Generating software for MUB complementary sequence constructions

Hanieh Roodashty

November 2015

Master Thesis

Department of Informatics
University of Bergen
Norway

To my
parents,
husband,
sisters and
daughter.

Acknowledgements

First of all I would like to express my gratitude to my supervisor, Professor Matthew Geoffrey Parker, for providing me with the opportunity to work in the interesting world of MUBs.

I am very thankful to Mari Garaas Løchen for being so forthcoming, punctual and helpful whenever I need her for educational and administration issues.

Tor Helleseth and Tetiana Yarigina deserve to be acknowledged for the great inputs within classical cryptosystems and basic code.

I started my academic training in Norway with the Master program in Telematics at the Norwegian University of Science and Technology n Trondheim. I would like to thank my professors at NTNU and especially the course coordinator Laurent Paquereau for the valuable academic experience in Trondheim.

Last but not least, I would like to thank my lovely husband Mahdi and my cute daughter Rihanna for always being there for me. I am also deeply grateful to my family members back home, my parents and sisters for their unconditional love and support.

Hanieh Roodashty

Bergen, November 2015

Summary

This master thesis has been performed at the Department of Informatics, University of Bergen between February and November 2015. The work has been supervised by Professor Matthew G. Parker as a part of the research interest within complementary construction using mutually unbiased bases.

This project is an attempt in the line of the study to improve the set size of the complementary sequences while keeping the upper bound of PAPR as low as possible and also maintaining the pairwise distinguishability. To perform this task, seeding the recursive construction with optimal mutually-unbiased bases in dimension 2 and 3 was used in this study.

To use the MUB-based sequences in OFDM containing systems, we generated program codes that constructed distinct arrays and sequences for dimension 2 and 3 seeding by MUBs with and without linear offset. The codes were produced in MATLAB environment.

The codes for both dimensions have delivered satisfactory results. The results for lower iterations have also matched with the manually calculated values based on theory.

Various strategies were used to increase the software speed as well as to decrease the resource demand, but still to run the codes for higher iterations there needs advanced and professional computing solutions such as supercomputers.

It has been attempted to generate the codes with maximum possible flexibility so that they can be used for other dimensions with minor adjustments. The codes have also the capability of conversion to other programming languages.

Table of Contents

List of Figures

List of Tables

Symbols and Nomenclature

Roman letters

A	Amplitude
c^d	Hilbert Space
n	Iteration +1
i	Iteration
d	Representative prime number

Abbreviations

AM	Amplitude Modulation
ASK	Amplitude Shift Keying
CDMA	Code Division multiple Access
DTE	Data Terminal Equipment
FDM	Frequency Division Multiplexing
FM	Frequency Modulation
FSK	Frequency Shift Keying
IFFT	Inverse Fast Fourier Transformation
MUB	Mutually Unbiased Base
OFDM	Orthogonal Frequency Division Multiplexing
PAPR	Peak to Average Power Ratio
PM	Phase Modulation
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature-PSK
TDM	Time Division Multiplexing
WDM	Wave-Length Division Multiplexing

Chapter 1

Introduction

An introduction to the master thesis work is provided in this chapter. The chapter begins with an explanation of the background of the work and then describes the drive and motivation for generating the MUB complementary sequence constructions.

1.1 Background

The word “telecommunication” adopted from the French word télécommunication consisting of the Greek prefix tele (τηλε-) which means "distant" and the Latin word communicare, meaning "to share" [1]. Telecommunication then literally means to share information from distance. Today, telecommunication is the act of exchanging information between two or more entities including the use of technology and it has its own developed branch of scientific research.

In telecommunications, communications systems consist of transmission systems, communication channels and also communication equipment at the end of a communication link that are called Data Terminal Equipment (DTE).

Transmission technologies are closely connected to physical layer protocols. This layer includes modulation, demodulation, error control and multiplexing.

By multiplexing, several signals can be transferred through the same channel so as to use the available bandwidth capacity in an efficient way.

The following are some of the most well-known techniques for multiplexing that are widely used nowadays:

- Frequency Division Multiplexing (FDM)
- Time Division Multiplexing (TDM)
- Wave-Length Division Multiplexing (WDM)
- Code Division multiple Access (CDMA)

Modulation is the process of conveying a data signal on another signal that can be physically transmitted. By Modulation, the data signal is transferred from Base Band to the frequencies of Band Pass. Modulation itself can be either analog or digital. Important types of digital modulation are:

- ASK (Amplitude Shift Keying)
- FSK (Frequency Shift Keying)
- PSK (Phase Shift Keying)
- QPSK (Quadrature-PSK)
- QAM (Quadrature Amplitude Modulation)
- Orthogonal Frequency Division Multiplexing (OFDM) modulation

By using OFDM [2][3] as a multiplexing technique, each carrier signal is modulated with its own specific frequency where each of them is orthogonal to all other carriers. Then all the signals can be closely packed even with overlaps.

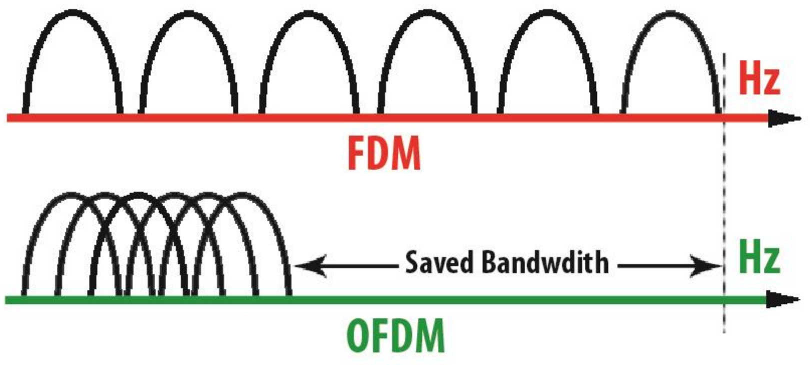


FIGURE 1-1: Comparison of FDM and OFDM approach to bandwidth consumption.

OFDM has several applications in industry including but not limited to high data rate wireless systems, telecommunication standards like 802.11a, 802.11g and mobile communications.

OFDM is well known for the possibility to send higher bit rates in a given bandwidth using less frequency compared to other techniques, yielding higher bandwidth efficiency. It is also a very resistant method to frequency selective fading if compared to single carrier systems. Another advantage of OFDM is not being sensitive to time synchronization errors.

On the other hand, there are some weak points associated with the OFDM technique where the most important one is the high peak to average power ratio (PAPR) [2][4] that this master thesis addresses. Sensitivity of OFDM to frequency synchronization is also a minor issue to be resolved when working with this method.

The peak-to-average-power ratio (PAPR) is the peak amplitude squared (this is the peak power) of the waveform divided by the root mean square (RMS) value squared (this is average power) of the waveform as shown in equation (1-1).

$$\mathrm{PAPR}_{\mathrm{db}} = 10\log_{10}\frac{|x|\mathrm{peak}^2}{x_{\mathrm{RMS}}{}^2} \tag{1-1}$$

If the wave type is OFDM, the value of PAPR in typical systems may be around 12 dB and in QPSK is 0 dB and in QAM is equal to 3.7 dB meaning that OFDM has the highest PAPR among available systems. This is a big concern associated with the OFDM systems.

High PAPR for the OFDM systems may be caused by independent modulated subcarriers coupled together to form a signal to be transmitted. The subcarriers in this case have been formed via an inverse fast Fourier transformation (IFFT) operation so the transmitted signal does not have a flat inverse Fourier spectra.

The importance of high PAPR will enforce some restrictions and also further significant issues for power amplifiers in transmitters [4]. In one hand the power amplifiers work efficiently when the PAPR is low. On the other hand, high PAPR makes the peak of the signal move to the nonlinear region of the power amplifier and reduces the efficiency of power amplification. Additionally high PAPR signals require a larger range of dynamic linearity than analog circuits, which often results in expensive devices.

Consequently, there is a motivation to do research on the current issues and shortcomings of OFDM including its high PAPR, leading to properly utilize the capabilities of OFDM.

As of now there have been several methods suggested to address this issue by increasing the average values of the peak powers in order to decrease the ratio of the peak power to the peak average or PAPR. By decreasing the PAPR, there will be a decrease in bit rate, an increase in the bandwidth efficiency and also an increase in the bit errors.

In this project we will emphasize on a technique that uses a designed code [4][5][6] to simultaneously possess both low PAPR and appropriate error correction capabilities.

In 1999 Davis and Jedwab [5] suggested a coding scheme for OFDM systems using binary modulation with a high code rates for moderate carries. They showed that it is feasible to form standard 2^h-array complementary sequences where the resulting length would be 2^n. These sequences had second- order cosets of first-order Reed-Muller codes that are from the $RM2^h$ (1, n) family [7][8][9][10]. This in practice means that the sequences are pairwise distinguishable. Some other researchers such as [7][11][12][13] [14][15][16] exhibited that the constructed arrays have the same structure as the complementary set construction. In this case, the sequences are provided by the projection of the arrays. Although this was a good idea, the set size alteration could lead to a change in upper bound of PAPR and also pairwise distinguishability.

It is then of great interest to be able to enhance the set size of the complementary sequences while the upper bound of PAPR keeps being as low as reasonably achievable and also the pairwise distinguishability is fulfilled. The method to meet all these criteria has been proposed by these references [7][17][18][19] where Mutually Unbiased Bases (MUBs) are central to the sequence constructions.

It has been shown [7] that practical MUBs for telecommunication purposes should have the following characteristics:

- Each sequence has a near-flat Fourier spectrum.
- Pairwise distance between any two sequences: minimum
- Sequences set is as large as possible (with near-flat constraint)

In order to construct the complementary sequences with the abovementioned properties, Matthew G. Parker and Gaofei Wu [7] suggested an algorithm to uniquely generate complementary sets of arrays exploiting a set of MUBs. The suggested algorithm can potentially be seeded of any MUB with any dimension and that will consequently need the corresponding formulation of the size of arrays and sequences.

Addressing the practical need to have the outcome of this algorithm, we need a software package that can help generate the relevant arrays and sequences. In this project we have focused on a MUB of dimension 2 as studied in the reference [7] with some modifications and also a MUB of dimension 3. The resulting software of this project is aimed to uniquely generate arrays and sequences and the number of the generated items.

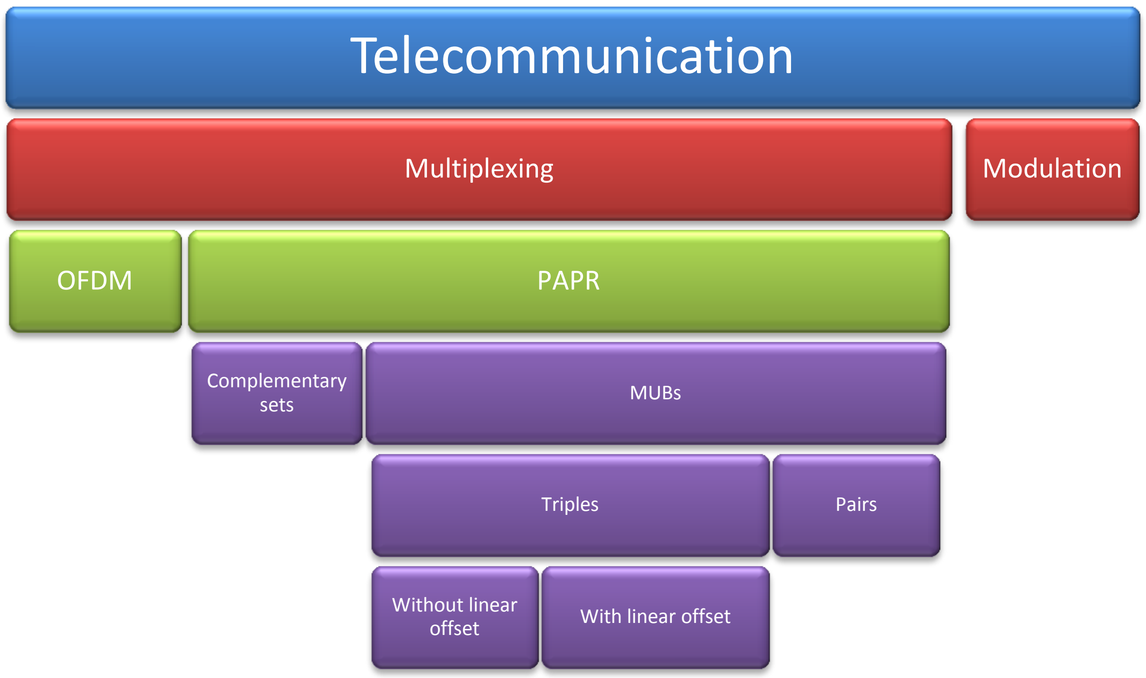


FIGURE 1-2: Overview of the hierarchy and the project concept

Figure 1-2 shows an overview of the important subjects connected to telecommunication technology from one end and the content of this thesis from the other end.

