作品名　　稽古の不成立を未然に回避！―掛かり稽古のモデル化と一般化―

所属　東京学芸大学附属国際中等教育学校　　学年　5 年(高校 2 年)　　氏名　外山みちる

1　研究の要約

合気道の稽古方法の一つである掛かり稽古は、参加者が交代で「投げ(1 人)」と「受け(複数人)」の役割を担いながら進行する。その際、参加人数により「投げ」の役割が全員に回らないことがある（以降「不成立」と呼ぶ）。本研究の目的は、掛かり稽古が不成立となる条件を一般化し、不成立な稽古を避ける方法を提案することである。本研究では、掛かり稽古を数字の並べ替えと捉え、n, m の 2 変数を用いてモデル化した。モデル上で「距離」という概念を定義することで、稽古が成立する必要十分条件「$n-1$ と $m+1$ が互いに素である」を証明することができた。

2　研究の動機と目的

2.1　掛かり稽古とは

掛かり稽古とは合気道の稽古方法の一つである。ここで 2 つの合気道用語を定義する。

投げ：合気道において護身術をする役割。またそれをする人。
受け：合気道において投げを攻撃する役割。またそれをする人。

掛かり稽古では、「投げ」と「受け」を【ルール】に則って順番に交代する。

【ルール】

変数の定義

n (エヌ)：$n \geq 2$を満たす自然数。列全体の人数を表す。
m (エム)：$m < n$を満たす自然数。「受け」の人数を表す。

① 1 列に並んだ n 人の先頭が「投げ」、そこから m 人が「受け」を担当する（$n-m-1$人には役割が与えられない）。

② 「投げ」は「受け」一人ずつと対戦し、対戦した順に「受け」は列の最後尾につく。

③ m 人の「受け」が対戦し終わったとき、「投げ」も列の最後尾につく。

④ その時点で列の先頭にいる人が新たな「投げ」、そこに続く m人が「受け」となる。

⑤ ②～④を繰り返す。ただし、既に「投げ」を経験した人に再度「投げ」の役割が回ってきた時点で終了とする。

2.2　研究の動機

習い始めて 11 年目になる合気道でおこる掛かり稽古の不成立は、私にとって、とても身近な日常生活の問題である。しかし、実際の稽古では掛かり稽古の不成立が起きても、順序を変えるなどで対応するため、不成立の回避という考えは起こらない。数学をテーマに課題研究をすると決めたとき、掛かり稽古における並べ替えはランダム性をもたず、人数にのみに依存するため、数理的なアプローチで解決できるのではないかと気づいた。

2.3　先行研究

掛かり稽古について、もしくは【ルール】と同様の並べ替えに関する先行研究を見つけることはできなかった。

そこで、科学技術振興機構による第 15 回数学キャラバンでの発表資料[1]から以下の点を参考にした。

- トランプのカードを混ぜる手法のうちランダム性を含まないリフルシャッフル（N = 2m 枚のカードを m 枚ずつ上下 2 組に分け、1 枚ずつ互い違いに組み合わせる）について考えると、シャッフルを何回か繰り返すことでカードの並びが元に戻る。
- N 枚のカードにリフルシャフルを L 回繰り返した とき，カードの並びが元に戻るための必要十分条件は

$$2^L \equiv 1 \bmod N-1$$

また、トランプのシャッフルを題材にした群論に関する書籍[2]か

[1] 鈴木武史(岡山大学理学部数学科). 「カードのシャフルと合同式」. 広がる数学 VI―第 15 回 JST 数学キャラバン―. 2016 年

[2] 飯高茂. 『数学のかんどころ 16　群論、これは面白い：トランプで学

ら以下のことが分かった。

- N = 2m 枚のカードを 1 回リフルシャッフルしたときの並び順は初期状態を(m+1)倍してから(2m+1)で割ったあまりである。

表 1. 10 枚のカードを 1 回シャッフルした結果（書籍の内容から自身で作成）

① 初期状態	1	2	3	4	5	6	7	8	9	10
② ①×6	6	12	18	24	30	36	42	48	54	60
③ ②%11	6	1	7	2	8	3	9	4	10	5

