The 14th International Conference on Dependable Systems, Services and Technologies, DESSERT'2024
11-13 October, 2024, Athens, Greece

A Vector Division Algorithm for Constructing Orthogonal Latin Hypercube Designs

Maria Boufi
Department of Mathematics
National Technical University of Athens
Athens, Greece
marboufi@gmail.com

Christos Koukouvinos
Department of Mathematics
National Technical University of Athens
Athens, Greece
ckoukouv@math.ntua.gr

Marilena Mitrouli
Department of Mathematics
National and Kapodistrian University of Athens
Athens, Greece
mmitroul@math.uoa.gr

***Abstract*—Latin hypercube designs are commonly used in many applications such as, computer and physical experiments, data selection by sampling techniques and recently in machine learning. The aim of this work is to develop an algorithm concerning their construction. The proposed algorithm is based on a vector division procedure and employs only permutations and negations of vectors. Therefore, it produces the orthogonal Latin hypercube designs cost effectively without performing any floating point computations. The algorithm is compared with other existing methods and it is proved that the developed method produces Latin hypercube designs that are isomorphic with the ones produced by other methods.**

***Keywords*—Latin hypercube design, vector division, permutations, negations, effective algorithm**

I. Introduction

Combinatorial designs are widely used in various fields of scientific research. Depending on the case, designs with special properties are required. A commonly used category of such designs are Latin hypercube designs. Introduced by McKay, Beckman and Conover [6], these designs are extensively utilized for planning computer experiments, due to their space-filling properties. A lot of research has been conducted concerning methods for their construction.

A Latin hypercube design (LHD) is an $n \times m$ array, each column of which is a permutation of the elements of the set $S_n^{(1)} = \{1, 2, ..., n\}$. We say that a LHD is in its centered form, if each column is a permutation of the elements of the set $S_n^{(2)} = \{v_1, v_2, ..., v_n\}$, where $v_i = i - (n+1)/2$, $1 \leq i \leq n$. In computer experiments, the rows of a design represent the runs, while the columns represent the factors.

A LHD will said to be symmetric if for any row d, $-d$ is also one of the rows in the design.

In statistical research, it is often essential to examine the correlation between factors of a design. For this purpose, the notion of the correlation matrix is introduced. For any $n \times m$ Latin hypercube design L, we define

$$R = \frac{12}{n(n^2-1)} L^T L$$

to be the correlation matrix of L. An $n \times m$ LHD L is called orthogonal (abbreviated OLHD) if and only if $R = I_m$, where I_m is the identity matrix of order m. If a LHD is symmetric and orthogonal, we will use the abbreviation SOLHD.

If a LHD L is orthogonal, then the interchanging of columns and the multiplication of columns by -1 does not affect the orthogonality. Same holds for the symmetry.

Orthogonal Latin hypercube designs are useful in many applications. Hence, a lot of research is focused on their construction. Some examples are [10], where OLHDs of orders $2^m \times (2m-2)$ and $(2^m+1) \times (2m-2)$, for $m \geq 2$, are constructed, [1], where the previous construction is extended in order to provide more columns and [4], where an algorithm for creating OLHDs of n runs, $n \leq 20$, is presented. Some interesting constructions can be found in [5] and [8]. For more recent constructions see [2], [3] and [9].

The structure of the paper is as follows: In section 2, we present an algorithm for constructing symmetric orthogonal Latin hypercube designs of order $2^t \times 2^{t-1}$ and we illustrate our method with an example. In section 3, we compare our algorithm with a method introduced by Sun et al. [7] and we show that the two approaches give isomorphic results. In section 4 are drawn conclusions.

Throughout the paper we will work with LHDs in their centered form.

II. The algorithm

Next, we present an algorithm for constructing SOLHDs of order $2^t \times 2^{t-1}$, $\forall t \in \mathbb{N}$. The core element of the algorithm is a Vector Division (VD) procedure which divides a given vector into parts of a specific length. Then it reverses the order of the elements of each part and negates the second half of each reversed part.