1.1.1 Problem statement

The algorithm suggested [7] for the unique generation of MUB based arrays and sequences were considered. The number of complementary sequences in dimension 2 and dimension 3 initially require computerized approaches to generate them.

These enumerations then help to confirm the associated theoretical derivations.

The current master thesis is focused on developing software to address the need of MUB complementary sequence constructions in the two mentioned dimensions. The results of the thesis can be instrumental to systematically verify the results of the already suggested algorithms for uniquely generation of MUB based arrays and sequences.

1.1.2 Thesis structure

The thesis consists of four chapters, bibliography and two appendices.

In chapter 1 we introduced the concept and applications of MUBs and the statement of the problem to be addressed in the thesis.

In chapter 2 the details of the algorithm and the steps taken to prepare the code are stated.

Chapter 3 comprises the results and outputs of the code and relevant discussions.

The conclusions and recommendations for future work are reflected in chapter 4.

Finally the source code written for the thesis following a brief user manual is presented as the appendices.

Chapter 2

Methodology

This chapter briefly presents the concept and application of OFDM technique and the methodology for generating MUB-based sequences and arrays. The chapter eventually presents the step-by-step design of the software generated in this thesis.

2.1 **OFDM**

Modulation in telecommunication is the process of carrying an information signal on another signal, called a carrier signal, which can be physically transmitted in order to have a high efficiency of transmission.

When modulating on a data signal occurs, one property of the carrier signal, for instance amplitude, frequency or phase changes according to the changes in data signal.

Some common modulation techniques are amplitude modulation (AM), frequency modulation (FM), phase modulation (PM) and orthogonal frequency division multiplexing (OFDM) modulation.

By multiplexing, multiple digital or analog data are combined into one signal so that by sharing, expensive resources in transferring channels can be used more cost effectively.

OFDM is a combination of modulation and multiplexing [20]. The OFDM is a form of multi-carrier modulation technique. In this technique the bandwidth is shared among multiple individual modulated data signals.

OFDM in many ways is similar to conventional frequency division multiplexing (FDM). One of the minor differences is the method of signal modulating-demodulating.

FDM is also a multi-carrier technique that divides the bandwidth of the channel among all carriers. There is no relationship between the sub-carriers of the data signal. Each of them has its own frequency and is modulated individually. A part of this method is to provide enough frequency space between each data carrier. There will be no overlap between the data carriers so that they can be distinguishable at the receiver side. In this method the use of resources of the channel is not efficient since there is much bandwidth wasted.

On the other hand, in OFDM, the frequencies of carriers are chosen so that the carriers are orthogonal to each other. Overlap between the data carriers is allowed here and as a result the frequency space is not required in this technique. This practically means that we can generate a larger number of carriers over a channel with OFDM compared to FDM and then more bitrate can be transmitted in lower frequency bandwidth.

Another strong point for OFDM is the capability to be adapted to environments containing high radio frequency (RF) interference.

Finally, OFDM is more practical in harsh multi-path environments.

These advantages make OFDM outstanding among other multi carrier modulation schemes.

OFDM has been adopted by several technologies including but not limited to [2]:

- Asymmetric Digital Subscriber Line (ADSL) services
- IEEE 802.11a, g, n (WLANs)
- IEEE 802.20 Mobile Broadband Wireless Access
- IEEE 802.15.3a (Wireless PAN)
- IEEE 802.16d, e (WiMAX),
- DAB (Digital Video Broadcast)
- DVB-T (digital terrestrial television broadcast)
- DVB-H: Digital Broadcast Services to Handheld Devices

The technology of OFDM is in the early stage of developments and naturally there are still some issues to be resolved. First of all, OFDM is fairly sensitive to Doppler shift and in general it does not give proper results when subjected to offset in carrier frequency.

The OFDM systems also possess large dynamic range amplitude. Therefore we need radio frequency power amplifiers with a high peak–to-average-ratio (PAPR) to handle the incident signals when compared to single-carrier systems. This high PAPR can be nominated as the greatest challenge hindering a mass use of OFDM in industry.

As discussed in the first chapter, the PAPR is calculated from the equation (2-1):

$$\mathrm{PAPR_{db}} = 10\log_{10}\frac{|\mathrm{x}|\mathrm{peak}^2}{\mathrm{x_{RMS}}^2} \tag{2-1}$$

In a case that we have N signals all with the same phase, the resulting peak power of them will be N times their average power. As a consequence there will be a high PAPR wave in the system that needs severe amplifying to be transmitted [6]. This shows the importance of characterization PAPR in OFDM systems and solving the problem of high PAPR.

There has been significant research into the PAPR problem making it a central topic in OFDM systems [21], [22]. Furthermore there have been many attempts to reduce the PAPR through various techniques such as coding schemes [23][24], clipping [25], [26], phase optimization [27], Tone Reservation (TR) and Tone Injection (TI)[28] and Partial Transmission Sequence (PTS) and Selective Mapping (SLM) [29], [30]. These methods are basically either of the signal distortion type (clipping for instance) or signal scrambling techniques e.g. block codes and PTS.

It has been proposed that an appropriate compromise needs to be made between the extent of reducing PAPR and the power consumption for transmission, data loss rate, Bit-Error-Ratio (BER) performance and also practicality of the system implementation [4].

An interesting idea to effectively reduce the PAPR has been suggested in [23] and later [31] to utilize block coding via selecting a set of code words. Selecting the right code words with considerations of M-ary phase modulation scheme and the coding rate suitable for encoding-decoding are keys in this technique. This method however requires a comprehensive search operation to select the right code words which can be resource demanding for search, storage and encoding operations. The technique is also problematic when it comes to the error correction.

Golay complementary sequences were later suggested as code words [8] to decrease the PAPR of the signal to 2 or values less than 2 which was a breakthrough at the time.

A development in this field later proposed by Davis and Jedwab was that a given second-order cosets of generalized first order Reed–Muller codes RM_{2h} (1,n) can form large sets of binary length 2^n Golay complementary pairs [5]. They eventually used the joint effect of block coding and Golay complementary sequences to enjoy the strongpoints of both and at the same time to address their shortcomings through the combination. As a result, their outcome had a reduced PAPR, while maintaining error correction within an acceptable range, included high code rates and also effective encoding-decoding capabilities.

Later several research works have shown that the structure of the complementary sets are in fact array structures [14][15][13][12][16] and the sequence sets can be generated by projections of the arrays. It is then of interest to have complementary constructions that can expand set size without significantly changing the upper bound of PAPR or the pairwise distinguishability.

Finally it has been showed [7] that the issue of construction of large sets of complementary sequences while preserving proper pairwise distinguishability can be addressed by seeding the recursive construction with optimal mutually-unbiased bases (MUBs) [19][17][18].

2.2 **Optimal mutually-unbiased bases (MUB)**

The first person to initiate the notion of MUB was Schwinger who showed the concept in 1960 [19]. Then the first researcher to study the applications of MUB was Ivanovic [17] who used MUBs for quantum state determination.

Nowadays MUBs have a lot of important applications in quantum computation including quantum state tomography and quantum cryptographic schemes [32][33][34]. MUB problems also include dozens of mathematical contributions that have been developed in the context of communication theory and there is potential for progress in the field such as the use of MUBs to construct complementary sequences that is a central part of the present thesis.

In quantum theory, a pair of bases $\{u_1,\ldots,u_d\}$ and $\{v_1,\ldots,v_d\}$ in Hilbert space c^d [35] is called mutually unbiased if they are orthonormal and the square of the magnitude of the inner product between any two bases u_i and v_j equals the inverse of the dimension d [32] as shown in equation(2-2):

$$\Delta^2\,(u_j, v_j) = \left|< u_i, v_j >\right|^2 = \frac{1}{d} \qquad \textit{(2-2)}$$

A conventional challenge with MUBs is how to find the maximum number of mutually unbiased bases in the d-dimensional Hilbert space c^d.

This problem was first addressed by Ivanovic where d was a prime number [17] and then by Wootters and Fields where *d* was an integer power of a prime number [35] but it is still an open question [36] for arbitrary *d*.

Approaching the upper bound, Wootters and Fields proved [35] that it is not possible to find more than d+1 MUB in any d-dimensional Hilbert space c^d. If there exist d+1 MUBs in a Hilbert space where d is a power of a prime number, the MUB is considered an optimal MUB.

For the lower bound on MUBs, if *d* is a prime number decomposition i.e. $d = P_1^{n1} P_2^{n2} \dots P_k^{nk}$ and with $P_1^{n1} < P_2^{n2} < \cdots < P_k^{nk}$ then $P_1^{n1} + 1$ will be the minimal MUB.

In this case, the number of constructed MUBs obeys the following equation:

$$P_1^{n1} + 1 \le \#MUB \le d + 1 \qquad (2\text{-}3)$$

2.2.1 MUB for constructing large sets of complementary pairs

A method has already been presented to construct large sets of complementary sequences by seeding with optimal MUB by G. Wu and M. G. Parker [7]. They considered an optimal MUB as M_d and constructed complete sets of complementary sequences that are seeded by a MUB of dimension 2 (M_2) although their main construction works on any MUB of any dimension. But developing formula for the size of the arrays and sequences sets for higher MUB constructions is still open. In this work they developed techniques for constructing sequence and arrays by an optimal MUB of dimension 2. We generated the same arrays and sequences that were seeded by the MUB of dimension 2 and also dimension 3 these will be discussed in the next chapter.

2.2.1.1 Using M_2 to construct complementary pairs

In the reference mentioned above [7] it was considered that M2={I, H, N} where matrices $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and $N = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix}$. Here it should be noted that *i* equals $\sqrt{-1}$.

I, H and N are unitary matrices meaning that HH*=H*H=I and NN*=N*N=I where I is identity matrix and * means transpose conjugate. In addition to these matrices, the Pauli matrix $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ was also considered a permutation matrix that could be used for constructing complete sets of MUBs of dimension 2. Here a set of complementary array pairs were constructed over the alphabet {0, 1, i, -1, -i}.

We shall let F(z) define a complementary set with length d sequences and with degree d-1 polynomials in variable z. $F_k(z)$ is a univariate polynomials with degree d-1 where the coefficients of these polynomials are considered as sequences.

$F(\mathbf{z})$ is also considered as a complementary set of n-dimensional $d_{k,0} \times d_{k,1} \times \ldots \times d_{k,n-1}$ arrays and $F_k(\mathbf{z})$ is a multivariate polynomial in variables z $=\{z_0, z_1,\ldots,z_{n-1}\}$ where the coefficients of these polynomials will be considered as arrays.

We can define arrays and sequences as generalized Boolean functions if we write: $f_k(x): \mathbb{F}_2^n \rightarrow A$ where A={0,1,i, -1, -i}.

The set of complementary pairs of arrays $F(\mathbf{z})$ are constructed as a recursive function and B_n set is the set of distinct complementary arrays that is obtained from $F(\mathbf{z})$.

F(z) is constructed from $F(\mathbf{z})$ by projections $z^{2^{\pi(i)}}$, $0 \leq i < n$ and fixed $B_{\downarrow,n}$ as the set of unique complementary sequences that can be obtained from F(z).

