出現回数:",result[3], "ブロックサイズ:", result[4],
125             "会合数:", result[5])
126     else:
127         if result[1] == 1:
128             print("ブロックサイズが一定でないため BIBD□ではありません")
129         elif result[1] == 2:
130             print("会合数が一定でないため BIBD□ではありません")
131         elif result[1] == 3:
132             print("v>k>=2□を満たしていません")
133         else:
134             print("何かエラーが起きてます")
135
136
137 #test:check_blocksize()
138 #print(check_blocksize(test1))
139
140 #test:cal_pointnum()
141 #print(cal_pointnum(test1))
142
143 #test:cal_numberOfMeeting()
144 #print(cal_numberOfMeeting(test2))
145
146 #test:cal_numberOfMeeting()
147 #print(cal_numberOfMeeting(test2))
148
149 #test:cal_numberOfMeeting()
150 #print(cal_numberOfMeeting(test2))

```

```

151
152 #test:check_BIBD()
153 #check_BIBD(test2)
154
155
156 if __name__ == "__main__":
157     subset_family = input_array()
158     #print(subset_family)
159     result = check_BIBD(subset_family)
160     make_output(result)

```

1.4 プログラムの説明

初めに今回のプログラムでは入力された集合族を配列に変換してその配列を様々な条件に合うか判定する流れである。ただし、課題 2,3 において入力されたブロックの要素 (点) が int 型であるときと同様に、ペアであるときにもプログラムが正常に作動するように以下のように入力データを加工した。

プログラム 2 入力データ subset_family の表現

```

1 subset_family = [[0], [1], [2], [4], [5], [8], [10]], [[1], [2], [3], [5], [6],
    [9], [11]], [[2], [3], [4], [6], [7], [10], [12]], [[3], [4], [5], [7], [8],
    [11], [13]], [[4], [5], [6], [8], [9], [12], [14]], [[5], [6], [7], [9],
    [10], [13], [0]], [[6], [7], [8], [10], [11], [14], [1]], [[7], [8], [9],
    [11], [12], [0], [2]], [[8], [9], [10], [12], [13], [1], [3]], [[9], [10],
    [11], [13], [14], [2], [4]], [[10], [11], [12], [14], [0], [3], [5]], [[11],
    [12], [13], [0], [1], [4], [6]], [[12], [13], [14], [1], [2], [5], [7]],
    [[13], [14], [0], [2], [3], [6], [8]], [[14], [0], [1], [3], [4], [7], [9]]]
2
3
4 subset_family = [[[0, 0], [0, 1], [1, 0], [1, 2], [2, 0], [2, 3]], [[0, 1], [0,
    2], [1, 1], [1, 3], [2, 1], [2, 0]], [[0, 2], [0, 3], [1, 2], [1, 0], [2, 2],
    [2, 1]], [[0, 3], [0, 0], [1, 3], [1, 1], [2, 3], [2, 2]], [[1, 0], [1, 1],
    [2, 0], [2, 2], [3, 0], [3, 3]], [[1, 1], [1, 2], [2, 1], [2, 3], [3, 1], [3,
    0]], [[1, 2], [1, 3], [2, 2], [2, 0], [3, 2], [3, 1]], [[1, 3], [1, 0], [2,
    3], [2, 1], [3, 3], [3, 2]], [[2, 0], [2, 1], [3, 0], [3, 2], [0, 0], [0,
    3]], [[2, 1], [2, 2], [3, 1], [3, 3], [0, 1], [0, 0]], [[2, 2], [2, 3], [3,
    2], [3, 0], [0, 2], [0, 1]], [[2, 3], [2, 0], [3, 3], [3, 1], [0, 3], [0,
    2]], [[3, 0], [3, 1], [0, 0], [0, 2], [1, 0], [1, 3]], [[3, 1], [3, 2], [0,
    1], [0, 3], [1, 1], [1, 0]], [[3, 2], [3, 3], [0, 2], [0, 0], [1, 2], [1,
    1]], [[3, 3], [3, 0], [0, 3], [0, 1], [1, 3], [1, 2]]]

```

上記のように入力要素が int 型であるとき一つのブロックを 1 次元配列で表現、集合族 2 次元配列として表現すればよいが、差集合の要素がペアであるとき要素を 1 次元配列、ブロックを 2 次元配列、ブロックの集合を 3 次元配列として表現する必要があるため入力データは 3 次元配列で統一的に扱う。

1～11 行目

入力されたブロックの濃度 (ブロックサイズ) が一定であるか判定する関数 `check_blocksize()` についての記述である。引数に集合族を 3 次元配列として受け取り、各集合の配列の長さ `len(subset_family[i])` に対して `i` を変数としてループし、異なるブロックサイズのものが存在した時点で `False`、すべての集合の濃度が一定であれば最終的に `True` とブロックサイズを返す。

14～23 行目

入力された要素 (点) から重複を許さない点集合の計算と、その要素数を計算する関数 `cal_pointnum()` についての記述である。引数として部分集合族を 3 次元配列の形で受け取り、全要素 (点) に対して初期化されたリスト `unique_points` の要素に対象の要素が含まれていなかったらリストに追加する操作をループする。最後に計算した点集合とその配列の長さ (点の個数) を返す。

25～38 行目

会合数 λ を求めるためにデザインの結合行列を求める関数 `make_incidenceMatrix()` についての記述である。先述した `cal_pointnum(subset_family)` で点集合 X と点の個数 v を格納、入力データからブロック数を意味する集合族の配列の長さを b とする。また、デザインの結合行列は $v \times b$ 行列であるので全要素 0 である $v \times b$ 行列を `incidence_matrix` として初期化する。ある点 X があるブロックの中に含まれているかを判定する走査を (点の個数) \times (ブロックの数) 回ループする。返戻地は 0,1 の要素で構成された結合行列を返す。このとき、出力で見やすくするために `numpy` を用いた。

41～60 行目

結合行列を用いて会合数を求め、一定であるか判定する関数 `cal_numberOfMeeting()` についての記述である。先ほどの `make_incidenceMatrix(subset_family)` を呼び出し、結合行列 M に対して、 $M \times M^T$ を計算する。52～59 行目で非対角成分の値がすべて一致していれば `True` とその会合数を返し、異なるものがあれば `False` を返す。ループの際に対角成分 ($i=j$) の時は処理をスキップしている。

62～86 行目

入力を受け付け処理をする関数 `input_array()` についての記述である。ここは課題の趣旨とはそれるため説明は省略する。工夫点としては入力の簡潔化のため、配列の形で入力するのではなく、`Space` と `Enter` を用いて 2 次元配列のように集合族を入力できるようにした。次節で実行結果のスクリーンショットを添付したので入力方法に関してはそちらを参照していただきたい。

89～118 行目

今まで実装したサブ関数を使用し、デザインが BIBD であるか判定するメイン関数 `check_BIBD()` についての記述である。パラメータの数値を `all_parameter` に格納しながら判定条件を満たしていれば次の条件に進み、満たしていない条件が存在した時点でこの関数の実行を終了するためにそれぞれ `return False, 1` (または `2,3`) を返す。すべての条件を満たしたときは `return True, all_parameter` を返す。

120～160 行目

134 行目までがメイン関数の返戻地に対応して BIBD であるのか否かと BIBD でないときはどの条件が満たしていないか出力する。160 行目まではテスト内容と他モジュールからの実行の際に実行しないようブロックした。

1.5 実行結果

以下がゼミで使用している教材に書かれている BIBD の例の実行と BIBD でないときの実行例である。入力方法としては 1 行にスペース区切りでブロックを入力し、次のブロックの記述の際は改行して入力する。最後にセミコロンの入力で標準入力を終了し、プログラムが進行する流れである。

