A-010

行列を用いたラテン超方格と直交配列の探索
Search of Latin hypercubes and orthogonal arrays using matrices

野澤 友希† 足立 智子†
Yuuki Nozawa Tomoko Adachi

1. はじめに

位数nのラテン方陣とは、大きさ$n \times n$の正方形にどの行・どの列でもすべて異なるようにn個のシンボルを配置したものである。ラテン方陣については文献[1,2]が詳しい。ラテン方陣を高次元にしたものが、ラテン超方格である。ラテン超方格についての先行研究は文献[3,4]がある。ラテン超方格は直交配列と関係があり、文献[5,6]の先行研究がある。

Ethier and Mullen[4]は、$GF(q)$上のd変数の一次多項式t本から、d次元のラテン超方格を構成する方法を提案した。

そこで本研究では、Ethier and Mullen の手法をもとに、行列の階数を計算し、行数を増やしていくことで、ラテン超方格と直交配列を探索する。

2. 先行研究の結果

2.1 統計学におけるラテン超方格

ラテン超方格や直交配列は、統計学における実験計画法やサンプリング手法で用いられている。

実験計画法は、データ分析の技術・手法の一つである。定められた計画に基づき測定されたデータを解析し、因子の効果や交互作用を明らかにする。

水準nの因子3個の三元配置において、実験回数をn^3からn^2へ減らすために、位数nのラテン方陣が用いられる。因子が多い多元配置においては、ラテン超方格が用いられる。水準nの因子が$n+1$個まで調べられる配置が得られたとき、これを互いに直交する完全配置と呼ぶ。これを直交配列の形に修正して、実験計画法でよく利用する。

統計実験の三原則の一つにランダム化がある。文献[7,8]によると、「誰かが前もって何が起こるかを決めることなく、もしくはどんな規則的なパターンもなく、なされたり選ばれたりなどすること」をランダムネス(randomness)という。ランダムネスの活用に無作為標本(random sample)や無作為割付(random assignment)があり、ラテン超方格や直交配列が用いられる。

2.2 互いに直交するラテン超方格

Lu and Adachi[3]や Ethier and Mullen[4]は、有限体$GF(q)$の性質を用いて、互いに直交するラテン超方格の構成法を提案した。二つのラテン超方格を重ね合わせたときに、シンボルの順序対がすべて異なる場合、直交するという。任意の二つが直交するような、位数qのd次元ラテン超方格の複数個の組を、$(d, d-1)-MOC(q)$と記す。

Ethier and Mullen[4]の手法では、$GF(q)$上のd変数の一次多項式t本から、d次元のラテン超方格を構成する。

補題1 ([4, Theorem4.4]) tをd以上の整数とする。
$f_i(x_1, x_2, \dots, x_d) = a_{i,0}x_1 + a_{i,1}x_2 + \cdots + a_{i,d-1}x_d \ (1 \le i \le t)$
をF_q上の一次多項式とする。このとき、$f_1, f_2, \dots, f_t$によって生成されるd次元超立方体(d-cube)は、行列$M = (a_{i,j})_{t \times d}$のすべての$d$行が線型独立であるときかつその場合に限り、$(d, d-1)-MOC(q)$を形成する。

3. 提案手法1

3.1 提案の概要

第2.2節の補題1より、行列Mがわかれば、ラテン超方格が構成できる。ラテン超方格がわかれば、文献[5]より、直交配列が構成できる。そこで本研究では、行列Mを探索する。

サイズ$t \times d$の行列Aから、任意のd個の行を選んで抜き出し、$d \times d$行列Bを構成する。この行列Bの階数がdであれば、補題1の条件を満たすため、Aが求める行列Mとなる。上のような条件を満たす行列Aを作成するために Python でプログラムを作成する。

3.2 アルゴリズム

$Step1$: サイズ$d \times d$の初期行列Cを用意する。ここで、行列Cの階数はdであり、行列Cの成分は$GF(q)$の元とする。ただし、$q \ge d$とする。

$Step2$: 行列Cに新しく1行加えて、$(d+1) \times d$行列Aを作成する。そのために、加える行の候補を考える。q種類のシンボルがd列分なので、考えられる行の組み合わせはq^d通りである。これを1行ずつ順番に加えていく。

$Step3$: 行列Aからd個の行を選んで抜き出す。ここで、考えられる行の抜き出し方の組み合わせは${}_{(d-1)}C_d$通りである。この抜き出した行列から作った${}_{(d-1)}C_d$個の行列Bすべてについて階数を調べる。

$Step4$: すべてのBの階数がdならば、Aに新しくもう1行加えて、$Step2$から繰り返す。階数がdでないBがあれば、その新しい行を不採用とし、別の候補と入れ替えて、$Step3$から行う。すべての行の候補が不適切なら、一つ前の行に戻り、その行を別の候補と入れ替えて、$Step3$から繰り返す。

$Step1$から$Step4$により、行列Cに対して、考えられるすべての行の追加パターンを試すことになる。

3.3 初期行列

初期行列Cとして、下記のC_1と単位行列Iを用意する。

$$C_1 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

†静岡理工科大学 Shizuoka Institute of Science and Technology

4. 提案手法1による結果および考察

4.1 結果

提案手法1によって探索した結果、表1を得た。

表1　提案手法1による行列Aの探索結果

実験番号	1-1	1-2	1-3[注1]	1-4
初期行列	C_1	I_3	C_1	I_5
初期行列サイズ	3×3	3×3	3×3	5×5
行列成分$GF(q)$	$GF(3)$	$GF(3)$	$GF(5)$	$GF(5)$
Aの最大行数(t)	6	6	12以上	10以上
Aの個数	約 200	約 30	中断	中断
実行時間	5.96秒	1.12秒	6時間	4.5時間
実行回数	193	33	中断	中断
実行結果の行数	60,031	9,865	91,303,519	12,434,697
実行環境(CPU)	Core™ i7-8565U	Core™ i7-8565U	Core™ i5-11400F	Core™ i7-8565U