As an example, we can imagine that the array $F_k(\mathbf{z}) = 1 - z_0 + z_1 + z_0 z_1$ can be projected down to a univariate polynomial by assigning $z_0=z$,$z_1=z^2$. This means that $F_k(z)= 1-z+z^2+z^3$ and by the assignment $z_0=z^2$, $z_1=z$ the result will be $F_k(z)= 1+z-z^2+z^3$.

Equation *(2-4)* is valid when a set of complementary pairs are seeded with M2={I, H, N}:

$$\Delta^2(B_{\downarrow,n}) = \max\{\Delta^2(u,v) \mid u \neq v, u,v \in B_{\downarrow,n}\} = \frac{1}{2} \qquad (2\text{-}4)$$

2.2.1.1.1 The complementary set construction

The recursive function construction to generate a set of complementary arrays is:

$$F_j(z_j) = P_j u_j V_j(z_j) F_{j-1}(z_{j-1}) \qquad (2\text{-}5)$$

where d=S=2, $P_j = \{I, X\}$ is permutation unitary, $u_j \in M_2$ and $V_j(z_j) = \begin{pmatrix} 1 & 0 \\ 0 & z_j \end{pmatrix}$.

A set of complementary sequences can be then constructed by projecting down the arrays where the sequences possess PAPR≤ S=d since every member of M_d is a unitary matrix and finally the size of the projected sequence set can be increased by selecting d! permutations of the rows of the unitary matrices at each stage of recursion. We will perform the recursive function (2-5) *n* times so as to reconstruct $F_{n-1}(\boldsymbol{Z}) = \begin{pmatrix} F_{n-1,0}(\boldsymbol{z}) \\ F_{n-1,1}(\boldsymbol{z}) \end{pmatrix}$ where $F_{n-1,k}(\boldsymbol{Z}) = c \sum_{x \in \mathbb{F}_2^n} f_{n-1,k}(x) \boldsymbol{z}^x$, $k \in \{0,1\}$, $\boldsymbol{z}^x = \prod_{j=0}^{n-1} z_j^{x_j}$, and *c* is some real constant such that $F_{n-1,k}(\boldsymbol{z})$ is normalized as an array. $f_{n-1,k}$ is defined with three conditions as follows:

1- uj=H and Pj=I, ∀j

In this case $f_{n-1,k}(x): \mathbb{F}_2^n \to \{1,-1\} = i^{2(kx_{n-1}+\sum_{j=0}^{n-2} x_j x_{j+1})}, k \in \{0,1\}$ which are binary complementary sequences as constructed in [5].

An example of this function is:

$u = (H, H, H) \Rightarrow f_{2,0}(x) = i^{2(x_0x_1+x_1x_2)}$

This function is illustrated by Figure 2-1a.

2- $u_j \in \{H, N\}$ and $l = (j, u_j = N)$

Here we have $f_{n-1,k}(x): \mathbb{F}_2^n \to \{1, i, -1, -i\} = i^{2(kx_{n-1}+\sum_{j=0}^{n-2} x_j x_{j+1})+\sum_{j=0}^{|l|-1} x_{l(j)}}$. These are quaternary complementary sequences as discussed in [5].

An example of this function is $u = (H, N, H) \Rightarrow f_{2,0}(x) = i^{2(x_0x_1+x_1x_2)+x_1}$ which is schematically shown in in Figure 2-1b.

3- $u_j \in \{I, H, N\}$, where $u_{n-1} \neq I, p = (j, u_j \in \{H, N\})$,s=(j,uj=I), $q(v) = j$ if $u_j \neq I$ and $u_i = I, \forall i, v < i < j, j < n, j \neq v$ and $q(v) = n$ otherwise.

$f_{n-1,k}(x): \mathbb{F}_2^n \to A = (\prod_{j=0}^{|s|-1}(x_{s(j)} + x_{q(s(j))} + 1))i^{2(kx_{p(|p|-1)}+\sum_{j=0}^{|p|-2} x_{p(j)}x_{p(j+1)})+\sum_{j=0}^{|l|-1} x_{l(j)}}$ was then constructed where p(-1)=n, x_n=0, and A=$\{1, i, -1, -i\}$.

An example of this function is

$$u = (H, I, I, N) \Longrightarrow f_{3,0}(x) = (x_1 + x_3 + 1)(x_2 + x_3 + 1)i^{2(x_0x_3)+x_3}$$

This function is also schematically presented in Figure 2-1c.

More generally, if $u_j \in \{I, H, N\}$ and for some t, $u_{n-1} = u_{n-2} = \cdots = u_{n-t} = I, 0 \leq t \leq n$, and $u_{n-t-1} \neq I$, then define b such that b(j)=1 for $j \geq n-t$, and b(j)=0 otherwise.

Then they constructed

$$f_{n-1,k}(x):$$

$$\mathbb{F}_2^n \to A, = (\prod_{j=0}^{|s|-1}(x_{s(j)} + x_{q(s(j))} + kb(s(j)) + 1))i^{2(kx_{p(|p|-1)}+\sum_{j=0}^{|p|-2} x_{p(j)}x_{p(j+1)})+\sum_{j=0}^{|l|-1} x_{l(j)}},$$

Where p(-1)=n, xn=0, and A=$\{1, i, -1, -i\}$.

An example of this function is

$$u = (N, H, I, I) \Longrightarrow f_{3,0}(x) = (x_2 + k + 1)(x_3 + k + 1)i^{2(kx_1+x_0x_1)+x_0}$$

This function is shown in Figure 2-1d.

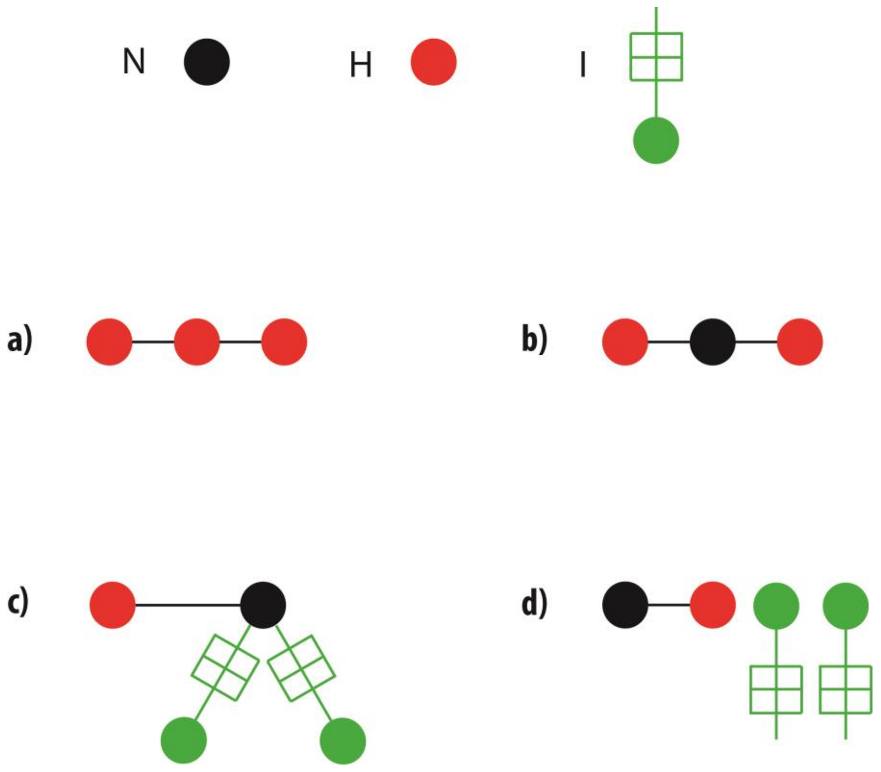


FIGURE 2-1: Graphical representation of the example functions

For drawing figure 1, they used the graphical language of [5][8][37][38][39] .

2.2.1.1.2 Enumerating arrays in B_n and sequences in $B_{\downarrow,n}$

We have already defined B_n but we also need to define $|B_{\downarrow,n}|$ which is the number of complementary sequences of length 2^n that can be generated from arrays in B_n by projection $z^{2^{\pi(i)}}$, $\pi \in S_n$ from the n-dimensional arrays down to one dimensional sequences of length 2^n. It is important to evaluate $|B_n|$ and also $|B_{\downarrow,n}|$ and to determine the number of arrays in B_n. It has been shown that the number of distinct arrays in B_n can be determined by:

$$|B_n| = \sum_{m=0}^{n} |B_n'|.2^{n-m} = \begin{cases} 2^{n-1}.\left(3^n + 3.3^{\frac{n}{2}} - 2\right), for\ n\ even, \\ 2^{n-1}.\left(3^n + 5.3^{\frac{n-1}{2}} - 2\right), for\ n\ odd, \end{cases} \tag{2-6}$$

where $|B_0'| = 1$.

Also for $|B_{\downarrow,n}|$, a recursive algorithm was first found [7] that could generate all sequences in $|B_{\downarrow,n}|$ uniquely. The algorithm was implemented first to generate unique sequences for

$u \in \{I, H\}^n$ and then for $u \in M_2^n$. For each case a mathematical relation for calculating the number of all complementary sequences has been formulated from the arrays in B_n by projection.

For each $F_{n-1,0}(\mathbf{z})$, there are n! possible projections, but not all these projections are unique. The reason is the likelihood that two IH strings generate the same sequence. In this situation there is a need to enforce some restrictions on the allowed permutation for generating unique sequences as described in details in [7]. To calculate the unique sequences in $|B_{\downarrow,n}|$ this equation can be used:

$$E_{IHN} = 3\sum_{k=0}^{n} 2^{k-2} k! \left\{ {n \atop k} \right\} + 2^n - \frac{1}{2} \qquad (2\text{-}7)$$

where $P_j \in \{I\}$.

Finally we can calculate all sequences in $|B_{\downarrow,n}|$ by the following equation where $P_j \in \{I, X\}$:

$$|B_{\downarrow,n}| = 2^n E_{IHN}(n) \qquad (2\text{-}8)$$

shows the number of unique IHN sequences where $P_j \in \{I\}, \forall j$ for n=1 to 6

TABLE 2-1: The number of unique IHN sequences where $P_j=\{I\}$ and n=i+1.

Parameter	Value					
n	1	2	3	4	5	6
$E_{IHN}(n)$	3	11	63	563	6783	99971
$\log_2(E_{IHN}(n))$	1.58	3.46	5.98	9.14	12.73	16.61
$\log_2(n!\,2^{n-1})$	0	2	4.58	7.58	10.91	14.49

As a result the PAPR upper-bound of the sequences in $|B_{\downarrow,n}|$ is 2. The reason for this is that I, H and N are $d \times d$ unitary, the value of $|B_{\downarrow,n}|$ is large since $|M_2| = 3 = d + 1$,the maximum value possible, and the value of $\Delta^2(B_{\downarrow,n})$ is small since $\Delta = \frac{1}{d} = \frac{1}{2}$ for M_2 is the minimum value possible here.

2.2.1.2 Using M_3 to construct complementary triples

How to determine a formula for the size of the array and sequence sets for an optimal MUB of dimension 3 is still an open question. We however generated unique arrays and sequences and attempted to calculate the number of distinct arrays and sequences for dimension 3 by our software in a similar manner to what has been done for dimension 2. Here the unique optimal MUB for d= 3 is obtained by $M_3=\{I, F, FD, FD^2\}$.

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2-9}$$

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \omega & 0 \\ 0 & 0 & \omega \end{pmatrix} \tag{2-10}$$

$$F = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{pmatrix} \tag{2-11}$$

$$W = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \tag{2-12}$$

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{2-13}$$

where $\omega = e^{\frac{2\pi i}{3}}$.