```
Python: check_BIBD + 〇 〇 ... ^ x
問題 出力 デバッグコンソール ターミナル ポート GITLENS
● PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/ch
eck_BIBD.py
2次元配列を入力してください。
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力終了します。
1 2 3
1 4 5
1 6 7
2 4 6
2 5 7
3 4 7
3 5 6;
入力されたデザインはBIBDの条件を満たしています。以下がパラメータです
点の個数X: 7 ブロック数: 7 出現回数: 3 ブロックサイズ: 3 会合数: 1
○ PS C:\Users\81809\Documents\ALL_CODE>
```

图 1 p1.Ex1.1 A(7,3,1)-BIBD

```
問題 出力 デバッグコンソール ターミナル ポート GITLINS Python: check_BIBD + 〇 〇 ... ^ x
PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/check_BIBD.py
2次元配列を入力してください。
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力が終わります。
1 2 3
4 5 6
7 8 9
1 4 7
2 5 8
3 6 9
1 5 9
2 6 7
3 4 8
1 6 8
2 4 9
3 5 7;
入力されたデザインはBIBDの条件を満たしています。以下がパラメータです
この個数X: 9 ブロック数: 12 出現回数: 4 ブロックサイズ: 3 会合数: 1
PS C:\Users\81809\Documents\ALL_CODE>
```

图 2 p1.Ex1.2 A(9,3,1)-BIBD


```
問題 出力 デバッグコンソール ターミナル ポート GITLENS Python: check_BIBD + - [ ] 倉 ... ^ x
PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/ch
eck_BIBD.py
2次元配列を入力してください。
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力終了します。
0 1 2 3
0 1 4 5
0 2 4 6
0 3 7 8
0 5 7 9
0 6 8 9
1 2 7 8
1 3 6 9
1 4 7 9
1 5 6 8
2 3 5 9
2 4 8 9
2 5 6 7
3 4 5 8
3 4 6 7;
入力されたデザインはBIBDの条件を満たしています。以下がパラメータです
点の個数x: 10 ブロック数: 15 出現回数: 6 ブロックサイズ: 4 会合数: 2
PS C:\Users\81809\Documents\ALL_CODE>
```

図3 p1.Ex1.3 A(10,4,2)-BIBD