Beginning with the first column

$$c_1 = \tfrac{1}{2}\left[1 \quad 3 \quad \dots \quad 2^t-3 \quad 2^t-1 \quad -1 \quad -3 \quad \dots \quad -(2^t-3) \quad -(2^t-1)\right],$$

we apply the VD procedure so that c_1 is divided into parts of length $2, 2^2, 2^3, \dots, 2^{t-1}$ and each part is reversed and negated as previously described. Subsequently, we apply the VD procedure to the new columns that have been created and we continue in a similar way, i.e. from each new column that is created we construct more new columns. The way in which we divide every column in order to create new ones depends on how the column itself was constructed. More specifically,

979-8-3315-1093-0/24/$31.00 © 2024 IEEE

if a column c_i was made by dividing a previous column into parts of length 2^s, then c_i will be divided into parts of lengths $2^{s+1}, 2^{s+2}, ..., 2^{t-1}$ and will provide $t-1-s$ new columns. If a column was made by dividing a previous column into parts of length 2^{t-1}, then it will not be used for constructing new columns. The algorithm ends when there are no new columns that can be created.

The way in which the elements of each new column are partitioned and rearranged guarantees that every column constructed is orthogonal to all the previous ones. Also, the number of columns that will be created is 2^{t-1}.

Since we aim to construct a symmetric OLHD L, we only need to construct the first half of each column of the design. The second half can be made from the first half by negating it. For ease of reference, we will use c_i to denote the first half of the i-th column of L, i.e. the whole column will be $[c_i^T \ -c_i^T]^T$. Thus, our algorithm constructs the upper half $C = [c_1 \ c_2 \dots c_{t-1}]$ of the SOLHD L and then obtains L as $[C^T \ -C^T]^T$. Also, we can omit the multiplication by $\frac{1}{2}$ in the steps of the method, since it does not alter the orthogonality or the symmetry, and we can multiply all the columns that we will have created by $\frac{1}{2}$ in the end.

A. Vector Division Algorithm

Throughout the algorithm we will use the following notation:

- $a : s : b$ denotes the vector $[a, a+s, a+2s, ..., a+ns]$, with $a + ns \leq b$, i.e. the vector that begins with a and moves with step s until it surpasses b. If $s = 1$, we denote $a : b$.
- $v(i)$ denotes the i-th entry of vector v. If i is a vector, $v(i)$ denotes the vector comprised by the corresponding entries of v.
- $A(i, j)$ denotes the (i, j) entry of matrix $A \in \mathbb{R}^{n \times m}$. If i and/or j are vectors, then $A(i, j)$ denotes the submatrix of A comprised by the corresponding entries of A as defined by i and j. $A(:, j)$ will be shorthand for $A(1 : n, j)$ and $A(i, :)$ will be shorthand for $A(i, 1 : m)$.

We will also consider the following functions readily available:

- $zeros(n, m)$ returns an $n \times m$ matrix with all its entries equal to zero.
- $length(v)$ returns the length of vector v.

First, we create the function VD that reads a vector c and a value k and applies the VD procedure on c by dividing it into parts of length 2^k, reversing the order of the elements of each part and negating the second half of each part.

function c=VD(c,k)
$m \leftarrow length(c)/(2^k)$
$\%m$ is the number of the parts in which c will be divided
for $i = 1 : m$
  %Reversion of the elements of part i
  $c(i \cdot 2^k - 2^k + 1 : i \cdot 2^k) \leftarrow c(i \cdot 2^k : -1 : i \cdot 2^k - 2^k + 1)$
  %Negation of the second half of part i
  $c(i \cdot 2^k - 2^k + 1 + 2^{k-1} : i \cdot 2^k) \leftarrow$
    $-c(i \cdot 2^k - 2^k + 1 + 2^{k-1} : i \cdot 2^k)$
end

In the following example, we demonstrate the VD procedure, as executed by function $VD(c, k)$, for $c = \begin{bmatrix}1 & 3 & 5 & 7 & 9 & 11 & 13 & 15\end{bmatrix}^T$ and $k = 1$.