The recursive function construction to generate a set of complementary arrays is:

$$F_j(z_j) = P_j u_j V_j(z_j) F_{j-1}(z_{j-1}) \tag{2-14}$$

where $P_j \in P_{jk} = \{W^j X^k \,|\, j\epsilon\{0,1\}, k\epsilon\,\{0,1,2\}\}$ are all permutations of 3x3 matrices, $u_j \in M_3$ and $V_j(z_j) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & z_j & 0 \\ 0 & 0 & z_j^2 \end{pmatrix}$. Based on this formula we started generating matrices with vector $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. We also constructed a set of complementary arrays and sequences over the alphabet $\{0, 1, \omega, \omega^2\}$.

Using the following algorithm, the Truth-Table (TT) in modular 3 was converted to Algebraic Normal Form (ANF):

Let us start by an example that TT=[0 0 1 0 2 0 1 2 2]

The ANF is given by $A \otimes A \times TT$ where $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix}$, so the ANF [0 1 2 1 2 0 2 1 2]= $x_0 + 2x_0^2 + x_1 + 2x_0x_1 + 2x_1^2 + x_0x_1^2 + 2x_0^2x_1^2$. Now if we apply {IF} i.e. u_1=I, u_0=F where P_0, P_1={I} in the recursive function *(2-8)* we will have:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & z_1 & 0 \\ 0 & 0 & z_1^2 \end{pmatrix} \begin{pmatrix} 1 + z_0 + z_0^2 \\ 1 + \omega z_0 + \omega^2 z_0^2 \\ 1 + \omega^2 z_0 + \omega z_0^2 \end{pmatrix} = \begin{pmatrix} 1 + z_0 + z_0^2 \\ z_1 + \omega z_0 z_1 + \omega^2 z_0^2 z_1 \\ z_1^2 + \omega^2 z_0 z_1^2 + \omega z_0^2 z_1^2 \end{pmatrix}$$

So the TT and ANF for corresponding phases and magnitudes are:

$$\mathrm{TT} \rightarrow \mathrm{ANF} \qquad\qquad \mathrm{TT} \rightarrow \mathrm{ANF}$$

$$\text{phase: } \begin{bmatrix}0\\0\\0\end{bmatrix} \rightarrow \begin{bmatrix}0\\0\\0\end{bmatrix}, \text{Magnitude: } \begin{bmatrix}1\\0\\0\end{bmatrix} \rightarrow \begin{bmatrix}1\\0\\2\end{bmatrix}$$

$$\Rightarrow (2x_0^2 + 1)\omega^0 \equiv 1 + z_0 + z_0^2$$

$$\text{phase: } \begin{bmatrix}0\\1\\2\end{bmatrix} \rightarrow \begin{bmatrix}0\\1\\0\end{bmatrix}, \text{Magnitude: } \begin{bmatrix}0\\1\\0\end{bmatrix} \rightarrow \begin{bmatrix}0\\2\\2\end{bmatrix}$$

$$\Rightarrow (2x_1^2 + 2x_1)\omega^{x_0} \equiv z_1 + \omega z_0 z_1 + \omega^2 z_0^2 z_1$$

$$\text{phase: } \begin{bmatrix}0\\2\\1\end{bmatrix} \rightarrow \begin{bmatrix}0\\2\\0\end{bmatrix}, \text{Magnitude: } \begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \begin{bmatrix}0\\1\\2\end{bmatrix}$$

$$\Rightarrow (2x_1^2 + x_1)\omega^{2x_0} \equiv z_1^2 + \omega^2 z_0 z_1^2 + \omega z_0^2 z_1^2$$

Determining ANF will be useful to generate unique arrays and sequences as also discussed for dimension 2. Here we also need to take three principles into account as following:

Principle A: Here we consider array u=XFF as an example and its function as $f=x_0x_1+x_0$ and another array u=FXF, that is generated by $f'=x_0x_1+x_1$. Then let a permutation like $x_0=x_1$ for f' where the new generated function is the same as *f*. So, to guarantee unique generations, only one of *f* or f' should be created. Consequently we must be cautious about the position of X and also W, X^2, WX and WX^2 in each combination to have all arrays unique.

Principle B: Let consider FD=Q and FD^2=R, the array u=FIQ and also u=IQF as graphically shown in Figure 2-2e and Figure 2-2f respectively. It is seen that IQF is a symmetrical reflection of FQI. Then these two arrays are considered as one and will be projected down to the same set of sequences. This phenomenon does not occur when u_n=I or in situations where I is the last object on the right hand side of the string.

When reversing the strings we have to face two different situations, a) when u∈ {F,Q,R} and b) when u∈ {I,F,Q,R}.
In situation a, we simply reverse the string symmetrically e.g. FQ will be reversed from QF. See Figure 2-2a and Figure 2-2b.
In situation b we consider I and the first adjacent substring to its right hand side as an irreversible unit. Here the whole string will be reversed except for the irreversible unit that keeps its configuration, e.g. FIQ will be reversed as IQF as illustrated in Figure 2-2e and Figure 2-2f. Now, a special case of situation b is when we have some neighboring *I*'s. In this case the irreversible unit will continue to be on the right side as long as it reaches to a non-*I* substring, so e.g. FIIQ will be reversed to IIQF. See Figure 2-2c and Figure 2-2d.

Principle C: If a string is symmetric it means that its reversal is equal to itself and no repeating combinations will be observed in this case, such as IFIF that is presented in Figure 2-2g. We, however, need to control whether the conditions for principle A apply.

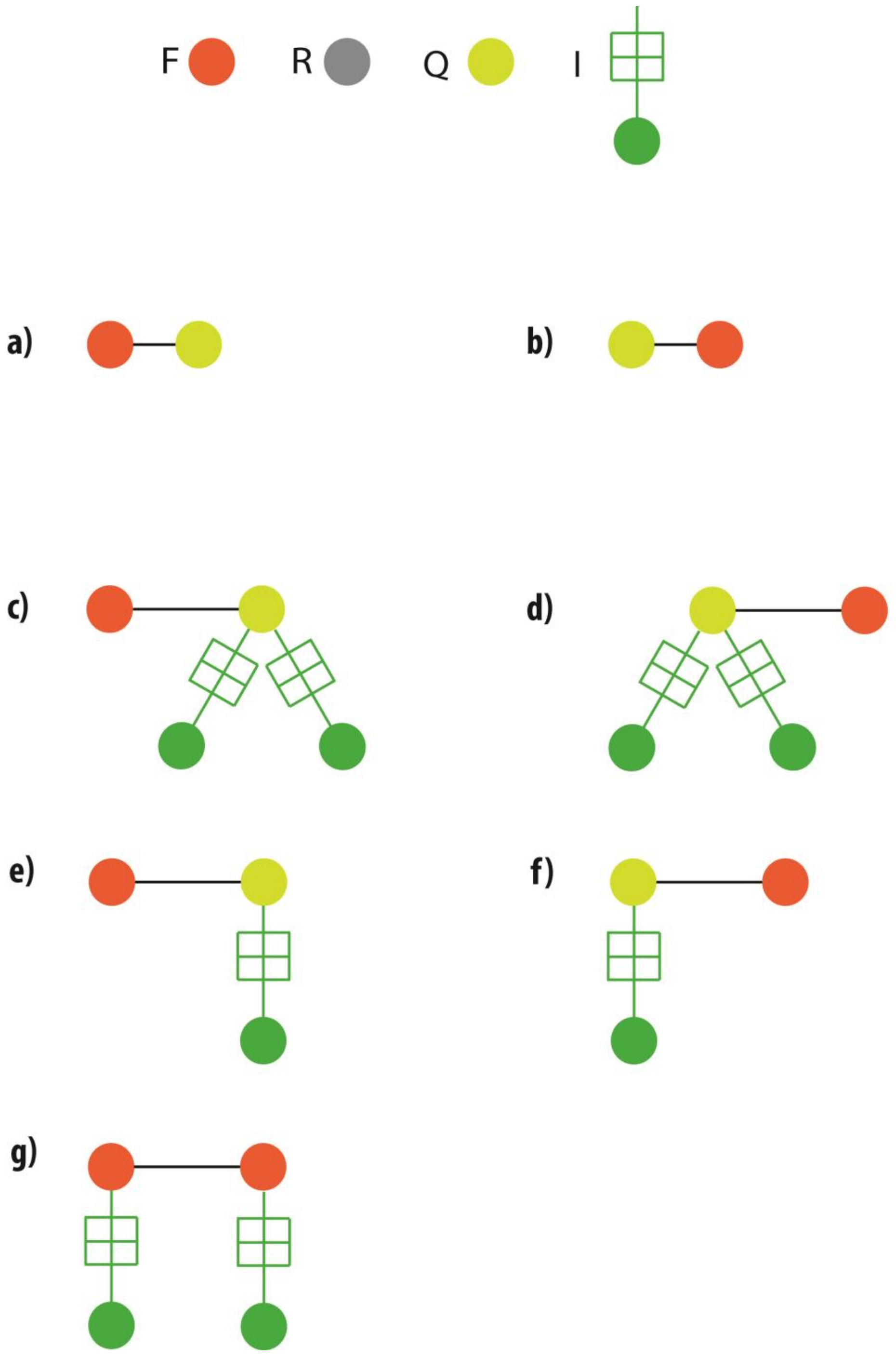


FIGURE 2-2: Schematic of some strings in dimension 3

2.3 Hardware and programming language

To perform this project, a personal MacBook Air with a 1.8-GHz Intel Core i5 processor was used as the hardware for the programming parts. MATLAB as a multi-paradigm numerical computing platform developed by MathWorks was also used as the programming language with the academic license provided by UiB. MATLAB is a robust tool to work with matrices and was a convenient option for this project where we mainly dealt with matrix manipulations. The code developed in this project can be interfaced with other languages such as C, C++, Java, Fortran and Python and also can be converted to C, C++ using a specific compiler.
The main strong point of the use of MATLAB for this thesis was the fact that its basic data element is the matrix itself. This made it straightforward to perform several mathematical operations on the construction of arrays and sequences required in the project. The relatively sophisticated mathematical content of the thesis also made it attractive to utilize an interactive computer language.

Although there were strong points associated with use of MATLAB in this thesis it also had some shortcomings including the large amount of memory required to run the codes. For high iterations and very complicated operations that involved matrices with more than 10000000 rows, the personal computer in use was practically incapable of running the code. The issue of low amount of memory compared to available capacity on normal computers however is the case for almost every language programming running such code with huge number of arrays and there is a need for probably super computers or a cluster of computers to be completely able to run for all iterations used in this project. At any rate, we have tried to ensure the quality of the code and the accuracy of the output. For the iterations that we managed to obtain output for, it has been proven that the accuracy and correctness of the generated software are maintained.

Chapter 3

Results and Discussion

The output of the generated code for the distinct MUB-based arrays and sequences for both dimension 2 and 3 is presented in this chapter. Relevant discussions and observations are also mentioned together with the results.

3.1 Introduction

The idea is to prepare a software package for generating complementary sets of arrays by using mutually unbiased bases (MUB) that satisfy all the following criteria:

1. Each sequence has near-flat Fourier spectra.
2. The pairwise distance between any two sequences (e.g. as measured by the inner-product) is as small as possible.
3. The set of sequences is as large as possible, given the near-flat constraint.

More accurately it is aimed to write a set of codes to generate arrays and sequences by using one, two or three MU bases in dimension 2 at first and then by using one, two, three or four MU bases in dimension 3. It is then targeted to calculate the total number of distinct arrays and unique sequences obtained in the two dimensions.

For generating arrays and sequences in dimension 2 the construction proposed by Gaofei Wu and Matthew G. Parker [7] was used. In addition to the criteria for adding the sequences and arrays mentioned in this reference, we also added some criteria for calculating the number of distinct arrays and unique sequences. As a result of these modifications, the outcomes of the current calculations are slightly different from those of reference [7] and this will be further discussed in this chapter.

With regard to the size of the array and sequence sets of dimension 3, no results existed before this thesis as complementary triples are a novel topic in MUB studies. The approach for dimension 3 then in this thesis was to generate distinct arrays and sequences first manually for a few iterations and then use the prepared software for the higher iterations.

3.2 Seeding with M_2 to generate complementary pairs

To generate complementary pairs, we introduce three sets {I}, {I, H}, {I, H, N} as one, two and three MU bases used in this study.