```
問題 出力 デバッグコンソール ターミナル ポート GITLENS Python: check_BIBD + - [ ] 倉 ... ^ x
PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/ch
eck_BIBD.py
2次元配列を入力してください。
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力終了します。
1 2 3
1 2 3 3;
ブロックサイズが一定でないためBIBDではありません
PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/ch
eck_BIBD.py
2次元配列を入力してください。
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力終了します。
1 2 3
2 3 4
3 4 5;
会合数が一定でないためBIBDではありません
PS C:\Users\81809\Documents\ALL_CODE>
```

図4 不適な入力やデザインがBIBDでないとき

2 課題2

2.1 課題内容

集合が与えられたとき、差集合の条件を満たしているかどうかを確認するプログラム

2.2 アルゴリズムの説明

差集合か判定するための条件は以下である。

- 入力された集合 D , 有限加法可換群 G に対して $D \subseteq G$ が成立
- $|D| = k$
- 任意の $x, y \in D (x \neq y)$ に対して、 $x - y$ はすべての $G \setminus \{0\}$ の要素がちょうど λ 回現れる。

上記を満たす集合であるかを判定する。差集合の定義として、 G は有限加法可換群であるが、授業内で G

は法 n に関する剰余類環 \mathbb{Z}_n しか扱っておらず、これが慣例でもあるため G はこれに限定する。
集合のほかに位数 n を入力し、要素が剰余類環の直積 $\mathbb{Z}_4 \times \mathbb{Z}_4$ にも対応できるプログラムを実装する。

2.3 プログラムの内容

プログラム 3 check_differenceSet.py

```
1 #課題 (2)
2 #差集合の条件を満たしているか判定
3 import numpy as np
4 from itertools import product
5
6
7 #入力を受け付ける処理
8 def input_array():
9     print("群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください")
10    print("最後にをつけて',';Enterを押すと入力が終了します")
11    print("例:  $\mathbb{Z}_4 \times \mathbb{Z}_4$  ならば 4,4; と入力")
12
13    input_lines = []
14    while True:
15        try:
16            line = input()
17            if line.endswith(';'):
18                line = line.rstrip(';')
19                input_lines.append(line)
20                break
21            input_lines.append(line)
22        except EOFError:
23            break
24
25    order = [int(x) for x in input_lines[0].split()]
26
27    if len(order) >= 2:
28        print("集合を入力してください")
29        print("要素をスペースで区切り、行ごとに Enterを押してください。")
30        print("最後にをつけて',';Enterを押すと入力終了します.")
31
32    input_lines_sets = []
33    while True:
34        try:
35            line_sets = input()
36            if line_sets.endswith(';'):
37                line_sets = line_sets.rstrip(';')
38                input_lines_sets.append(line_sets)
39                break
```

```

40         input_lines_sets.append(line_sets)
41     except ValueError:
42         print("無効な入力です")
43
44     # 入力されたテキストを 3 次元配列に変換
45     two_dimensional_array = []
46     for line in input_lines_sets:
47         row = [int(x) for x in line.split()]
48         two_dimensional_array.append(row)
49     #print(two_dimensional_array)
50     return two_dimensional_array, order
51 elif len(order) == 1:
52     print("集合を 1 行で入力してください")
53     print("要素をスペースで区切り、最後にをつけて;Enterを押すと入力が終わります")
54     try:
55         input_line = input()
56
57         # ';' が含まれているか確認
58         if ';' in input_line:
59             # ';' を取り除き、空白で分割して整数に変換
60             numbers = [[int(x)] for x in input_line.rstrip(';').split()]
61
62             return numbers, order
63         else:
64             print("入力が不正です")
65
66     except ValueError:
67         print("無効な入力です")
68     else:
69         print("正しく位数を入力してください")
70
71 #入力の不備や D が有限アーベル群の部分集合であるか判定
72 def check_input(differenceset, order):
73     dim = len(order)
74
75     #入力した位数の剰余類の要素であるか確認
76     for i in range(len(order)):
77         for element_index in range(len(differenceset)):
78             if order[i] < differenceset[element_index][i]:
79                 print("入力された集合の要素は指定された有限アーベル群の部分集合ではありません")
80                 return
81     if len(differenceset[element_index]) != dim:
82         print("入力された要素の中に指定した次元と異なるものがあります。")
83     return
84

```