[注1] 実験1-3のみ実行環境が異なる。

表1の実行回数は、提案手法1の$Step2$における新しく1行追加する処理を行った回数である。

$GF(3)$のもとで、サイズ3×3の初期行列C_1, I_3から得られた行列Aのうち、最も行数が高かったのは、どちらも3行増加した6行であった。実行回数は1-1が193回、1-2が33回となった。初期行列I_3の方が実行回数は少ないが、得られた行列Aの個数も少ない。

$GF(5)$のもとで、初期行列C_1について、提案手法1で探索したところ、6時間かけても終了しなかったため、途中で探索を打ち切った。ここで発見できた行列のうち、最も行数が高かったのは、9行増加した12行であった。

また、初期行列をサイズ5×5の単位行列I_5に変えて、探索をしたところ、4.5時間かけても終了しなかったため、途中で打ち切った。ここで発見できた行列のうち、最も行数が高かったのは、5行増加した10行であった。

4.2 考察

実行回数は、新しく行を追加するときに増加するため、この値が大きいほど、行列Mの条件を満たす行列Aを多く発見できたと考えられる。

また、全てのパターンを探索する場合、現状のプログラムでは時間がかかってしまうことがわかった。よって、すべてのパターンを調べるのではなく、条件を満たすものを一つ見つけることに目的を変更し、提案手法2を考案する。

5. 提案手法2

提案手法1の$Step2$と$Step4$を以下のように変更する。

$Step2'$: 行列Cに新しく1行加えて、$(d+1)\times d$行列Aを作成する。q種類のシンボルがd列分なので、考えられる行の組み合わせはg^q通りである。この候補の中からランダムに1行を選んで追加する。

$Step4'$: すべてのBの階数がdなら、Aに新しくもう1行加えて、$Step2$から繰り返す。階数がdでないBがあれば、その新しい行を不採用とし、別の候補と入れ替えて、$Step3$から行う。すべての行の候補が不適切なら、探索を終了する。

6. 提案手法2による結果

提案手法1で中断した実験1-3や1-4と同じ初期行列と$GF(q)$について、提案手法2で10回探索を行った。そして、発見できた行列のうち最も行数が高いもの(t_{max})と低いもの(t_{min})を調べ、表2にまとめた。

表2　提案手法2による行列Aの探索結果

実験番号	2-1	2-2	2-3
初期行列	C_1	I_5	I_7
初期行列サイズ	3×3	5×5	7×7
行列成分$GF(q)$	$GF(5)$	$GF(5)$	$GF(7)$
Aの最大行数(t_{max})	13	10	12以上
Aの最小行数(t_{min})	10	8	中断
平均実行時間	1.85秒	60.23秒	1時間

$GF(5)$のもとで、初期行列C_1について、提案手法2で探索して得られた行列Aのうち、最も行数が高かったのは13行であった。平均実行時間は1.85秒であった。提案手法1で見つかったもの(12行)を超える13行の行列Aを発見できた。

$GF(5)$のもとで、初期行列I_5について、提案手法2で探索して得られた行列Aのうち、最も行数が高かったのは10行であった。平均実行時間は60.23秒であった。提案手法1で見つかったものと同じ10行の行列Aを発見できた。

さらに、$GF(7)$のもとで、初期行列I_7について、提案手法2で探索したところ、1時間かけても終了しなかったため、途中で探索を打ち切った。ここで発見できた行列のうち、最も行数が高かったのは、5行増加した12行であった。

7. おわりに

アルゴリズムの改良として、次が考えられる。

行列Aにおいて、行列Cにあたる最初の3行の部分と、追加した行との組み合わせでできる行列Bの階数がdでなければ、その行の候補を使用したAは、絶対に不適切なものになる。よって、最初にCと追加する行の候補とで階数を調べ、追加には適さない行を以降の探索候補から省く。

また、現在のプログラムは階数の確認を、行列Aからできるすべての行列Bに対して行っているが、これを新たに追加した行を含む行列Bとのみ行うようにすることで、確認の回数を減らす。

参考文献

[1] A.D. Keedwell and J. D´enes, Latin Squares and Their Applications, 2nd ed., Elsevier, (2015).

[2] C.F. Laywine and G.L. Mullen, Discrete Mathematics Using Latin Squares, John Wiley & Sons, (1998).

[3] X.-N. Lu and T. Adachi, “On Dimensionally Orthogonal Diagonal Hypercubes”, IEICE TRANS. FUNDAMENTALS, Vol.E103-A, No.10, pp.1211-1217 (Oct., 2020).

[4] J.T. Ethier and G.L. Mullen, “Strong forms of orthogonality for sets of hypercubes”, Discrete Math., vol.312, no.12, pp.2050–2061, (2012).

[5] B. Tang, “Orthganal Array-Based Latin Hypercubes”, Journal of the American Statistical Association, vol.88, Issue 424, pp.1392-1397 (Dec., 1993).

[6] A.S. Hedayat, N.J.A. Sloane, John Stufken, Orthogonal arrays : theory and applications, Springer Science+Business Media New York, (1999).

[7] 濵田悦生, 狩野裕, データサイエンスの基礎, 講談社, (2019)

[8] A.S. Hornby, Oxford Aduanced Learner’s Dictionary of Current English, 9th ed., Oxford University Press, (2015).