The matrices used for dimension 2 are:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3-1}$$

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3-2}$$

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3-3}$$

$$N = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & i \\ 1 & -i \end{pmatrix} \tag{3-4}$$

$$R_i = \begin{pmatrix} 1 & 0 \\ 0 & z_i \end{pmatrix} \tag{3-5}$$

where i=√-1 and I, H, N are unitary matrices as already described in the previous chapter.

We can construct multivariate polynomials in order to generate the required sequences and arrays. The structures for dimension 2 will be formed as follows:

$$1+z_0+z_1+z_0z_1+z_2+z_0z_2+z_1z_2+z_0z_1z_2+,... \quad (3\text{-}6)$$

It was decided to take the following six steps in order to generating complementary pairs (arrays and sequences) by using 2 x 2 unitary matrices:

1. Multivariate (array) construction using {H}
2. Univariate (sequence) construction using {H}
3. Multivariate (array) construction using {H, N}
4. Univariate (sequence) construction using {H, N}
5. Multivariate (array) construction using {H, I, N}
6. Univariate (sequence) construction using {H, I, N}

The software was generated to cover step 1 and 2 and also step 5 and 6 which are considered as the main objectives of the software. The following will be the explanation on how the software has been prepared and the relevant output achieved.

3.2.1 Multivariate construction using {H}

Constructions of arrays were started either with the two polynomials 1 and 1, or with the two polynomials 1 and -1. These polynomials were considered as column vectors $\begin{pmatrix}1\\1\end{pmatrix}$ or $\begin{pmatrix}1\\-1\end{pmatrix}$.

At the i'th step each vector was multiplied by matrices of the form PUR_i, where P= {I, X} and for this part U={H}. The global constant $1/\sqrt{2}$ from H was neglected as it does not impact on the final outcome of the software and at the same time it made the generation of arrays and sequences in each iteration more straightforward, so:

$$H = \begin{pmatrix}1 & 1\\1 & -1\end{pmatrix} \quad (3\text{-}7)$$

Then, after the i= 0'th iteration, the following four vectors will be obtained:

$$\begin{pmatrix}1+z_0\\1-z_0\end{pmatrix} = HR_0\begin{pmatrix}1\\1\end{pmatrix}$$

$$\begin{pmatrix}1-z_0\\1+z_0\end{pmatrix} = XHR_0\begin{pmatrix}1\\1\end{pmatrix}$$

$$\begin{pmatrix}1-z_0\\1+z_0\end{pmatrix} = HR_0\begin{pmatrix}1\\-1\end{pmatrix} \quad (3\text{-}8)$$

$$\begin{pmatrix}1+z_0\\1-z_0\end{pmatrix} = XHR_0\begin{pmatrix}1\\-1\end{pmatrix}$$

Obviously there is a repetition here, so after the i=0'th iteration we obtain two distinct vectors, where vector elements are univariate polynomials in z_0. Additionally there are only two distinct elements (polynomials) in these vectors, i.e. $1+z_0$ and $1-z_0$. As discussed in the previous chapter the coefficients of these distinct elements in turn generate arrays so we will have two arrays after the i=0'th iteration.

After the i=1st iteration we obtained the following four vectors:

$$\begin{gathered}\begin{pmatrix}1+z_0+z_1-z_0z_1\\1+z_0-z_1+z_0z_1\end{pmatrix} = HR_1\begin{pmatrix}1+z_0\\1-z_0\end{pmatrix}\\ \begin{pmatrix}1+z_0-z_1+z_0z_1\\1+z_0+z_1-z_0z_1\end{pmatrix} = XHR_1\begin{pmatrix}1+z_0\\1-z_0\end{pmatrix}\\ \begin{pmatrix}1-z_0+z_1+z_0z_1\\1-z_0-z_1-z_0z_1\end{pmatrix} = HR_1\begin{pmatrix}1-z_0\\1+z_0\end{pmatrix}\\ \begin{pmatrix}1-z_0-z_1-z_0z_1\\1-z_0+z_1+z_0z_1\end{pmatrix} = XHR_1\begin{pmatrix}1-z_0\\1+z_0\end{pmatrix}\end{gathered} \tag{3-9}$$

So after the i=1st iteration we obtained four distinct vectors where vector entries are bivariate polynomials in z_0 and z_1. Also there are only four distinct elements in these vectors namely $1+z_0+z_1-z_0z_1$, $1+z_0-z_1+z_0z_1$, $1-z_0+z_1+z_0z_1$ and $1-z_0-z_1-z_0z_1$.

After the i=2nd iteration we obtained the following eight vectors:

$$\begin{pmatrix}1+z_0+z_1-z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2\\1+z_0+z_1-z_0z_1-z_2-z_0z_2+z_1z_2-z_0z_1z_2\end{pmatrix}=HR_2\begin{pmatrix}1+z_0+z_1-z_0z_1\\1+z_0-z_1+z_0z_1\end{pmatrix}$$

$$\begin{pmatrix}1+z_0+z_1-z_0z_1-z_2-z_0z_2+z_1z_2-z_0z_1z_2\\1+z_0+z_1-z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2\end{pmatrix}=XHR_2\begin{pmatrix}1+z_0+z_1-z_0z_1\\1+z_0-z_1+z_0z_1\end{pmatrix}$$

$$\begin{pmatrix}1+z_0-z_1+z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2\\1+z_0-z_1+z_0z_1-z_2-z_0z_2+z_1z_2-z_0z_1z_2\end{pmatrix}=HR_2\begin{pmatrix}1+z_0-z_1+z_0z_1\\1+z_0+z_1-z_0z_1\end{pmatrix}$$

$$\begin{pmatrix}1+z_0-z_1+z_0z_1-z_2-z_0z_2+z_1z_2-z_0z_1z_2\\1+z_0-z_1+z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2\end{pmatrix}=XHR_2\begin{pmatrix}1+z_0-z_1+z_0z_1\\1+z_0-z_1+z_0z_1\end{pmatrix} \quad (3\text{-}10)$$

$$\begin{pmatrix}1-z_0+z_1+z_0z_1-z_2-z_0z_2-z_1z_2-z_0z_1z_2\\1-z_0+z_1+z_0z_1-z_2+z_0z_2+z_1z_2+z_0z_1z_2\end{pmatrix}=HR_2\begin{pmatrix}1-z_0+z_1+z_0z_1\\1-z_0-z_1-z_0z_1\end{pmatrix}$$

$$\begin{pmatrix}1-z_0+z_1+z_0z_1-z_2+z_0z_2+z_1z_2+z_0z_1z_2\\1-z_0+z_1+z_0z_1-z_2-z_0z_2-z_1z_2-z_0z_1z_2\end{pmatrix}=XHR_2\begin{pmatrix}1-z_0+z_1+z_0z_1\\1-z_0-z_1-z_0z_1\end{pmatrix}$$

$$\begin{pmatrix}1-z_0-z_1-z_0z_1+z_2-z_0z_2+z_1z_2+z_0z_1z_2\\1-z_0-z_1-z_0z_1-z_2+z_0z_2-z_1z_2-z_0z_1z_2\end{pmatrix}=HR_2\begin{pmatrix}1-z_0-z_1-z_0z_1\\1-z_0+z_1+z_0z_1\end{pmatrix}$$

$$\begin{pmatrix}1-z_0-z_1-z_0z_1-z_2+z_0z_2-z_1z_2-z_0z_1z_2\\1-z_0-z_1-z_0z_1+z_2-z_0z_2+z_1z_2+z_0z_1z_2\end{pmatrix}=XHR_2\begin{pmatrix}1-z_0-z_1-z_0z_1\\1-z_0+z_1+z_0z_1\end{pmatrix}$$

Here the vector entries are trivariate polynomials in z_0, z_1, and z_2. Furthermore there are only eight distinct elements in these vectors.

Now looking at the trend for *i*=1, 2 and 3 above, it can be said that after the i'th iteration, we will obtain 2^{i+1} distinct vectors and 2^{i+1} distinct polynomials in *i* variables, z_0, z_1, . . . z_{i-1}. The relevant number of distinct arrays is then obtained by:

$$|B_n| = 2^{i+1} \quad (3\text{-}11)$$

The number of obtainable distinct arrays from the software is called H_arrays in this thesis. The output of the code is the same as that of the formula above presented in Table 3-1 for n=1 to 5.

TABLE 3-1: The number of unique H arrays where p={I, X} and n=i+1.

Parameter	Value					
n	1	2	3	4	5	6
$\lvert B_n \rvert$ (#unique H array)	2	4	8	16	32	64

The software in this project deals with large numbers and it is vital to decrease the time consumed to run the code by any possible means. One of the approaches to increase the speed of the software was to define a type of multiplication of matrices where the summation of operators is omitted for example:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ z_0 z_1 \end{pmatrix} \tag{3-12}$$

We also figured out that we do not need to calculate the matrices that are multiplied by $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ since the results of these matrices are swapped from the product of the matrices and $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

Table 3-2 shows the results of the H_arrays function for the iterations from 0 to 2 where the distinct arrays and the number of them are displayed.

Table 3-2: The output of the code H_arrays for i=0-2.

Output of H_arrays

```
>> H_arrays
Enter your iteration: 2
Iteration = 0
Number of distinct arrays = 2
     1    1
     1   -1
Iteration = 1
Number of distinct arrays = 4
     1    1    1   -1
     1    1   -1    1

     1   -1    1    1
     1   -1   -1   -1
Iteration = 2
Number of distinct arrays = 8
     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
```

3.2.2 Univariate construction using {H}

In this section complementary sequences of length 2^n were constructed. These sequences are obtained by projection of the arrays already produced in the last section using the formula $z_i = z^{2^{\pi(i)}}, \pi \in S_n$, from the n-dimensional arrays down to 1 dimensional sequences of length 2^n.

There are n! possible projections here, but not all these projections are unique. We need to elaborate this with an example. The multivariate polynomial $1+z_0+z_1-z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2$, one of the eight distinct arrays for the i=2 iteration, is considered as the example here.

We can project this multivariate polynomial down to a univariate polynomial with coefficients from the alphabet {1, -1} by projecting z_0, z_1, and z_2 to suitable powers of z.

The coefficients of this univariate polynomial will describe a sequence of length $2^3 = 8$ in the alphabet {1, -1} i.e. (1 1 1 -1 1 1 -1 1).
There are 3!=6 possible projections for each array but only 3!/2=3 of them are in the set of distinct sequences. The following is the 3!/2=3 projections that were chosen:

$$\begin{aligned} z_0 &= z, z_1 = z^2, z_2 = z^4 \Longrightarrow 1 + z + z^2 - z^3 + z^4 + z^5 - z^6 + z^7 \\ z_0 &= z^2, z_1 = z, z_2 = z^4 \Longrightarrow 1 + z + z^2 - z^3 + z^4 - z^5 + z^6 + z^7 \\ z_0 &= z^2, z_1 = z^4, z_2 = z \Longrightarrow 1 + z + z^2 + z^3 + z^4 - z^5 - z^6 + z^7 \end{aligned} \tag{3-13}$$

The rest of the possible projections provide the same univariate polynomials as above. Then the number of distinct univariate polynomials (sequences) generated by the arrays is given by:

$$E_H(n)= 2^n \frac{n!}{2} \tag{3-14}$$

where n=i+1.

In this thesis the number of obtainable distinct sequences from the software is named H_sequences. The output of the code is the same as that of the formula above presented in Table 3-3 for n=1 to 5.

TABLE 3-3: The number of distinct H sequences for p={I, X} and n=i+1.