```

85 #組み合わせ求める関数
86 def generate_combinations(input_list):
87     ranges = [range(x) if x > 1 else [0] for x in input_list]
88
89     # 全組み合わせを生成
90     combinations = list(product(*ranges))
91     #print(combinations)
92
93     # 生成した組み合わせを出力
94     result_list = []
95     for combo in combinations:
96         flat_combo = list(combo)
97         result_list.append(flat_combo)
98
99     return result_list
100
101
102 #入力された各剰余類の下で各集合要素の減算を行う
103 def cal_substruct_inMod(differenceset, order):
104     cal_substruct = [[0] * len(order) for _ in range(len(differenceset)*(len(
105         differenceset)-1))]
106     element_num = len(differenceset) * len(differenceset)
107     count = 0
108
109     for element_i in range(len(differenceset)):
110         for element_j in range(element_i+1,len(differenceset)):
111             # 2要素の減算 (与えられた方の中で)
112             for index in range(len(order)):
113                 quotient, result = divmod(differenceset[element_i][index] -
114                     differenceset[element_j][index], order[index])
115                 #print(differenceset[element_i][index], differenceset[element_j][
116                     index],result)
117                 cal_substruct[count][index] = result
118                 count += 1
119
120     #逆方向で減算
121     for element_i in range(len(differenceset)):
122         for element_j in range(element_i+1,len(differenceset)):
123             for index in range(len(order)):
124                 quotient, result = divmod(differenceset[element_j][index] -
125                     differenceset[element_i][index], order[index])
126                 #print(differenceset[element_i][index], differenceset[element_j][
127                     index],result)
128                 cal_substruct[count][index] = result
129                 count += 1

```

```

126
127     return cal_substruct
128
129 #計算した集合が入力した位数の有限加法アーベル群の部分集合か判定
130 def isSuperset(all_Element_G, cal_Element_G, order):
131     set1 = set(map(tuple, cal_Element_G))
132     set2 = set(map(tuple, all_Element_G))
133
134     return set1.issuperset(set2)
135
136
137 #v 個の要素が何回計算した集合 (cal_Element_G)に含まれているか計算
138 def count_meetings(all_Element_G, cal_Element_G, v):
139     count_meetings = [0] * (v-1) #0 元除く
140     for i in range(len(cal_Element_G)):
141         for j in range(len(all_Element_G)):
142             if cal_Element_G[i] == all_Element_G[j]:
143                 count_meetings[j] += 1
144
145     return count_meetings
146
147 def check_differenceset():
148     differenceset, order = input_array()
149     #print(differenceset,order)
150     check_input(differenceset, order)
151
152     #v の計算
153     v = np.prod(order)
154     #print(v)
155
156     #入力集合の濃度
157     cardinaryOfSet = len(differenceset)
158
159     #群 G の全要素を準備
160     all_Element_G = generate_combinations(order)
161     #差集合の定義より 0 元以外の要素が対象
162     #入力した位数の次元個の 0 を持つタプルを除いたアーベル群
163     zero_list = [0] * len(order)
164     all_Element_G_non0 = [row for row in all_Element_G if row != zero_list]
165     #print(all_Element_G_non0)
166
167     #入力された各剰余類の下で各集合要素の減算を行う
168     cal_Element_G = cal_substruct_inMod(differenceset, order)
169     #print(cal_Element_G)
170
171

```