Example 1. Function $VD(c, k)$, for $c = \begin{bmatrix}1 & 3 & 5 & 7 & 9 & 11 & 13 & 15\end{bmatrix}^T$ and $k = 1$
$m = 8/2^1 = 4$, i.e. c will be divided into 4 parts of length 2:

$$[\ 1 \quad 3 \mid 5 \quad 7 \mid 9 \quad 11 \mid 13 \quad 15\ ]^T$$

Iteration $i = 1, \dots, 4$ begins:
$i = 1$
$c(i \cdot 2^k - 2^k + 1 : i \cdot 2^k) \equiv c(1 : 2) \leftarrow$
  $c(i \cdot 2^k : -1 : i \cdot 2^k - 2^k + 1) \equiv c(2 : -1 : 1)$

$$c = [\ 3 \quad 1 \mid 5 \quad 7 \mid 9 \quad 11 \mid 13 \quad 15\ ]^T$$

$c(i \cdot 2^k - 2^k + 1 + 2^{k-1} : i \cdot 2^k) \equiv c(2) \leftarrow$
  $-c(i \cdot 2^k - 2^k + 1 + 2^{k-1} : i \cdot 2^k) \equiv -c(2)$

$$c = [\ 3 \quad -1 \mid 5 \quad 7 \mid 9 \quad 11 \mid 13 \quad 15\ ]^T$$

$i = 2$
$c(3 : 4) \leftarrow c(4 : -1 : 3)$

$$c = [\ 3 \quad -1 \mid 7 \quad 5 \mid 9 \quad 11 \mid 13 \quad 15\ ]^T$$

$c(4) \leftarrow -c(4)$

$$c = [\ 3 \quad -1 \mid 7 \quad -5 \mid 9 \quad 11 \mid 13 \quad 15\ ]^T$$

$i = 3$
$c(5 : 6) \leftarrow c(6 : -1 : 5)$

$$c = [\ 3 \quad -1 \mid 7 \quad -5 \mid 11 \quad 9 \mid 13 \quad 15\ ]^T$$

$c(6) \leftarrow -c(6)$

$$c = [\ 3 \quad -1 \mid 7 \quad -5 \mid 11 \quad -9 \mid 13 \quad 15\ ]^T$$

$i = 4$
$c(7 : 8) \leftarrow c(8 : -1 : 7)$

$$c = [\ 3 \quad -1 \mid 7 \quad -5 \mid 11 \quad -9 \mid 15 \quad 13\ ]^T$$

$c(8) \leftarrow -c(8)$

$$c = [\ 3 \quad -1 \mid 7 \quad -5 \mid 11 \quad -9 \mid 15 \quad -13\ ]^T$$

Iteration $i = 1, \dots, 4$ ends
Finally,

$$c = [\ 3 \quad -1 \quad 7 \quad -5 \quad 11 \quad -9 \quad 15 \quad -13\ ]^T$$

Next, we present the function LHD that reads a value t and returns a SOLHD of order $2^t \times 2^{t-1}$. As we mentioned before, depending on how a column was constructed, we apply the VD procedure accordingly in order to make new columns. Thus, we need a $1 \times 2^{t-1}$ vector d whose every entry $d(i)$ stores the information needed in order to construct new columns from column c_i. More specifically, $d(i)$ will have a value s which will correspond to the lengths $2^s, 2^{s+1}, ..., 2^{t-1}$ of each part in which column c_i will have to be divided. If $d(i) = t$, c_i will not be used for creating new columns.