Parameter	Value				
n	1	2	3	4	5
$E_H(n)$ (#distinct H sequences)	1	4	24	192	1920

The code finds coefficients of multivariate polynomials, which are called arrays, and also finds distinct sequences by determining the negative elements in multivariate polynomials and then finding the indices of them in all possible permutation of z_0, z_1, z_2 and higher indices. Then the value of these indices for elements in any permutation is assigned to -1. Now the rest of the sequence elements in each permutation are assigned to 1. This way we get n! possible sequences for each array, but n!/2 are distinct as already discussed. According to *(3-14)* we will then have $2^n(n!/2)$ distinct sequences in each iteration.
For example for i=2, we have z_0, z_1 and z_2 in multivariate polynomials so 3! permutations for these variables meaning that we will have six different combinations as listed below:

$$\begin{aligned}
(z_0, z_1, z_2) &\rightarrow 1, z_0, z_1, z_0z_1, z_2, z_0z_2, z_1z_2, z_0z_1z_2 \\
(z_1, z_0, z_2) &\rightarrow 1, z_1, z_0, z_0z_1, z_2, z_0z_2, z_1z_2, z_0z_1z_2 \\
(z_2, z_0, z_1) &\rightarrow 1, z_2, z_0, z_0z_2, z_1, z_1z_2, z_0z_1, z_0z_1z_2 \\
(z_0, z_2, z_1) &\rightarrow 1, z_0, z_2, z_0z_2, z_1, z_0z_1, z_1z_2, z_0z_1z_2 \\
(z_2, z_1, z_0) &\rightarrow 1, z_2, z_1, z_1z_2, z_0, z_0z_2, z_0z_1, z_0z_1z_2 \\
(z_1, z_2, z_0) &\rightarrow 1, z_1, z_2, z_1z_2, z_0, z_0z_1, z_0z_2, z_0z_1z_2
\end{aligned} \qquad \textit{(3-15)}$$

Now for the multivariate polynomial $1+z_0+z_1-z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2$ we see that z_0z_1 and z_1z_2 have negative coefficients as marked in red in *(3-15)*. The rest of the terms have positive coefficients so the six combinations according to the length of the sequence in the *i'th* iteration are listed in Table 3-4:

Table 3-4: All projections for *$1+z_0+z_1-z_0z_1+z_2+z_0z_2-z_1z_2+z_0z_1z_2$*

Sequences
(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)

As can be seen, three of the sequences above are alike so we will have 3!/2=3 distinct univariate polynomials as following:

$$\begin{aligned}
&1+z+z^2-z^3+z^4+z^5-z^6+z^7 \\
&1+z+z^2+z^3+z^4-z^5-z^6+z^7 \\
&1+z+z^2-z^3+z^4-z^5+z^6+z^7
\end{aligned} \qquad \textit{(3-16)}$$

To find the negative elements in the multivariate the polynomials we created a set of elements (1, z_0, z_1, z_0z_1, z_2, etc.) with a new subfunction called [d5,m]=permutation_ di2(iter).

In the initial steps of this master thesis we have utilized symbolic variables in Matlab software for generating z_0, z_1, z_2 etc. but the code would take too much time for processing. Therefore to improve the speed of the software, the new subfunction was formed with numeric values. The results showed that the running time for the code was significantly reduced by using numeric values. In this subfunction *iter* is *i* and *m* is the prime numbers that we consider as z_0, z_1, ... and the numbers are assigned to the values in order, e.g. z_0=2, z_1=3, z_2=5, z_3=7, z_4=11, z_5=13, z_6=17 and so on. The reason to use prime numbers is to make a unique set of variables. If the numbers for z_0, z_1, ... z_n are not prime numbers, the numbers that are produced by the products/combinations of these variables can be repetitions of the variables themselves. For instance, imagine that z_5 is not 13 or any other prime number and instead, it is a non-prime number such as 14. Then having z_0z_3 (=14), it has already appeared in the set and we then face to a repetition. d5 is all possible permutations of the variables. The subfunction has also been used in our main code for generating sequences.

It has been attempted to create sequences and arrays in the same code that in turn increased the speed compared to generating the sequences and arrays separately in two different codes. The function H_sequences can then provide both distinct arrays and sequences at the same time.

Table 3-5 shows the results of the H_sequences function for the iterations from 0 to 2 where the distinct sequences and the number of them are displayed.

Table 3-5: The output of the code H_sequences for i=0-2.

Output of H_sequences

```
>> H_sequences
Enter your iteration: 2
Iteration = 0
Number of distinct arrays = 2
Number of unique sequences = 2
     1   -1
     1    1
Iteration = 1
Number of distinct arrays = 4
Number of unique sequences = 4
     1   -1   -1   -1
     1   -1    1    1
     1    1   -1    1
     1    1    1   -1
Iteration = 2
Number of distinct arrays = 8
Number of unique sequences = 24
     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
     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
     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
```

3.2.3 Multivariate construction using {H, I, N}

In this section two codes were separately created for two special cases; P= {I, X}, U={H, I, N} and P={I}, U={H, I, N} for the situations with and without linear offset respectively.

The following are the principles that were considered for finding distinct arrays. For the P={I,X} principles A, B and C are applicable while principles B and C are applicable for P={I} since there is no X available in this case.

Principle A: To start explaining this principle we consider an example of an array u=IXIH with Boolean function $f=(x_0+x_2+1)(x_1+x_2)i^0$ and another array u=XIIH, where $f^{'}=(x_0+x_2)(x_1+x_2+1)i^0$. The arrays that are generated by the recursive function presented in the previous chapter are not unique. To elaborate this we can consider that x_0 is swapped with x_1 in $f^{'}$ where the new generated function is the same as *f*. So, to guarantee unique generations, only one of *f* or $f^{'}$ should be created. The same goes for XIIXH and IXIXH. So, for the combination IIH we will have six distinct arrays instead of eight arrays. Consequently we must be cautious about the different positions of X in each combination to have a proper picture of unique arrays. We also considered two arrays equal if the only difference in their functions are to be multiplied by global constants and everything else is the same between them. For example, if we consider u=NXI, the matrix generated by u is $\begin{bmatrix} z_0 + iz_1 \\ z_0 - iz_1 \end{bmatrix}$. If we swap z_0 and z_1 in the first row we will have z_1+iz_0 which is the same as the second row if multiplied by –i (global constant) meaning that we can consider these two arrays as one.

Principle B: Let us consider the arrays u=IHN and u=NIH drawn in a graphical language as shown in Figure 3-1 parts a and b respectively. It can be observed that the figure of NIH (b) is a symmetrical reflection of IHN (a). Then these two arrays are considered as one and will be projected down to the same set of sequences. This phenomenon does not occur when u_n=I or in situations where I is the last object on the right hand side of the strings. For reversing the strings we have two different situations, a) when u∈ {H,N} and b) when u∈ {I,H,N}. In situation a, we simply reverse the string symmetrically e.g. NHNH will be reversed as HNHN. See Figure 3-1c and Figure 3-1c.

On the other hand in situation b we consider *I* and the first adjacent substring to its right hand side as an irreversible unit. Here the whole string will be reversed but not the irreversible unit that keeps its configuration, e.g. HIHN will be reversed as NIHH (see Figure 3-1 parts e and f). Now, a special case of situation b is when we have some neighboring I's. In this case the irreversible unit will be continued to the right side as long as it reaches to a non-I substring, so e.g. NIIH will be reversed to IIHN and IIINH will be reversed to HIIIN (see Figure 3-1g and Figure 3-1h and also Figure 3-1i and Figure 3-1j).

Principle C: If a string is symmetric it means that its reversal is equal to itself and no repeating combinations will be observed in this case, such as IHIH. We, however, need to control whether the conditions for principle A apply.

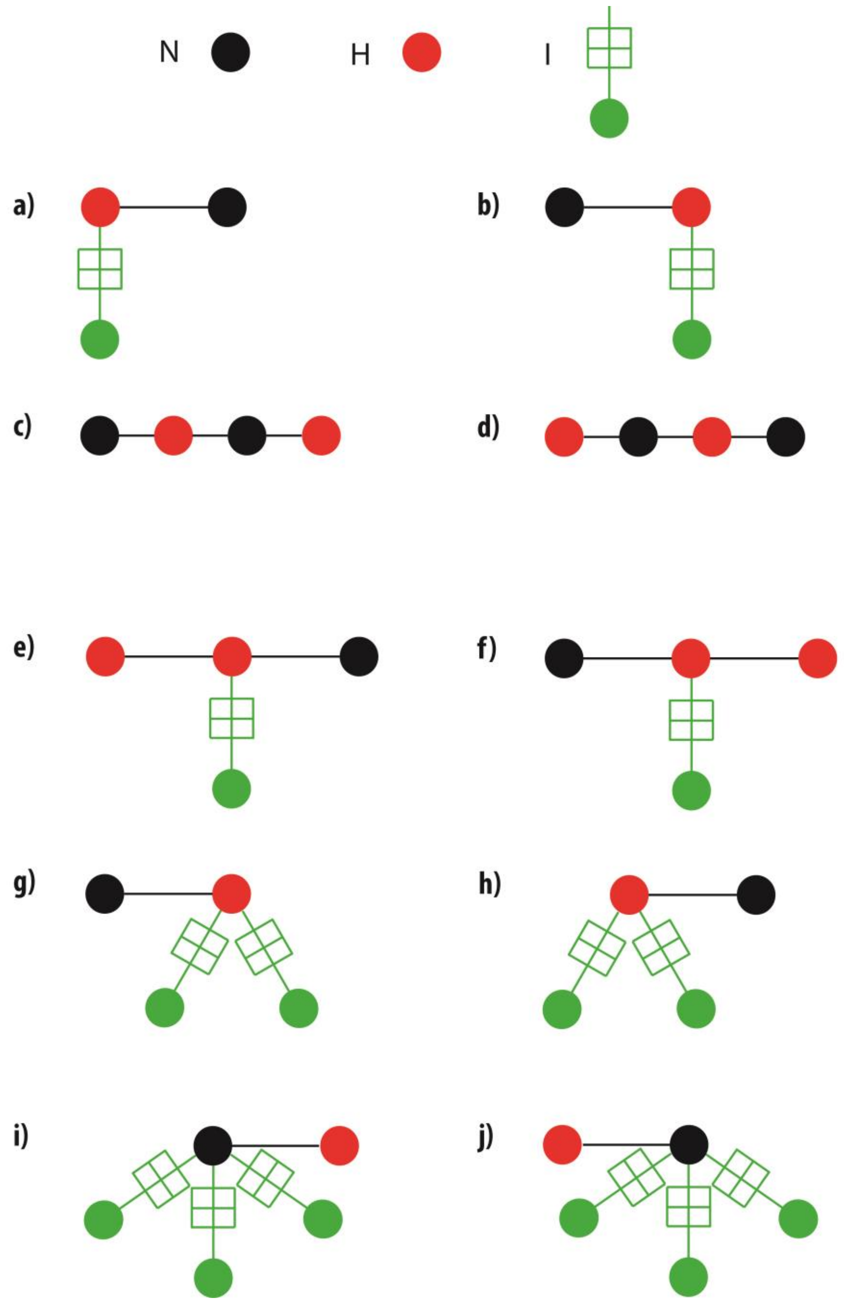


Figure 3-1: Symbolic representation of the IHN combinations

We wrote two functions individually and used them in two main codes for generating arrays for the two special cases of P= {I, X} and P={I}. The first function is w1=used_matrices(n) where n=i+1 and w1, the output of the function, are matrices (combinations) that are required to generate distinct arrays. When preparing the functions, principles A to C should apply. For example when i=1 w1 is II, IH, IN, HI, HH, HN, NI, 00 and NN. As we see the value of NH is equal to zero since NH is symmetric with HN and according to principle B we must ignore one of them.

The second function is [d5,m]=permutation_di2(iter) where *iter=i* and *m* is the prime numbers that we considered as z_0, z_1,...z_i. d5 is defined as a cell array that covers all possible permutations of the set of these numbers as 1, z_0, z_1, z_0z_1 and etc.
For example when *iter*=2 then m will be {2} for z_0, {3} for z_1 and d5{1}={1,2,3,6} as 1, z_0, z_1, z_0z_1 and d5{2}={1,3,2,6} as 1, z_1, z_0, z_0z_1.