```

172     #入力された集合 D の任意の要素 x,y に対して{x-y} が群  $G \setminus \{0\}$  の部分集合となっていない
    とき
173     if not isSuperset(all_Element_G_non0, cal_Element_G, order):
174         #λ 回 G の要素が出現するところか一回も出現していない要素がある
175         return False,0
176
177     #v 個の要素が何回計算した集合 (cal_Element_G)に含まれているか計算
178     meetings = count_meetings(all_Element_G_non0, cal_Element_G, v)
179     #print(meetings)
180
181     #λ を計算し、一定なら性質満たす
182     meet = meetings[0] #会合数の初期化
183     for i in range(len(meetings)):
184         if meetings[i] != meet: #会合数が一定でない
185             return False, 0
186
187     #differenceSet の性質をすべて満たしているときの出力
188     #課題3のために order も返す
189     return True, v,len(differenceset), meet, differenceset, order, all_Element_G
190
191
192 #実行
193 if __name__ == "__main__":
194     result = check_differenceset()
195     #print(result)
196
197     if result[0]:
198         print("入力された集合は",(result[1],result[2],result[3]),"-differenceSet□です")
199     else:
200         print("入力された集合は differenceSet□ではありません")

```

2.4 プログラムの説明

1～69 行目

入力に関する関数 `input_array()` に関する記述である。今回の課題の趣旨とは関係ないため解説は省略する。

71～83 行目

入力集合 D が G の部分集合であるか判定する関数 `check_input()` についての記述である。引数として位数と、入力された集合を 3 次元配列として受け取り、(各要素 `difference[element_index][i]`)<(位数 n) が全要素に対して成立しないものがあればエラー文を出力し、その時点で `return` する。また、要素の型が違うとき (例えば位数を 4 4; と入力し $\mathbb{Z}_4 \times \mathbb{Z}_4$ のとき、要素が 1 や、(3,3,3) であるとき) をキャッチし、エラー文を出力している。

86~99 行目

群 G を生成する関数 `generate_combinations()` についての記述である。群 G のすべての要素が存在するか確かめるために群 G を用意する。このとき入力された位数に対して全通りの組合せを求めている。この関数の返戻地は、位数の入力が 15; ならば `result_list=[0,1,2,3,4,,,14]`, 入力が 4 4; ならば `result_list=[[0,0],[0,1],[0,2],,,[3,3]]` となる。

102~127 行目

与えられた法の下で D の要素 x,y に対して $x-y$ の計算を行う関数 `cal_substruct_inMod()` についての記述である。入力された集合の配列 `differenceSet` の長さを k とすると、 $x-y$ の組み合わせは ${}_kP_2$ 通りである。そのため計算結果を格納する 2 次元配列 `cal_substruct` は [[一つの要素 (入力位数が複数の時は要素がタプルとなる)],[],, ${}_kP_2$ 個,,,] となる。`cal_substruct` に $x-y$ の計算結果が何回現れたかを格納するために適宜 `count` をインクリメントしていく。108,118 行目で $x-y,y-x$ どちらも行う必要があるため、大きなループ処理を 2 回行っている。最後に計算結果を格納した配列を返す。

129~134 行目

先述した関数 `generate_combinations()` で生成した群 G と、関数 `cal_substruct_inMod()` で全通りの $x-y$ を格納した配列を引数として受け取り (実際に引数で渡しているのはメイン関数内)、部分集合であるかを `set` を使用して判定し `True` か `False` を返す。

137~145 行目

先ほどの関数では部分集合であるかどうかのみ判定したがここでは群 G の要素が何回 $x-y$ で現れるか計算する関数 `count_meetings()` についての記述である。ここで差集合の定義にある $G \setminus 0$ の 0 元を除外している。返戻値は位数 3 ならば `[4,4,4]` のように群 G の濃度を長さとする 1 次元配列となる。

147~189 行目

今までの関数を実行するメイン関数 `check_meetings()` についての記述である。差集合となる条件を満たしていない時点で `False`、すべての条件をクリアしたときに `True` を返す。

192~200 行目

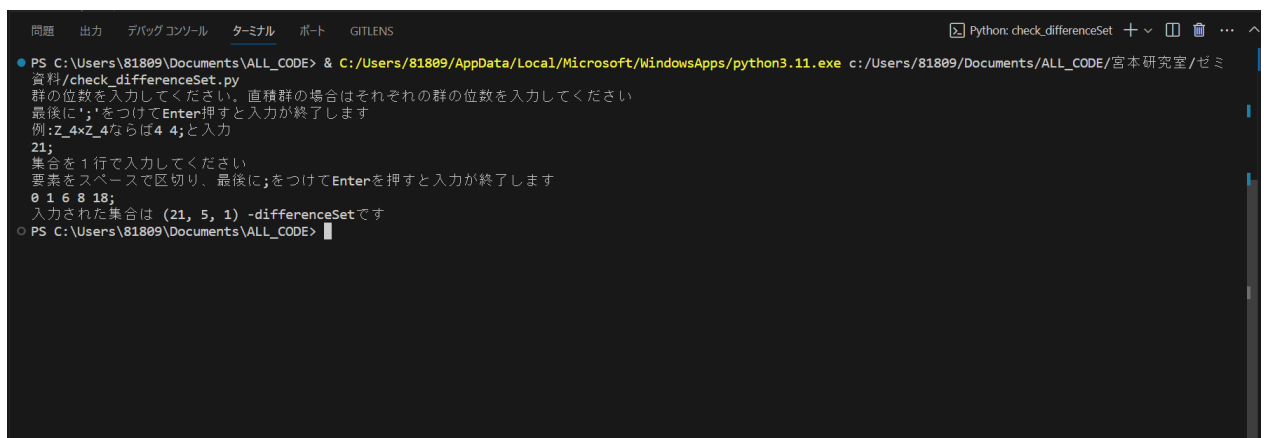
メイン関数を実行して得た真偽値をもとに最終出力を行う。ここは課題 1 と同様にファイルが直接呼び出されたときのみ実行するようにインポートはの際は実行をブロックしている。

2.5 実行結果

以下はゼミで使用している教材にある差集合の例と、p.21にある $(16,6,2)$ -differenceSet in \mathbb{Z}_{16} が存在しない例である ($(16,6,2)$ -differenceSet in $\mathbb{Z}_4 \times \mathbb{Z}_4$ から拡張して作成した)。

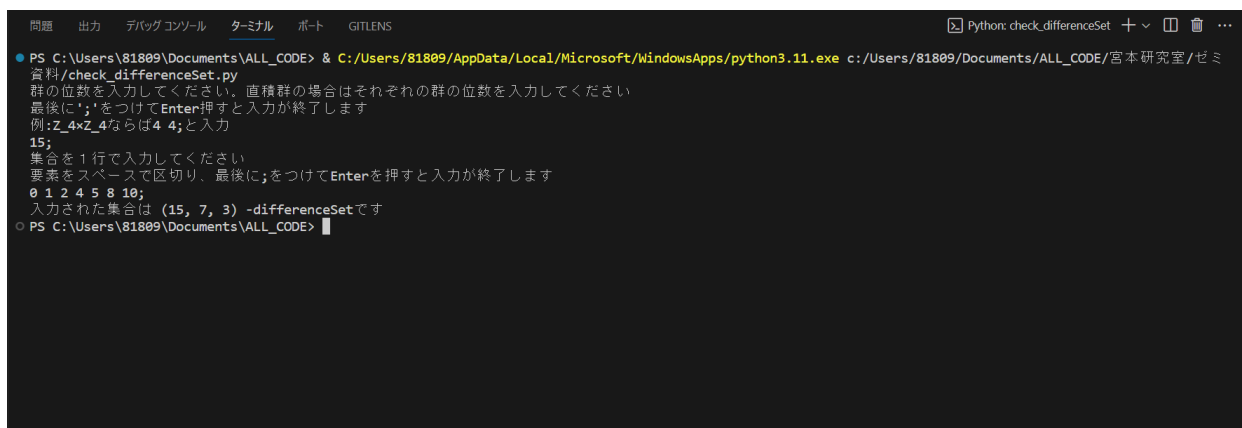
入力形式は初めに位数を入力する。このとき判定したい群 (剰余類環) が直積群? (剰余類環の直積で表されている環, 部分環を持つ環) であるときは 4×4 ; のように入力する。(差集合の定義では群、今回限定して考えているのは剰余類の環であるため群と環の表記が混ざってしまいました。もしこの辺りの解釈が誤っている場合、ご指摘をいただきたいです。)

その後調べたい集合を課題1と同様な形を入力する (直積群であるときとそうでないときで入力の形が少し変わっている。)