2.4 本研究の意義と課題の設定

掛かり稽古は、全員に「投げ」の役割が回ることを目的としているが、(n, m)の組合せによっては一部の人にしか「投げ」が回らないことがある。これは掛かり稽古の目的を果たせておらず、円滑な稽古の進行を妨げる。

成立　：n回の並べ替えによってn人全員が「投げ」を経験できる稽古
不成立：一部の人しか「投げ」を経験できない稽古
　　：n回以下の並べ替えで複数回「投げ」を経験する人がいる

本研究は、掛かり稽古における列の並べ替えの「成立・不成立」を決める条件をn, m の 2 変数を用いて一般化することを課題とする。

一般化された条件に具体的な数字を当てはめることで、稽古をする前に「成立・不成立」を判定できるようになる。これは稽古の効率化に貢献する。

また、掛かり稽古における列の並べ替えもトランプのシャッフルも、数字の並べ替えと捉えることができる。掛かり稽古の成立・不成立条件について一般化することは、シャッフルに関する合同式を用いたアプローチを発展させることになる。

3 方法

3.1 今年度の研究で明らかにする事柄

1. 掛かり稽古の「成立・不成立」を決定する条件
   (ア) 先行研究における並べ替えの題材はリフルシャッフルだったため「成立・不成立」といった考え方がない。よって、まずは組合せ(n, m)を「成立」と「不成立」に分類する。
   (イ) リフルシャッフルではすべてのカードが同様のルールのもとで次のポジションに移動するが、掛かり稽古においては「投げ」と「受け」で移動の仕方が違うため、並べ替えの結果を先行研究のように数列（例：[1,2,3,4,5,6,7,8,9,10]）に対する操作で示すことができない。よって、掛かり稽古に即したモデルを開発する。
2. 「投げ」をする順番はどのようにして得られるか
3. 自身が「投げ」のとき「受け」をするのは誰か

3.2 研究に用いる手法と妥当性

※研究過程で作成したプログラムの具体的なコードは【付録】に記す。
言語は python を使用した。実行環境があればだれでも再現可能である。

<3.1.1.(ア)について>

まず、(n, m)を引数として渡すと並び替えの経過を文字列として出力するプログラム（プログラム1と呼ぶ）を作る。プログラム内では人を people1 = [1,2,3,・・・,n]という配列として捉え、以下のような操作が行われている。

① index が$[m + 1]$~$[n - 1]$の要素を空の配列（people2 = [ ]）に結合する。これは掛かり稽古において役割がなかった人が次の並び順において先頭にくる様子を表している。
② index が[1]~$[m]$の要素を people2 に結合する。これは、対戦が済んだ順に「受け」が列に戻る様子を表している。
③ index が[0]の要素を people2 に結合する。これは、「投げ」が列の最後尾につくことを表している。
④ people1 を people2 で上書きし、people1 を出力する。
⑤ ①~④を繰り返す。ただし、people1 が初期状態([1,2,3,・・・,n])に戻った時点で終了とする。

このようにプログラム 1 は、掛かり稽古の【ルール】に基づいた操作をして結果を出力するため、正しい並べ替えの結果を得ることができる。

次に、任意の自然数 N を指定したら、$2 \leq n < N$を満たす自然数nと、$m < n$を満たす自然数m の組合せ(n, m)で定義される掛かり稽古について、並び替えの成立・不成立を出力する

ぶ群』. 共立出版. 2013 年.

プログラム（プログラム 2 と呼ぶ）を作る。プログラム 1 に、$(N-1)^2/2$通りの (n,m) の組合せを引数として与え、成立・不成立を判定させ、csv データとして出力する。

成立・不成立の定義より、n回未満の並べ替えで 1 が再び配列の先頭にくるならば不成立、n回目に 1 が再び配列の先頭にくるならば成立と判定できるため、この手法も妥当である。