We constructed a set of complementary array pairs over the alphabet {0, 1, i, -1, -i}, i=√-1. For instance where P={I, X}, after the i=0'th iteration we obtain the following eight vectors:

$$
\begin{aligned}
&IR_0\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}1\\z_0\end{pmatrix} && IIR_0\begin{pmatrix}1\\-1\end{pmatrix}=\begin{pmatrix}1\\-z_0\end{pmatrix}\\
&IHR_0\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}1+z_0\\1-z_0\end{pmatrix} && IHR_0\begin{pmatrix}1\\-1\end{pmatrix}=\begin{pmatrix}1-z_0\\1+z_0\end{pmatrix}\\
&INR_0\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}1+iz_0\\1-iz_0\end{pmatrix} && INR_0\begin{pmatrix}1\\-1\end{pmatrix}=\begin{pmatrix}1-iz_0\\1+iz_0\end{pmatrix}\\
&XIR_0\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}z_0\\1\end{pmatrix} && XIR_0\begin{pmatrix}1\\-1\end{pmatrix}=\begin{pmatrix}-z_0\\1\end{pmatrix}\\
&XHR_0\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}1-z_0\\1+z_0\end{pmatrix} && XHR_0\begin{pmatrix}1\\-1\end{pmatrix}=\begin{pmatrix}1+z_0\\1-z_0\end{pmatrix}\\
&XNR_0\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}1-iz_0\\1+iz_0\end{pmatrix} && XNR_0\begin{pmatrix}1\\-1\end{pmatrix}=\begin{pmatrix}1+iz_0\\1-iz_0\end{pmatrix}
\end{aligned}
\qquad (3\text{-}17)
$$

The elements of the vectors are univariate polynomials containing z_0. Additionally there are only six distinct elements (polynomials) in these vectors, namely 1, z_0, $1+z_0$, $1-z_0$, $1+iz_0$, $1-iz_0$. Here $-z_0$ is the same as z_0 since they differ only by a global constant.

For preparation of the code we did not consider vector $\begin{pmatrix}1\\-1\end{pmatrix}$ as its outcomes were the same for vector $\begin{pmatrix}1\\1\end{pmatrix}$. We also did not need to write codes for calculating X when it is placed in the first position at the left hand side in any combinations. The reason is that all the matrices can be formed by $F_j(z_j) = P_jU_jR_j(z_j)F_{j-1}(z_{j-1})$ (P_j=I, and P_{j-1}={I, X}) and if performing a swap of the rows of these matrices, Pj=X will also be calculated. For finding distinct arrays we only need the matrices that are generated by Pj=I.

For the generation of the arrays without linear offset we did not need to add p=X to our combinations. For instance after the *i*=1 iteration we obtain nine matrices with two rows and four columns where P=I and the matrices' entries are bivariate polynomials containing z_0, z_1. Then there are sixteen arrays where we also take principle A into consideration.

We created both sequences and arrays using unique software again similar to what has been done in the last section to increase efficiency and speed of the software. Using the function construct_arrays_sequences_without_linearoffset we can get both distinct arrays and sequences for P={I} i.e. without linear offset.
The Table 3-6 shows the number of distinct arrays for n=1 to n=5.

TABLE 3-6: The number of unique IHN arrays for p={I} and n=i+1.

Parameter	Value					
n	1	2	3	4	5	6
$\|B_n\|$ (#unique IHN array)	6	16	42	120	342	1008

For generating distinct arrays and sequences with linear offset i.e. P={I,X} we wrote the function construct_arrays_sequences_di2 and received back the number of distinct arrays and sequences. The results only for the distinct arrays are summarized in Table 3-7 .

TABLE 3-7: The number of unique IHN arrays for p={I,X} and n=i+1.

Parameter	Value				
n	1	2	3	4	5
$\|B_n\|$ (#unique IHN array)	6	28	130	677	3581

These results are slightly different from the reference [7] since the distinct arrays here are determined with linear offset (p={I,X}) and also take principle A to C into account.

It is also of interest to point out that principle A was a time consuming part of the code and to check its criteria added dramatically to the running time of the software. For example for u_1=(I, I, I), the three combinations with linear offset are u_2=(I, XI, I), u_3=(I, XI, XI) and u_4=(I, I, XI) and their generated arrays are:

$$
\begin{aligned}
&u_1 = (I, I, I) \text{ the arrays are} : \begin{bmatrix} 1 \\ z_0 z_1 z_2 \end{bmatrix} \\
&u_2 = (I, I, XI) \text{the arrays are}: \begin{bmatrix} z_0 \\ z_1 z_2 \end{bmatrix} \\
&u_3 = (I, XI, I) \text{ the arrays are}: \begin{bmatrix} z_0 z_1 \\ z_2 \end{bmatrix} \\
&u_4 = (I, XI, XI) \text{ the arrays are}: \begin{bmatrix} z_1 \\ z_0 z_2 \end{bmatrix}
\end{aligned}
\qquad (3\text{-}18)
$$

It can be seen that in u_2 when z_0 and z_1 are swapped, we would get the arrays of u_3. Likewise, when z_0 and z_2 are swapped in u_2, we would get the arrays of u_4. As a result, the four distinct arrays here are 1, $z_0z_1z_2$, z_0 and z_1z_2.

We have figured out that the first element of the arrays can have only two values, either 0 or 1. So we could limit our search for finding the arrays that only differ by a global constant over the arrays which have zero in the first column.

We made matrices using {I,H,N} and {I, X} so that they have the output in an ordered configuration so that we can search for and find the linear offsets of each combination and check principle A for them. Using this method the search operation was restricted and the software functioned much faster as a consequence.

Table 3-8 shows the results of the construct_arrays_sequences_without_linearoffset function for the iterations from 0 to 1.

Table 3-8: The output construct_arrays_sequences_without_linearoffset, i=0-1.

Output without linear offset
>> construct_arrays_sequences_without_linearoffset Enter your iteration: 1 Iteration = 0 Number of distinct arrays = 6 [1,   0] [0,   1] [1,   1] [1, -1] [1, 1i] [1, -1i] Iteration = 1 Number of distinct arrays = 16 [1, 0,   0,   0] [0, 0,   0,   1] [1, 1,   0,   0] [0, 0,   1, -1] [1, 1i,   0,   0] [0, 0,   1, -1i] [1, 0,   0,   1] [1, 0,   0, -1] [1, 1,   1, -1] [1, 1, -1,   1] [1, 1i,   1, -1i] [1, 1i, -1, 1i] [1, 0,   0, 1i] [1, 0,   0, -1i] [1, 1i, 1i,   1] [1, 1i, -1i, -1]

Table 3-9 shows the results of the construct_arrays_sequences_di2 function for the iterations from 0 to 1.

Table 3-9: The output of the code construct_arrays_sequences_di2 for i=0-1.

Output with linear offset

```
>> construct_arrays_sequences_di2
Enter your iteration: 1
Iteration = 0
Number of distinct arrays = 6
[1,   0]
[0,   1]
[1,   1]
[1,  -1]
[1,  1i]
[1, -1i]
Iteration = 1
Number of distinct arrays = 28
[ 1,   0,   0,   0]
[ 0,   0,   0,   1]
[ 1,   1,   0,   0]
[ 0,   0,   1,  -1]
[ 1,  1i,   0,   0]
[ 0,   0,   1, -1i]
[ 1,   0,   0,   1]
[ 1,   0,   0,  -1]
[ 1,   1,   1,  -1]
[ 1,   1,  -1,   1]
[ 1,  1i,   1, -1i]
[ 1,  1i,  -1,  1i]
[ 1,   0,   0,  1i]
[ 1,   0,   0, -1i]
[ 1,  1i,  1i,   1]
[ 1,  1i, -1i,  -1]
[ 0,   1,   0,   0]
[ 1,  -1,   0,   0]
[ 0,   0,   1,   1]
[ 1, -1i,   0,   0]
[ 0,   0,   1,  1i]
[ 0,   1,   1,   0]
[ 0,   1,  -1,   0]
[ 1,  -1,  -1,  -1]
[ 1, -1i,   1,  1i]
[ 1, -1i,  -1, -1i]
[ 0,   1,  1i,   0]
[ 1, -1i, -1i,   1]
```

3.2.4 Univariate construction using {H, I, N}

In this section two codes were separately created for P={I} and P={I, X}. The complementary sequences of length 2^n are constructed from the arrays that were created in the last section for P={I} and P={I, X} by the projections $z_i = z^{2^{\pi(i)}}, \pi \in S_n$ from the n-dimensional arrays down to 1 dimensional sequences of length 2^n. In addition to applying the three principles A, B and C discussed before, here we also need to check that all sequences in the complementary sets are unique.

There are n! possible projections for each array, but not all these projections are unique. For example for i=1 we have the string u=NXI in the set of combinations and it is equal to $\begin{bmatrix} 0 & 1 & \mathrm{i} & 0 \\ 0 & 1 & -\mathrm{i} & 0 \end{bmatrix}$. Each array has 2! possible projections so there are four sequences for this string: [0 ,1 ,i ,0], [0, 1, –i ,0], [0, i, 1, 0] and [0, -i, 1, 0]. The third sequence is the same as the second one after multiplying by the global constant *–i*. If the fourth sequence is multiplied by the global constant *i*, the result will be the same as the first sequence. In this case we can omit two of the four mentioned sequences and eventually there will be two unique sequences for this combination.

In our code for any iteration it will be checked that all sequences are distinct. It will be also checked that there are no sequences differing from each other only by a multiplicative global constant.

For generating unique sequences without linear offset i.e. P={I} we wrote the function construct_arrays_sequences_without_linearoffset and received back the number of distinct arrays and sequences. The results only for the distinct sequences are summarized in Table 3-10.

TABLE 3-10: The number of distinct IHN sequences for p={I} and n=i+1.

Parameter	Value					
n	1	2	3	4	5	6
$E_{IHN}(n)$ (#distinct IHN sequence)	6	24	150	1318	15466	225962

Furthermore we obtain unique sequences for P={I,X} by the function construct_arrays_sequences_di2. The number of unique sequences obtained from this function is shown in Table 3-11.

Table 3-11: The number of distinct IHN sequences for p={I,X} and n=i+1.

Parameter	Value				
n	1	2	3	4	5
$E_{IHN}(n)$ (#distinct IHN sequence)	6	44	504	8755	207170

Table 3-12 shows the results of the construct_arrays_sequences_without_linearoffset function for the iterations from 0 to 1.

Table 3-12: The output construct_arrays_sequences_without_linearoffset

Output without linear offset

```
>> construct_arrays_sequences_without_linearoffset
Enter your iteration: 1
Iteration = 0
Number of unique sequences = 6
[ 1,   0]
[ 0,   1]
[ 1,   1]
[ 1,  -1]
[ 1,  1i]
[ 1, -1i]
Iteration = 1
Number of unique sequences = 24
[ 1,  0,  0,   0]
[ 1,  0,  0,   1]
[ 1,  0,  0,  1i]
[ 1,  0,  1,   0]
[ 1,  0, 1i,   0]
[ 1,  1,  0,   0]
[ 1,  1,  1,  -1]
[ 1,  1, 1i, -1i]
[ 1, 1i,  0,   0]
[ 1, 1i,  1, -1i]
[ 1, 1i, 1i,   1]
[ 0,   0,   0,   1]
[ 0,   0,   1, -1i]
[ 0,   0,   1,  -1]
[ 0,   1,   0, -1i]
[ 0,   1,   0,  -1]
[ 1,   0,   0, -1i]
[ 1,   0,   0,  -1]
[ 1, -1i,  1i,  -1]
[ 1,   1,  -1,   1]
[ 1,  1i, -1i,  -1]
[ 1,  1i,  -1,  1i]
[ 1,  -1,   1,   1]
[ 1,  -1,  1i,  1i]
```

Likewise Table 3-13 shows the results of the construct_arrays_sequences_di2 function for the iterations from 0 to 1.

TABLE 3-13: The output of the code construct_arrays_sequences_di2 for i=0-1.