function L=LHD(t)
$p \leftarrow 1$ %counter for the number of constructed columns
$d \leftarrow zeros(1, 2^{t-1})$
%We create matrix C, where we will store all the
%columns $c_i = C(:, i)$ that we will construct.
$C \leftarrow zeros(2^{t-1}, 2^{t-1})$
%Create $c_1 = C(:, 1)$
$C(:, p) \leftarrow [1 : 2 : 2^t - 1]^T$
$d(p) \leftarrow 1$
$p \leftarrow p + 1$
%Construct $c_2, ..., c_{2^{t-1}}$
for $i = 1 : 2^{t-1}$
  if $d(i) < t$
    for $j = d(i) : t - 1$
      $C(:, p) \leftarrow VD(C(:, i), j)$
      $d(p) \leftarrow j + 1$
      $p \leftarrow p + 1$
    end
  end
end
$L \leftarrow 1/2 \cdot [C^T, -C^T]^T$

Complexity of the algorithm

The function LHD, constructs a SOLHD of order $2^t \times 2^{t-1}$ without performing any floating point computations and thus without introducing any rounding error. It requires only permutations and negations of columns that are executed rapidly. In the next table we demonstrate the execution times required for the construction of SOLHDs of various orders.

t	Time (sec)
4	$1.999855041503906 \cdot 10^{-3}$
5	$3.000020980834961 \cdot 10^{-3}$
6	$9.000062942504883 \cdot 10^{-3}$
7	$1.900005340576172 \cdot 10^{-2}$
8	$4.099988937377930 \cdot 10^{-2}$
9	$9.299993515014648 \cdot 10^{-2}$
10	0.212000131607056
11	0.497999906539917
12	1.203000068664551

Table I
EXECUTION TIMES OF $LHD(t)$, FOR $t = 4, \ldots, 12$

B. An illustrative example

Next, we present the detailed execution of the function $LHD(t)$, for $t = 4$, that results in a SOLHD of order 16×8.

Example 2. Function $LHD(t)$, for $t = 4$
$p \leftarrow 1$, $d \leftarrow zeros(1, 2^{4-1})$, $C \leftarrow zeros(2^3, 2^3)$
$C(:, p) \equiv C(:, 1) \leftarrow [1 : 2 : 2^4 - 1]$

$$C(:, 1) = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \end{bmatrix}^T$$

$d(p) \equiv d(1) \leftarrow 1$, $p \leftarrow 2$
Iteration $i = 1, ..., 2^3$ begins:
$i = 1$
$d(i) \equiv d(1) = 1 < t \equiv 4$
Iteration $j = d(1), ..., 4 - 1 = 1, 2, 3$ begins:
$j = 1$
$C(:, p) \equiv C(:, 2) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 1), 1)$

$$C(:, 2) = \begin{bmatrix} 3 & -1 & 7 & -5 & 11 & -9 & 15 & -13 \end{bmatrix}^T$$

$d(p) \equiv d(2) \leftarrow j + 1 \equiv 1 + 1 = 2$, $p \leftarrow 3$
$j = 2$
$C(:, p) \equiv C(:, 3) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 1), 2)$

$$C(:, 3) = \begin{bmatrix} 7 & 5 & -3 & -1 & 15 & 13 & -11 & -9 \end{bmatrix}^T$$

$d(p) \equiv d(3) \leftarrow j + 1 \equiv 2 + 1 = 3$, $p \leftarrow 4$
$j = 3$
$C(:, p) \equiv C(:, 4) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 1), 3)$

$$C(:, 4) = \begin{bmatrix} 15 & 13 & 11 & 9 & -7 & -5 & -3 & -1 \end{bmatrix}^T$$

$d(p) \equiv d(4) \leftarrow j + 1 \equiv 3 + 1 = 4$, $p \leftarrow 5$
Iteration $j = 1, 2, 3$ ends
$i = 2$
$d(i) \equiv d(2) = 2 < t \equiv 4$
Iteration $j = d(2), ..., 4 - 1 = 2, 3$ begins:
$j = 2$
$C(:, p) \equiv C(:, 5) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 2), 2)$

$$C(:, 5) = \begin{bmatrix} -5 & 7 & 1 & -3 & -13 & 15 & 9 & -11 \end{bmatrix}^T$$