<3.1.1.(イ)について>

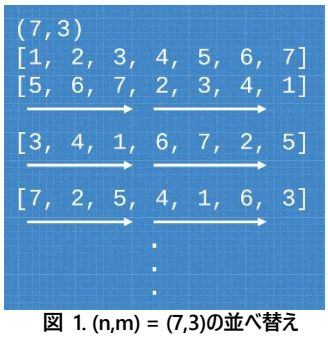

図 1. (n,m) = (7,3)の並べ替え

例えば$(n,m)=(7,3)$において、1 回並べ替えた配列は[5,6,7,2,3,4,1]である。図 1 より、1 を除く部分に関しては並び順が維持されていることが分かる。その次の並べ替えでも[6,7,2,3,4,1]の部分に関しては並び順が維持される。このことより、掛かり稽古における並べ替えは、「投げ」のみの移動と捉えられる。

この性質を生かすために[1,2,3,・・・,n]の配列を円形に配置したモデルを提案する。

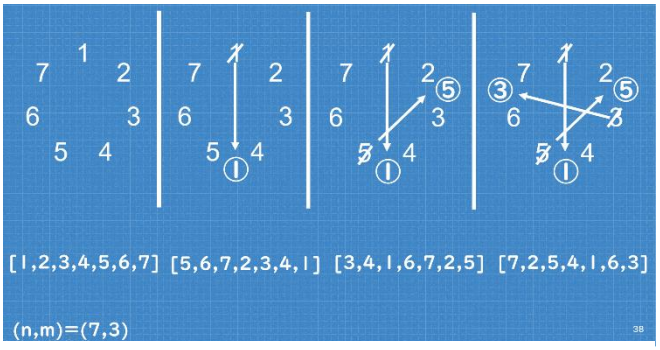

図 2. 円形のモデル(7,3)

図 2 は、左から順に、「初期状態」、「1 回並べ替え」、「2 回並べ替え」、「3 回並べ替え」を表しており、上段の円形のモデルと下段の配列は同じ状況を表している。

- 「1 回並べ替え」では「投げ」をした 1 が数字を m 個(= 3個)またいで、4 と 5 の間に移動する。このとき先頭は、時計回りで①の次にあたる 5 である。
- 「2 回並べ替え」でも同様に、「投げ」をした 5 が[6,7,2]の 3 つの数字をまたいで、2 と 3 の間に移動する。先頭は 3。
- この操作を 1 が再び「投げ」になるまで繰り返す。

ここで「3 回並べ替え」の次は 7 が投げをする。この 7 の両隣を見ると、片方で 1 がいなくなり、もう片方に③が入ってきている。動いていない 2 と 6 を基準に考えると、この状態の 7 は「初期状態」の 1 と同じポジションにあると捉えられる。おのずと、7 は「1 回並べ替え」の 1 と同じ挙動をするため、必然的に⑦は時計回りで①の前に入ることになる。すると次は①が「投げ」になる。よって、「操作を 1 が再び「投げ」になるまで繰り返す」ことは「7(一般的には n)が投げをするまで繰り返す」ことに置き換えられる。

【円形のモデルの操作】

① 1~n までの数字を時計回りに円形に並べる。
② 1 が「投げ」になる。
③ 「投げ」の数字を m個後の数字の次に移動させる。
④ 移動した「投げ」の数字の次の数字が、次の「投げ」になる。
⑤ ③～④を n が「投げ」になるまで繰り返す。

円形のモデルにおいて、先頭から時計回りに数字を読むと、対応する配列と同じ結果が得られることから、このモデルは状況を適切に表していると言える。

4 結果と考察

4.1 モデルとシミュレーションの結果

＜プログラム 1＞

```
>>> loop(7,4)
[1, 2, 3, 4, 5, 6, 7]
[6, 7, 2, 3, 4, 5, 1]
[5, 1, 7, 2, 3, 4, 6]
[4, 6, 1, 7, 2, 3, 5]
[3, 5, 6, 1, 7, 2, 4]
[2, 4, 5, 6, 1, 7, 3]
[7, 3, 4, 5, 6, 1, 2]
```