```
問題 出力 デバッグコンソール ターミナル ポート GITLENS Python: check_differenceSet + - 閉 ...
PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/check_differenceSet.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
21;
集合を1行で入力してください
要素をスペースで区切り、最後に;をつけてEnterを押すと入力終了します
0 1 6 8 10;
入力された集合は (21, 5, 1) -differenceSet です
PS C:\Users\81809\Documents\ALL_CODE>
```

図5 p14.Ex2.1 $A(21,5,1)$ -difference Set in $\mathbb{Z}/21$



```
問題 出力 デバッグコンソール ターミナル ポート GITLENS Python: check_differenceSet + - 閉 ...
PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ資料/check_differenceSet.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
15;
集合を1行で入力してください
要素をスペースで区切り、最後に;をつけてEnterを押すと入力終了します
0 1 2 4 5 8 10;
入力された集合は (15, 7, 3) -differenceSet です
PS C:\Users\81809\Documents\ALL_CODE>
```

図6 p14.Ex2.2 $A(15,7,3)$ -difference Set in $\mathbb{Z}/15$


```
問題 出力 デバッグ コンソール ターミナル ポート GITLENS Python: check_differenceSet + - [] ...
● PS C:\Users\81889\Documents\ALL_CODE> & C:/Users/81889/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81889/Documents/ALL_CODE/宮本研究室/ゼミ
資料/check_differenceSet.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
4 4;
集合を入力してください
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力終了します。
0 0
0 1
1 0
1 2
2 0
2 3;
入力された集合は (16, 6, 2) -differenceSetです
○ PS C:\Users\81889\Documents\ALL_CODE>
```

図 7 p14.Ex2.3 A(16,6,2)-difference Set in $Z/4 \times Z/4$

```
問題 出力 デバッグ コンソール ターミナル ポート GITLENS Python: check_differenceSet + - [] ...
● PS C:\Users\81889\Documents\ALL_CODE> & C:/Users/81889/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81889/Documents/ALL_CODE/宮本研究室/ゼミ
資料/check_differenceSet.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
16;
集合を 1 行で入力してください
要素をスペースで区切り、最後に;をつけてEnterを押すと入力終了します
0 4 6 8 11;
入力された集合はdifferenceSetではありません
○ PS C:\Users\81889\Documents\ALL_CODE>
```

図 8 p14 記載の A(16,6,2)-difference Set in $Z/16$ が存在しない出力

3 課題 3

3.1 課題内容

差集合から BIBD を構成し、(1) 及び (2) をチェックせよ

3.2 アルゴリズムの説明

差集合から BIBD を作成するために以下の性質を利用した。

有限加法可換群 G の部分集合である (v, k, λ) -differenceSet D に対して $Dev(D)$ を $Dev(D) = \{D + g; g \in G\}$ とするとき、 $(G, Dev(D))$ は *symmetric* な (v, k, λ) -BIBD である。

これより入力された differenceSet から $Dev(D)$ を作成し、課題 1 の関数を呼び込み BIBD であるか判定する。

3.3 プログラムの内容

プログラム 4 make_BIBD.py