$d(p) \equiv d(5) \leftarrow j + 1 \equiv 2 + 1 = 3$, $p \leftarrow 6$
$j = 3$
$C(:, p) \equiv C(:, 6) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 2), 3)$

$$C(:, 6) = \begin{bmatrix} -13 & 15 & -9 & 11 & 5 & -7 & 1 & -3 \end{bmatrix}^T$$

$d(p) \equiv d(6) \leftarrow j + 1 \equiv 3 + 1 = 4$, $p \leftarrow 7$
Iteration $j = 2, 3$ ends
$i = 3$
$d(i) \equiv d(3) = 3 < t \equiv 4$
Iteration $j = d(3), ..., 4 - 1 = 3$ begins:
$j = 3$
$C(:, p) \equiv C(:, 7) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 3), 3)$

$$C(:, 7) = \begin{bmatrix} -9 & -11 & 13 & 15 & 1 & 3 & -5 & -7 \end{bmatrix}^T$$

$d(p) \equiv d(7) \leftarrow j + 1 \equiv 3 + 1 = 4$, $p \leftarrow 8$
Iteration $j = 3$ ends
$i = 4$
$d(i) \equiv d(4) = 4 \not< t \equiv 4$
$i = 5$
$d(i) \equiv d(5) = 3 < t \equiv 4$
Iteration $j = d(5), ..., 4 - 1 = 3$ begins:
$j = 3$
$C(:, p) \equiv C(:, 8) \leftarrow VD(C(:, i), j) \equiv VD(C(:, 5), 3)$

$$C(:, 8) = \begin{bmatrix} -11 & 9 & 15 & -13 & 3 & -1 & -7 & 5 \end{bmatrix}^T$$

$d(p) \equiv d(8) \leftarrow j + 1 \equiv 3 + 1 = 4$, $p \leftarrow 9$
Iteration $j = 3$ ends
$i = 6$

$d(i) \equiv d(6) = 4 \not< t \equiv 4$
$i = 7$
$d(i) \equiv d(7) = 4 \not< t \equiv 4$
$i = 8$
$d(i) \equiv d(8) = 4 \not< t \equiv 4$
Iteration $i = 1, ..., 2^3$ ends
$L \leftarrow 1/2 \cdot [C^T, -C^T]^T$

Finally,

$$L = \frac{1}{2}\begin{bmatrix} 1 & 3 & 7 & 15 & -5 & -13 & -9 & -11 \\ 3 & -1 & 5 & 13 & 7 & 15 & -11 & 9 \\ 5 & 7 & -3 & 11 & 1 & -9 & 13 & 15 \\ 7 & -5 & -1 & 9 & -3 & 11 & 15 & -13 \\ 9 & 11 & 15 & -7 & -13 & 5 & 1 & 3 \\ 11 & -9 & 13 & -5 & 15 & -7 & 3 & -1 \\ 13 & 15 & -11 & -3 & 9 & 1 & -5 & -7 \\ 15 & -13 & -9 & -1 & -11 & -3 & -7 & 5 \\ -1 & -3 & -7 & -15 & 5 & 13 & 9 & 11 \\ -3 & 1 & -5 & -13 & -7 & -15 & 11 & -9 \\ -5 & -7 & 3 & -11 & -1 & 9 & -13 & -15 \\ -7 & 5 & 1 & -9 & 3 & -11 & -15 & 13 \\ -9 & -11 & -15 & 7 & 13 & -5 & -1 & -3 \\ -11 & 9 & -13 & 5 & -15 & 7 & -3 & 1 \\ -13 & -15 & 11 & 3 & -9 & -1 & 5 & 7 \\ -15 & 13 & 9 & 1 & 11 & 3 & 7 & -5 \end{bmatrix}.$$

By evaluating the correlation matrix of L, we get

$$R = \frac{12}{16(16^2 - 1)} L^T L = I_8,$$

which means that L is orthogonal.

III. Comparison with other methods

In [7], Sun, Lin and Liu developed a method attaining the construction of symmetric orthogonal Latin hypercube designs of order $2^t \times 2^{t-1}$, $t \in \mathbb{N}$. Next, we briefly describe their approach.