図 3. プログラム 1 の出力結果(7,4)

```
>>> loop(7,3)
[1, 2, 3, 4, 5, 6, 7]
[5, 6, 7, 2, 3, 4, 1]
[3, 4, 1, 6, 7, 2, 5]
[7, 2, 5, 4, 1, 6, 3]
[1, 6, 3, 2, 5, 4, 7]
[5, 4, 7, 6, 3, 2, 1]
[3, 2, 1, 4, 7, 6, 5]
[7, 6, 5, 2, 1, 4, 3]
[1, 4, 3, 6, 5, 2, 7]
[5, 2, 7, 4, 3, 6, 1]
[3, 6, 1, 2, 7, 4, 5]
[7, 4, 5, 6, 1, 2, 3]
[1, 2, 3, 4, 5, 6, 7]
12
```

図 4. プログラム 1 の出力結果(7,3)

＜プログラム 2＞

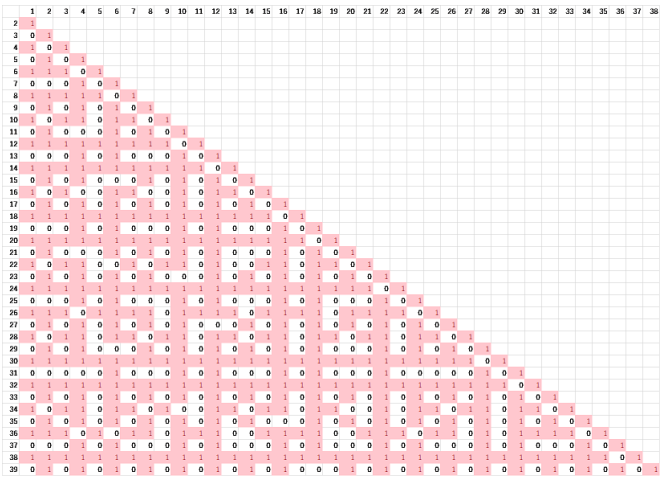

図 5. プログラム 2 の出力結果(N=40)

行番号が n 、列番号が m を表す。セルの値が 1 のとき成立、0 のとき不成立を表す。

＜円形のモデル＞

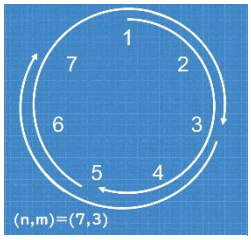

図 6.「投げ」の移動

数字の移動ではなく、「投げ」という役割が、1→5、5→3、3→7 という風に移動したことに注目すると、$(n,m)=(7,3)$のとき、「投げ」の役割は円形のモデルを 2 周して 7 にたどり着いたことが分かる。

ここで新たな変数を定義する。

k (ケイ)：$k \geq 2$を満たす自然数。n までに投げを経験する人数を表す。
l (エル)：投げになった n は、何周目の n なのかを表す自然数。
例）$(n,m)=(7,3)$のとき、$(k,l)=(3,2)$

また、円形のモデル上に「距離」という概念を定義する。

距離：
円形のモデル上での「投げ」の役割の移動に注目したとき、1 から 5 まで動いたら「距離 4 移動した」と考える単位。隣の数字までの間の幅が「距離 1」である。

そして、「距離」の概念に基づいて、「移動距離」という言葉を定義する。

移動距離：
「投げ」の役割が n にたどり着くまでに移動した距離のこと。
図 6 における白い矢印の総延長とも言える。

「移動距離」について 2 つの異なる方針で大きさを求めていく。

① 1 から 2 周目の 1 までの移動距離が 7 である点に着目

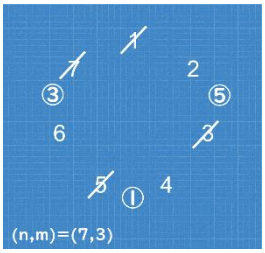

図 7. 矢印無しの円形のモデル

図 7 は、$(n,m)=(7,3)$についての円形のモデルで 7 が投げをした時点の様子である（矢印は削除した）。

1 以外のポジションでは、／と◯が隣り合っていることが分かる。

l周して n にたどり着いたときその「移動距離」はおおよそ $nl-1$ である。しかし、「投げ」の役割が 十をまたぐ度に、1 少ない距離で次の数字までたどり着ける。十をまたぐ回数は $l-1$ 回である。