```
1 import check_BIBD
2 import check_differenceSet
3
4 #多次元配列の足し算 D+g をする
5 def add_elements(array, value, order):
6     result = []
7
8     if isinstance(array[0], list):
9         # 2 次元の場合
10        result = [[(element[i] + value[i]) % order[i] for i in range(len(element))]
11                  for element in array]
12    else:
13        # 1 次元の場合
14        result = [element + value for element in array]
15
16    return result
17
18 def make_BIBD(result):
19     #他のファイルからの返戻値を格納
20     v = result[1]
21     differenceSet = result[4]
22     order = result[5]
23     all_Element_G = result[6]
24     #print(differenceSet)
25     #print(all_Element_G)
26     #print(v)
27
28     #演算
29     symmetric_BIBD = []
30     for i in range(v): #D+g を g の個数 (v 個)分ループ
31         added_element = add_elements(differenceSet, all_Element_G[i], order)
32         symmetric_BIBD.append(added_element)
33
34     return symmetric_BIBD
35
36
37 def toDifferenceSetFromBIBD():
38     result = check_differenceSet.check_differenceset()
39     if not result[0]: #そもそも差集合でない
40         print("入力された集合は differenceSet_□ではありません")
```

```

41         return
42
43     #BIBD を計算
44     BIBD = make_BIBD(result)
45     #print(BIBD)
46
47     #課題 1 使用して BIBD か判定
48     isBIBD = check_BIBD.check_BIBD(BIBD)
49     check_BIBD.make_output(isBIBD)
50     return
51
52 toDifferenceSetFromBIBD()

```

3.4 プログラムの説明

1～15 行目

D+g の計算を行う関数 `add.elements()` についての記述である。あるブロック、G のある要素 g、位数を受け取り、位数に注意しながら足し算を行い、計算後のブロックを返す。

18～34 行目

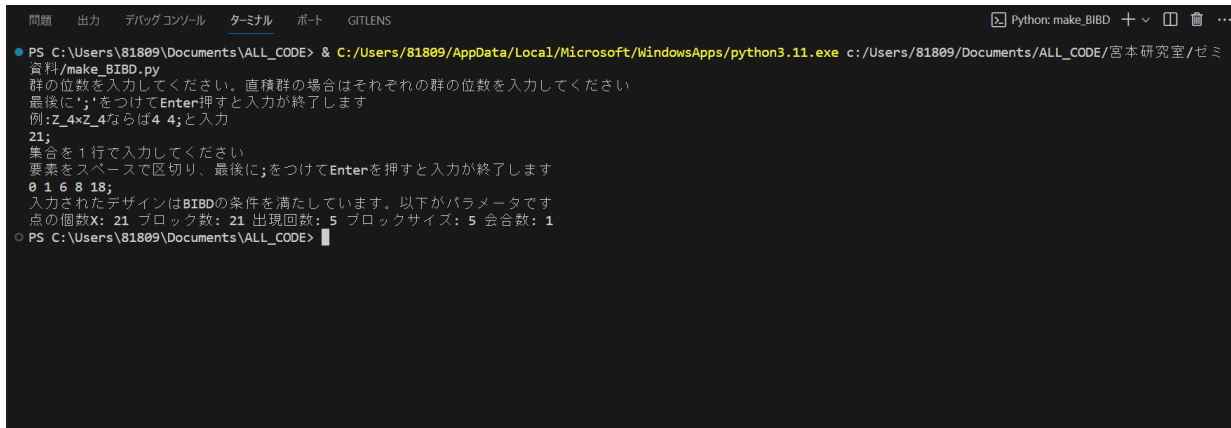
他ファイルからインポートした値を引数にとり、先ほどの `add.elements()` を $|G|$ 回呼び出しながら symmetric な BIBD を計算する関数 `make_BIBD()` についての記述である。

37～52 行目

メイン関数 `toDifferenceSetFromBIBD()` についての記述である。課題 2 のファイルからインポートして入力された集合が `differenceSet` か判定、先ほどの `add.elements()` で BIBD を計算、課題 1 のファイルからインポートして計算したデザインが BIBD か判定している。出力は課題 1 で出力した関数を使用しているため、課題 1 の時の同じである。

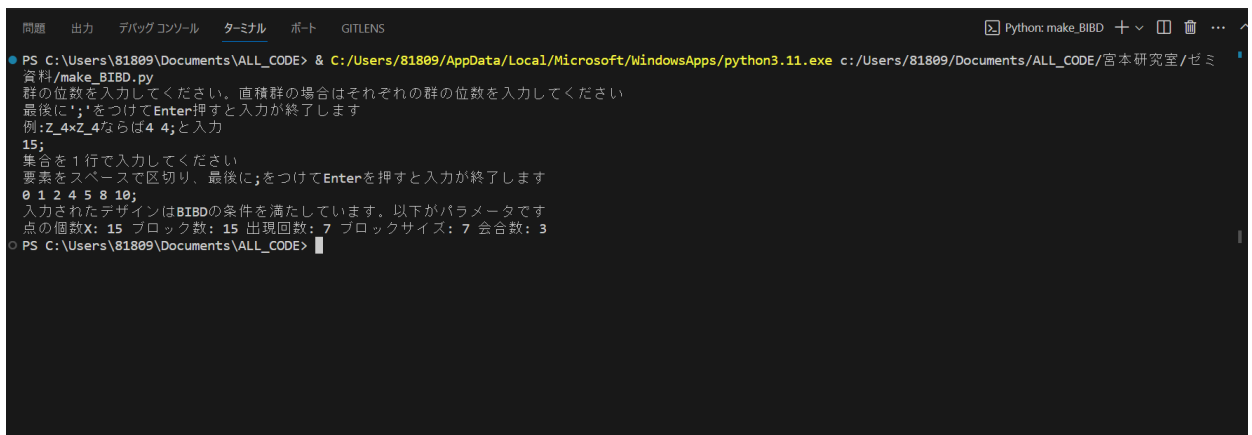
3.5 実行結果

以下が課題 2 と同じデザインを入力したときの実行結果である。