The Sun et al. method

Step 1. For $c = 1$, let

$$S_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, T_1 = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}.$$

Step 2. For $c > 1$, define S_c and T_c as

$$S_c = \begin{bmatrix} S_{c-1} & -S^*_{c-1} \\ S_{c-1} & S^*_{c-1} \end{bmatrix},$$

$$T_c = \begin{bmatrix} T_{c-1} & -(T^*_{c-1} + 2^{c-1} S^*_{c-1}) \\ T_{c-1} + 2^{c-1} S_{c-1} & T^*_{c-1} \end{bmatrix},$$

where the * operator works on any matrix with an even number of rows by multiplying the entries of the top half of the matrix by -1 and leaving those in the bottom half unchanged.

Step 3. Let $H_c = T_c - S_c/2$, $L_c = (H_c^T, -H_c^T)^T$

Example 3. For $c = 1$:

$$S_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, T_1 = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix},$$

$$H_1 = \frac{1}{2}\begin{bmatrix} 1 & 3 \\ 3 & -1 \end{bmatrix} \text{ and}$$

$$L_1 = \frac{1}{2}\begin{bmatrix} 1 & 3 \\ 3 & -1 \\ -1 & -3 \\ -3 & 1 \end{bmatrix}.$$

For $c = 2$:

$$S_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix},$$

$$T_2 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & -1 & -4 & 3 \\ 3 & 4 & -1 & -2 \\ 4 & -3 & 2 & -1 \end{bmatrix},$$

$$H_2 = \frac{1}{2}\begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & -1 & -7 & 5 \\ 5 & 7 & -1 & -3 \\ 7 & -5 & 3 & -1 \end{bmatrix} \text{ and}$$

$$L_2 = \frac{1}{2}\begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & -1 & -7 & 5 \\ 5 & 7 & -1 & -3 \\ 7 & -5 & 3 & -1 \\ -1 & -3 & -5 & -7 \\ -3 & 1 & 7 & -5 \\ -5 & -7 & 1 & 3 \\ -7 & 5 & -3 & 1 \end{bmatrix}.$$

For $c = 3$:

$$S_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix},$$

$$T_3 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & -1 & -4 & 3 & 6 & -5 & -8 & 7 \\ 3 & 4 & -1 & -2 & -7 & -8 & 5 & 6 \\ 4 & -3 & 2 & -1 & -8 & 7 & -6 & 5 \\ 5 & 6 & 7 & 8 & -1 & -2 & -3 & -4 \\ 6 & -5 & -8 & 7 & -2 & 1 & 4 & -3 \\ 7 & 8 & -5 & -6 & 3 & 4 & -1 & -2 \\ 8 & -7 & 6 & -5 & 4 & -3 & 2 & -1 \end{bmatrix},$$

$$H_3 = \frac{1}{2}\begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \\ 3 & -1 & -7 & 5 & 11 & -9 & -15 & 13 \\ 5 & 7 & -1 & -3 & -13 & -15 & 9 & 11 \\ 7 & -5 & 3 & -1 & -15 & 13 & -11 & 9 \\ 9 & 11 & 13 & 15 & -1 & -3 & -5 & -7 \\ 11 & -9 & -15 & 13 & -3 & 1 & 7 & -5 \\ 13 & 15 & -9 & -11 & 5 & 7 & -1 & -3 \\ 15 & -13 & 11 & -9 & 7 & -5 & 3 & -1 \end{bmatrix}$$