よって、「移動距離」は

$nl-1-(l-1)=l(n-1)$ である。 ･･･（A）

②「投げ」が 1 回交代する当たりの距離から計算

$(n,m)=(7,3)$において、1→5 までの距離は 4 である。一般的に考えると、距離は「投げ」1 人当たり $m+1$ である。n までに「投げ」を経験する人数は k人であることから、

「移動距離」は $k(m+1)$ である。･･･（B）

※以降登場する L.C.M.(,)はかっこ内の 2 数の最小公倍数を、G.C.D.(,)はかっこ内の 2 数の最大公約数をあらわす。

（A），（B）より、

$$l(n-1)=k(m+1)$$
$$=L.C.M.(n-1,m+1) \quad \cdots(C)$$

等式 C において、n,m は与えられた自然数であるため独立変数だが、l,k は n,m に従う従属変数であり、式の値が $L.C.M.(n-1,m+1)$ になるような値をとる。最小公倍数である理由は n が 1 度目に投をするまでの「移動距離」を求めているからである。$t \times L.C.M.(n-1,m+1)$移動するとき、n は t 回目の投げを経験する（ここで t は任意の自然数）。

また、k の定義より、成立 ⇔ $k=n-1$である。

(C)より、$k=n-1$ ⇔ $G.C.D.(n-1,m+1)=1$

よって、成立 ⇔ $G.C.D.(n-1,m+1)=1$

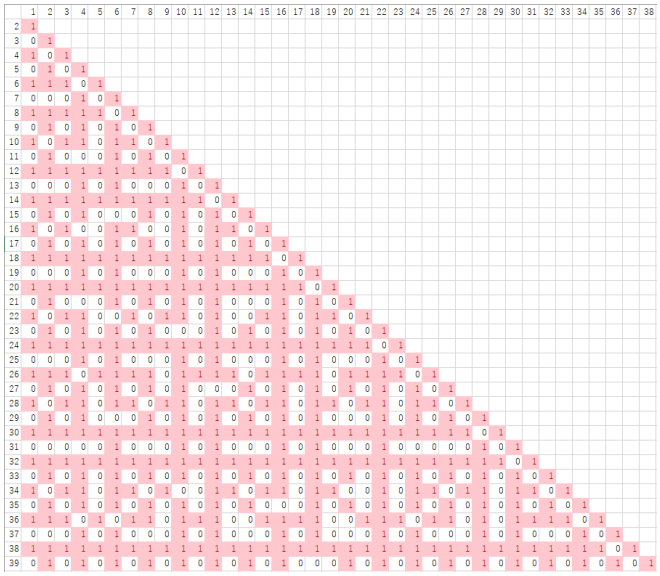

図 8. n-1 と m+1 が互いに素

図 8 は、n-1 と m+1 が互いに素なら 1、互いに素でないなら 0 の値を持つ表データの画像である。行番号は n、列番号が m を表す。図 5 と図 8 の対応するセル同士を引き算した結果は、どのセルについても 0 であった。これは有限な範囲において、【ルール】に基づく「成立・不成立」と式に基づく「成立・不成立」が一致していることを示す。ここからも円形のモデルより導かれた成立条件の妥当性がうかがえる。

4.2　3.1.2 で述べた「投げ」をする順番に関して

4.2.1　「投げ」をする順番を調べる意義

```
>>> loop(7,4)
[1, 2, 3, 4, 5, 6, 7]
[6, 7, 2, 3, 4, 5, 1]
[5, 1, 7, 2, 3, 4, 6]
[4, 6, 1, 7, 2, 3, 5]
[3, 5, 6, 1, 7, 2, 4]
[2, 4, 5, 6, 1, 7, 3]
[7, 3, 4, 5, 6, 1, 2]
```

図 3. プログラム 1 の出力結果 (7,4)