```
Python: make_BIBD + - ...
● PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ
資料/make_BIBD.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
21;
集合を1行で入力してください
要素をスペースで区切り、最後に;をつけてEnterを押すと入力終了します
0 1 6 8 18;
入力されたデザインはBIBDの条件を満たしています。以下がパラメータです
点の個数X: 21 ブロック数: 21 出現回数: 5 ブロックサイズ: 5 会合数: 1
○ PS C:\Users\81809\Documents\ALL_CODE>
```

図 9 p14.Ex2.1 $A(21,5,1)$ -difference Set in $Z/21$



```
Python: make_BIBD + - ...
● PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ
資料/make_BIBD.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
15;
集合を1行で入力してください
要素をスペースで区切り、最後に;をつけてEnterを押すと入力終了します
0 1 2 4 5 8 10;
入力されたデザインはBIBDの条件を満たしています。以下がパラメータです
点の個数X: 15 ブロック数: 15 出現回数: 7 ブロックサイズ: 7 会合数: 3
○ PS C:\Users\81809\Documents\ALL_CODE>
```

図 10 p14.Ex2.2 $A(15,7,3)$ -difference Set in $Z/15$

```
問題 出力 デバッグコンソール ターミナル ポート GITLENS Python: make_BIBD + ~ [] 倉 ... ^
● PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ
資料/make_BIBD.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
4 4;
集合を入力してください
要素をスペースで区切り、行ごとにEnterを押してください。
最後に';'をつけてEnterを押すと入力終了します。
0 0
0 1
1 0
1 2
2 0
2 3;
入力されたデザインはBIBDの条件を満たしています。以下がパラメータです
点の個数X: 16 ブロック数: 16 出現回数: 6 ブロックサイズ: 6 会合数: 2
○ PS C:\Users\81809\Documents\ALL_CODE>
```

図 11 p14.Ex2.3 $A(16,6,2)$ -difference Set in $\mathbb{Z}/4 \times \mathbb{Z}/4$

```
問題 出力 デバッグコンソール ターミナル ポート GITLENS Python: make_BIBD + ~ [] 倉 ... ^
● PS C:\Users\81809\Documents\ALL_CODE> & C:/Users/81809/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/81809/Documents/ALL_CODE/宮本研究室/ゼミ
資料/make_BIBD.py
群の位数を入力してください。直積群の場合はそれぞれの群の位数を入力してください
最後に';'をつけてEnter押すと入力終了します
例:Z_4xZ_4ならば4 4;と入力
16;
集合を1行で入力してください
要素をスペースで区切り、最後に';'をつけてEnterを押すと入力終了します
0 4 6 8 11;
入力された集合はdifferenceSetではありません
○ PS C:\Users\81809\Documents\ALL_CODE>
```

図 12 p14 記載の $A(16,6,2)$ -difference Set in $\mathbb{Z}/16$ が存在しない出力