and

$$L_3 = \frac{1}{2}\begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \\ 3 & -1 & -7 & 5 & 11 & -9 & -15 & 13 \\ 5 & 7 & -1 & -3 & -13 & -15 & 9 & 11 \\ 7 & -5 & 3 & -1 & -15 & 13 & -11 & 9 \\ 9 & 11 & 13 & 15 & -1 & -3 & -5 & -7 \\ 11 & -9 & -15 & 13 & -3 & 1 & 7 & -5 \\ 13 & 15 & -9 & -11 & 5 & 7 & -1 & -3 \\ 15 & -13 & 11 & -9 & 7 & -5 & 3 & -1 \\ -1 & -3 & -5 & -7 & -9 & -11 & -13 & -15 \\ -3 & 1 & 7 & -5 & -11 & 9 & 15 & -13 \\ -5 & -7 & 1 & 3 & 13 & 15 & -9 & -11 \\ -7 & 5 & -3 & 1 & 15 & -13 & 11 & -9 \\ -9 & -11 & -13 & -15 & 1 & 3 & 5 & 7 \\ -11 & 9 & 15 & -13 & 3 & -1 & -7 & 5 \\ -13 & -15 & 9 & 11 & -5 & -7 & 1 & 3 \\ -15 & 13 & -11 & 9 & -7 & 5 & -3 & 1 \end{bmatrix}.$$

In the next example, we prove that the 16×8 SOLHD produced by the VD algorithm is isomorphic to the one produced by the Sun et al. method, i.e. one can be obtained by the other by interchanging and negating columns.

Example 4. Matrix L of Example 2 and matrix L_3 of Example 3 are isomorphic. More specifically, by rearranging and negating the columns $c_1, ..., c_8$ of L as follows

$$\begin{bmatrix} c_1 & c_2 & -c_5 & c_3 & -c_7 & -c_8 & -c_6 & c_4 \end{bmatrix}$$

we get matrix L_3.

It can also be proved that the SOLHDs of orders 32×16, 64×32 and 128×64 constructed by the VD algorithm are isomorphic to the corresponding SOLHDs provided by the Sun et al. method. A general proof for the equivalence between the two approaches is under study.

IV. Conclusions

In this work we developed an algorithm constructing symmetric orthogonal Latin hypercube designs of order $2^t \times 2^{t-1}$. The algorithm requires no floating point operations and thus it is effective and introduces no rounding error. The obtained Latin hypercube designs have good properties and can be used in many statistical applications. We compared our method with the one introduced by Sun et al. and we concluded that the resulting designs are isomorphic. A general proof for the equivalence between the two methods is under study. For future work, we consider combining this algorithm with other methods in order to produce Latin hypercube designs of other orders and with different properties.

References

[1] Cioppa, T.M., Lucas, T.W., Efficient nearly orthogonal and space-filling Latin hypercubes, Technometrics, 49, 45-55, 2007

[2] Guo, B., Li, X.R., Liu, M.Q., Yang, X., Construction of orthogonal general sliced Latin hypercube designs, Statistical Papers, 64, 987-1014, 2023

[3] Li, H., Yang, L., Liu, M.Q., Construction of space-filling orthogonal Latin hypercube designs, Statistical Probability Letters, 180, January 2022, 109245

[4] Lin, C.D., New Development in Designs for Computer Experiments and Physical Experiments, Ph.D. thesis, Simon Fraser University, 2008

[5] Lin, C.D., Bingham, D., Sitter, R.R., Tang, B., A new and flexible method for constructing designs for computer experiments, Annals of Statistics, 38, 1460-1477, 2010

[6] McKay, M.D., Beckman, R.J., Conover, W.J., A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics, 21, 239-245, 1979

[7] Sun, F.S., Liu, M.Q., Lin, D.K.J., Construction of orthogonal Latin hypecube designs, Biometrika 96, 971-974, 2009

[8] Sun, F.S., Liu, M.Q., Lin, D.K.J., Construction of orthogonal Latin hypecube designs with flexible run sizes, Journal of Statistical Planning and Inference, 140, 3236-3242, 2010

[9] Wei, Q., Yang, J.F., Liu, M.Q., Column expanded Latin hypercube designs, Journal of Statistical Planning and Inference, 234, January 2025, 106208

[10] Ye, K.Q., Orthogonal column Latin hypercubes and their application in computer experiments, Journal of the American Statistical Association, 93, 1430-1439, 1998