図 3 は$(n, m) = (7,4)$の並べ替えについて、プログラム 1 を用いて出力した結果である。

$\mathrm{G.C.D.}(6,5) = 1$ であることからも図 3 からも(7,4)が成立する組合せであることが分かる。

このとき、投げの順番は[1,6,5,4,3,2,7]である。2 列目を縦に読んでも[1,6,5,4,3,2,7]があらわれる。3 列目も同様である。

1 が「投げ」のときの「受け」は [2,3,4,5] である。配列 [1,6,5,4,3,2,7] において、1 にとっての[2,3,4,5]と 6 にとっての[7,2,3,4]は、相対的な位置関係が同じである。

このことから配列[1,6,5,4,3,2,7]を得る方法について考えることで、投げの順番(3.1.2)および自身が「投げ」のとき誰が「受けをするのか」(3.1.3)を明らかにできる。

4.2.2　「投げ」をする順番を調べる方法

定義④に基づいて「移動距離」を求める方法と同様に、i番目に投げをする人までの移動距離は$(m+1) \times (i-1) + 1$である。i番目に投げをする人が N $(1 \leq N \leq n$を満たす任意の自然数)番の場合、Nは移動距離を $n-1$ で割ったあまりである。これを

$$N = \{(m+1) \times (i-1) + 1\} \% (n-1)$$
$$= \{(m+1) \times i - m\} \% (n-1) \quad \cdots \text{(D)}$$

と表す。

配列[1,2,3,・・・,n]全体に「$\times (m+1)$」、「$-m$」,「$\%(n-1)$」の操作をすることで、式(D)の i に$1 \sim n$ まで代入したときのそれぞれのN が得られる。

```
(7,4)
[1,2,3,4,5,6,7]
↓ *(m+1)
[5,10,15,20,25,30,35]
↓ -(m)
[1,6,11,16,21,26,31]
↓ %(n-1)
[1,0,5,4,3,2,1]
[1,6,5,4,3,2,7]
```

図 9. 「投げ」の並び順を得る方法

※ $n-1$ で割ったあまりに $n-1$ と n は登場しない。よって配列の 0 は$n-1$ に、2 つ目の 1 は n に直す必要がある。

4.2.3　「投げ」をする順番と成立条件の証明

図 9 に示すような処理をするプログラム(プログラム 3 と呼ぶ)を作った。

```
>>> head(7,4)
[1, 6, 5, 4, 3, 2, 7]
>>> head(7,3)
[1, 5, 3, 1, 5, 3, 7]
```

図 10. プログラム 3 の出力結果

(7,4)は成立する組合せ、(7,3)は不成立の組合せである。

図 10 はこの 2 つについて「投げ」の順番を出力した結果だ。

成立する組合せにおいては、$1 \sim n$ までの数字が 1 つずつ並ぶ。対して、不成立の組合せでは、一部の数字が複数個並ぶ。

※この計算方法では、不成立の組合せについて正しく「投げ」の順番を出力できているとは言えないが、[1,5,3,7]の繰り返しになることは確かである。

$(m+1)$の倍数を$(n-1)$で割ったあまりは、$(m+1)$と$(n-1)$の最大公約数の倍数になる。

「投げ」の順番の計算過程において、$n-1$ で割ったあまりが 1 刻みになるためには、$m+1$ と $n-1$ の最大公約数が 1 である必要がある。

このことからも、$n-1$ と $m+1$ が互いに素であることが成立の必要条件だと分かる。

5　結論と今後の課題及び感想

5.1　結論

本研究では、掛かり稽古の「成立・不成立」と変数(n, m)の関係性を明らかにすることを目的として、掛かり稽古の【ルール】に基づいて、組合せ(n, m)を「成立・不成立」に分類した(図 5)。これは仮説や予測に基づくものではなく、実際の状況を再現しているので適当な処理であった。

また、掛かり稽古の性質に基づいてモデル化を行った。そのモデルはプログラム 1 を用いて出力した並べ替えと同様の状況を正しく表せるため適切であった。

さらに、円形のモデル上で「距離」という概念を定義することで、以下のことが分かった。

- $n-1$ と $m+1$ が互いに素であることが掛かり稽古を成立させる必要十分条件である。
- 配列[1,2,3,・・・,n]全体に「$\times(m+1)$」、「$-m$」,「$\%(n-1)$」の操作をすることで、「投げ」をする順番を得られる。
  また、これを用いれば自身が「投げ」のときに「受け」をする人を特定することができる。

5.2 今後の課題

4.2 において、「投げ」の順番の計算方法から、$n-1$ と $m+1$ が互いに素であることが、掛かり稽古が成立する必要条件であることを示せた。 今後、このアプローチから、必要十分条件であることを示していきたい。

今年度の研究では、変数 n, m は自然数としたが、今後、整数の範囲に拡張し、同様の性質が成り立つのか確かめたい。また、$n > m$ の制限についても、$n \leq m$に拡張していきたい。

5.3 感想

研究を始める前は、 n, mの条件式が稽古での実用には堪えないほど七面倒な計算を必要とするのではないかと心配していた。本研究を経て、「互いに素であるか、否か」という暗算で瞬時に「成立・不成立」が判定できる条件であると分かり、非常に喜ばしい。

「数学なんてなんの役に立つのかわからない」とこぼすクラスメイトもいる。しかし、数学を学ぶことで今まで問題とも思わずに見過ごしていたことに気づくことができた。この研究を通じて数学が日常生活の問題解決に役立つことを実感できたのは、数学を学ぶ大きなモチベーションに繋がっている。

6 参考文献

[1] 鈴木武史(岡山大学理学部数学科). 「カードのシャフルと合同式」. 広がる数学 VI―第 15 回 JST 数学キャラバン―. 2016 年

[2] 飯高茂. 『数学のかんどころ 16 群論、これは面白い：トランプで学ぶ群』. 共立出版. 2013 年.

【付録】

プログラム 1

```
import copy

def oneTurn(n,m):
    global people1, people2
    people2 = []

    for i in range(m+1,len(people1)):
        people2.append(people1[i])
    for i in range(1,m+1):
        people2.append(people1[i])
    people2.append(people1[0])

    people1 = copy.copy(people2)

    return people1

def loop(n,m):
    global people1, people2
    people1 = []
    for i in range(n):
        people1.append(i+1)

    check = []
    for i in range(n):
        check.append(i+1)

    print(people1)
    oneTurn(n,m)
    print(people1)
    count = 1

    while people1!=check:
        oneTurn(n,m)
        count = count+1
        print(people1)
    print(count)

#実行の際は>>>loop(7,3)のように適当な引数を渡す
```

プログラム 2

```
import copy
import pandas as pd

def oneTurn(n,m):
    global people1, people2
    people2 = []

    for i in range(m+1,len(people1)):
        people2.append(people1[i])
    for i in range(1,m+1):
        people2.append(people1[i])
    people2.append(people1[0])

    people1 = copy.copy(people2)

    return people1

def kakari(n,m):
    global people1, people2
    people1 = []
    for i in range(n):
        people1.append(i+1)

    check = []
    for i in range(n):
        check.append(i+1)

    oneTurn(n,m)
    count = 1

    while people1!=check:
        oneTurn(n,m)
        count = count+1

    if count == n:
        return 1
    else:
        return 0

def all(n):
    oxList = []
    indexList = []
    columnsList = []

    for i in range(2,n):
        oxList.append([])
        indexList.append(str(i))
        columnsList.append(str(i-1))
        j = 1
        while j<i:
            answer = kakari(i,j)
            j = j + 1
            oxList[-1].append(answer)

    df =
pd.DataFrame(data=oxList,index=indexList,columns=columnsLis
t)
    df = df.fillna("")
    df.to_csv("kakarigeiko-{}.csv".format(n))

    print(df)

#実行の際は>>>all(40)のように適当な引数を渡す
```

プログラム 3

```
def head(n,m):
    people1 = []
    people2 = []
    for i in range(n):
        people1.append(i+1)
    for i in people1:
        if i==n:
            num = n
        else:
            num = (i*(m+1)-m)%(n-1)
            if num == 0:
                num = num+n-1
        people2.append(num)
    print(people2)

#実行の際は>>>head(7,3)のように適当な引数を渡す
```