Output with linear offset

```
>> construct_arrays_sequences_di2
Enter your iteration: 1
Iteration = 0
Number of unique sequences = 6
[ 1,   0]
[ 0,   1]
[ 1,   1]
[ 1,  -1]
[ 1,  1i]
[ 1, -1i]
Iteration = 1
Number of unique sequences = 44
[  0,   0,   0,   1]
[  0,   0,   1,   0]
[  0,   0,   1, -1i]
[  0,   0,   1,   1]
[  0,   0,   1,  1i]
[  0,   0,   1,  -1]
[  0,   1,   0,   0]
[  0,   1,   0, -1i]
[  0,   1,   0,   1]
[  0,   1,   0,  1i]
[  0,   1,   0,  -1]
[  0,   1,   1,   0]
[  0,   1,  1i,   0]
[  0,   1,  -1,   0]
[  0,  1i,   1,   0]
[  1,   0,   0,   0]
[  1,   0,   0, -1i]
[  1,   0,   0,   1]
[  1,   0,   0,  1i]
[  1,   0,   0,  -1]
[  1,   0, -1i,   0]
[  1,   0,   1,   0]
[  1,   0,  1i,   0]
[  1,   0,  -1,   0]
[  1, -1i,   0,   0]
[  1, -1i, -1i,   1]
[  1, -1i,   1,  1i]
[  1, -1i,  1i,  -1]
[  1, -1i,  -1, -1i]
[  1,   1,   0,   0]
[  1,   1, -1i,  1i]
[  1,   1,   1,  -1]
[  1,   1,  1i, -1i]
[  1,   1,  -1,   1]
[  1,  1i,   0,   0]
[  1,  1i, -1i,  -1]
[  1,  1i,   1, -1i]
[  1,  1i,  1i,   1]
[  1,  1i,  -1,  1i]
[  1,  -1,   0,   0]
[  1,  -1, -1i, -1i]
[  1,  -1,   1,   1]
[  1,  -1,  1i,  1i]
[  1,  -1,  -1,  -1]
```

3.3 Seeding with M_3 to generate complementary triples

To begin with the generation of complementary triples, we introduce four sets {I}, {I, F}, {I, F, FD} and {I, F, FD, FD2} as one, two, three and four MU bases used in this study.

The matrices used for dimension 3 are:

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3-19}$$

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \omega & 0 \\ 0 & 0 & \omega \end{pmatrix} \tag{3-20}$$

$$F = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{pmatrix} \tag{3-21}$$

$$R_i = \begin{pmatrix} 1 & 0 & 0 \\ 0 & z_i & 0 \\ 0 & 0 & z_i^2 \end{pmatrix} \tag{3-22}$$

$$W = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \tag{3-23}$$

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{3-24}$$

where $\omega = e^{\frac{2\pi i}{3}}$ in.

There are 3!=6 possible permutations for the rows of a 3x3 matrix. Then $P_{jk} = \left\{W^j X^k \,|\, j\epsilon\{0,1\}, k\epsilon\,\{0,1,2\}\right\}$ generates all 6 permutations.

The procedure to generate arrays and sequences for dimension 3 is the same as dimension 2 but with different matrices.

We started with $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ as column vector. At the *i*'th step each vector was multiplied by matrices of the form PUR_i, where $P\epsilon P_{jk}$ and U={I, F, FD, FD^2}.

We constructed a set of complementary array and sequences over the alphabet $\{0, 1, \omega, \omega^2\}$.

Furthermore the structure of variables in multivariate polynomial for dimension 3 was considered as $1+Z_0+Z_0^2+Z_1+Z_0Z_1+Z_0^2Z_1+Z_1^2+Z_0Z_1^2+Z_0^2Z_1^2$ and so on.

Let us consider FD=Q and FD^2=R. After the i=0'th iteration we will obtain the following 24 vectors:

$$\begin{aligned}
\mathrm{IIR}_0\begin{pmatrix}1\\1\\1\end{pmatrix} &= \begin{pmatrix}1\\ z_0\\ z_0^2\end{pmatrix}\\
\mathrm{IFR}_0\begin{pmatrix}1\\1\\1\end{pmatrix} &= \begin{pmatrix}1+z_0+z_0^2\\ 1+\omega z_0+\omega^2 z_0^2\\ 1+\omega^2 z_0+\omega z_0^2\end{pmatrix}\\
\mathrm{IQR}_0\begin{pmatrix}1\\1\\1\end{pmatrix} &= \begin{pmatrix}1+\omega z_0+\omega z_0^2\\ 1+\omega^2 z_0+z_0^2\\ 1+z_0+\omega^2 z_0^2\end{pmatrix}\\
\mathrm{IRR}_0\begin{pmatrix}1\\1\\1\end{pmatrix} &= \begin{pmatrix}1+\omega^2 z_0+\omega^2 z_0^2\\ 1+z_0+\omega z_0^2\\ 1+\omega z_0+z_0^2\end{pmatrix}
\end{aligned} \tag{3-25}$$

The other 20 vectors are generated by placing W, X, WX, WX^2, X^2 instead of *I* in each of the above vectors. The value of these vectors is obtained by swapping the rows.

We consider matrix $\mathrm{WX} = \begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix}$ as an example. Here the vectors WXIR0, WXFR0, WXQR0, WXRR0 are generated by swapping rows 1 and 2.
In a similar manner to dimension 2, we consider two cases for generating arrays and sequences: with and without linear offset and the corresponding codes were created separately.

The generation of sequences and arrays without linear offset here were based on the principle B and C discussed in section 3.2.3. Also the generation of sequences and arrays with linear offset were based on principles A, B and C.

Two subfunctions were then individually created for generating arrays for the two main codes. The function w1=used_matrices_di3(n) provides some of the necessary matrices (combinations) for generating arrays after considering principles B and C. For example when *i*=1, w1 will be: II, IF, IQ, IR, FI, FF, FQ, FR, QI, 00, QQ, QR, RI, 00, 00, RR.

It can be observed that the value of QF equals to zero since QF is symmetric with FQ and according to principle B we need to count only one them and also RF and RQ are equal to zero.

The second function is [d5,m]=all_permutation_di3(iter) where *iter*=*i* and *m* is the prime numbers that we considered as z_0, z_1 and so on as discussed before. d5 is defined as a cell array which can be all possible permutations of the set of these numbers as 1, z_0, z_0^2, z_1 and more.

For example when *iter*=2 the *m* will be {2} for z_0 and {3} for z_1 and d5{1} will be {1,2,4,3,6,12,9,18,36} as 1, Z_0, Z_0^2, Z_1, Z_0Z_1, $Z_0^2Z_1$, Z_1^2, $Z_0Z_1^2$, $Z_0^2Z_1^2$ and d5{2} is {1,3,9,2,6,18,4,12,36 } as 1, Z_1, Z_1^2, Z_0, Z_1Z_0, $Z_1^2Z_0$, Z_0^2, $Z_1Z_0^2$, $Z_1^2Z_0^2$.

Sequences and arrays without linear offset were generated by the function construct_arrays_sequences_without_linearoffset_di3.

The Table 3-14 and Table 3-15 show the number of distinct arrays and sequences without linear offset for n=1 to 5.

TABLE 3-14: The number of unique IFQR arrays for p={I}

Parameter	Value				
n	1	2	3	4	5
$\|B_n\|$	11	39	139	517	1993

TABLE 3-15: The number of distinct IFQR sequences for p={I}

Parameter	Value				
n	1	2	3	4	5
$E_{IFQR}\ (n)$	11	64	633	8054	136305

For the generation of distinct arrays and sequences for P={I, W, X, WX, WX^2, X^2} the function construct_arrays_sequences_di3 has been used. The number of distinct arrays and sequences were obtained from this function as shown in Table 3-16 and
Table 3-17 for different values of *n*.

TABLE 3-16: The number of unique IFQR arrays for $P\epsilon P_{jk}$

Parameter	Value		
n	1	2	3
$\|B_n\|$	11	188	3725

TABLE 3-17: The number of unique IFQR sequences for $P \epsilon P_{jk}$

Parameter	Value		
n	1	2	3
$E_{IFQR}\ (n)$	11	350	20405

In dimension 2 we needed to detect the sequences that are similar except for being multiplied by a global constant. In dimension 3 however we figured out based on an inductive iteration that there is no need for such detection and this is an interesting observation for dimension 3.

Table 3-18 presents the resulting output without linear offset of the function construct_arrays_sequences_without_linearoffset_di3.

TABLE 3-18: The output of construct_arrays_sequences_without_linearoffset_di3.

Output without linear offset

```
>> construct_arrays_sequences_without_linearoffset_di3
Enter your iteration: 1
Iteration = 0
Number of distinct arrays = 11
Number of unique sequences = 11
   1              0               0
   0              1               0
   0              0               1
   1              1               1
   1        -0.5 + 0.87i    -0.5  - 0.87i
   1        -0.5  - 0.87i   -0.5 + 0.87i
   1        -0.5 + 0.87i    -0.5 + 0.87i
   1        -0.5  - 0.87i         1
   1              1         -0.5  - 0.87i
   1              1         -0.5 + 0.87i
   1        -0.5 + 0.87i          1
Iteration = 1
Number of distinct arrays = 39
Number of unique sequences = 64
```

Here the results for generated arrays and sequences for iteration=1 are not presented due to them being too lengthy.
Likewise Table 3-19 presents the output for the function construct_arrays_sequences_di3 for i=0 to 1.

Table 3-19: The output of the code construct_arrays_sequences_di3 for i=0-1.

Output with linear offset

```
>> construct_arrays_sequences_di3
Enter your iteration: 1
Iteration = 0
Number of distinct arrays = 11
Number of unique sequences = 11
  1              0                0
  0              1                0
  0              0                1
  1              1                1
  1        -0.5 + 0.87i     -0.5  - 0.87i
  1        -0.5  - 0.87i    -0.5 + 0.87i
  1        -0.5 + 0.87i     -0.5 + 0.87i
  1        -0.5  - 0.87i          1
  1              1          -0.5  - 0.87i
  1              1          -0.5 + 0.87i
  1        -0.5 + 0.87i           1
Iteration = 1
Number of distinct arrays = 188
Number of unique sequences = 350
```

Here also we did not present the arrays and sequences for iteration=1 due to being too lengthy.

Chapter 4

Conclusion and future work

In this chapter the conclusions of the work are summarized together with recommendations for future work.

4.1 Concluding remarks

This project has been in line with the research aiming to enhance the set size of the complementary sequences while the upper bound of PAPR keeps as low as reasonably achievable and also the pairwise distinguishability of the sequences is maintained. The approach used to perform the task was seeding the recursive construction with optimal mutually-unbiased bases. Such sequence construction is a mathematically complicated job and often needs computer aided solutions. To address this we generated program codes that constructed unique arrays and sequences for dimension 2 and 3 seeding by MUBs with and without linear offset.

The codes for both dimensions have delivered satisfactory results as far as the available computer resource can handle. The results for lower iterations have also perfectly matched with the manually calculated values based on theory.

In dimension 2 it was required to detect and remove the sequences that differ only by a global constant. In dimension 3 however we figured out that such problematic sequences are not produced.

It has been observed that the number of sequences increases almost exponentially with increasing iterations and the growth rate was significantly higher in dimension 3.

In general, generating software capable of handling very large numbers as we are facing in this project is resource demanding. We have taken different measures to make the code efficient and fast and the running time for the code has been significantly improved over the course of this thesis.

The code has been generated on a flexible platform with a customizable structure that can be straightforwardly converted to other programming languages and can also be used for other dimensions with slight modifications.

4.2 Recommendations for future work

OFDM with low PAPR is attractive for telecommunication purposes and we still need to understand the mathematical behavior of arrays and sequences better. The following list of investigations may be considered in this regard:

- ✓ To run the current code for high iterations or to produce sequences and arrays in higher dimensions there is a need for a super computer or clusters of computers in future projects.
- ✓ Although the present software can be tailored for higher dimensions, it is still of interest to specifically work on dimension 5 and higher.
- ✓ Finding a general formula for determining the number of arrays and sequences for dimension 2 and 3 precisely both with and without linear offset. This may be instrumental in order to make wide application of MUB-based sequences possible.
- ✓ The complementary sequences and associated MUBs from this project can be potentially used in encoding-decoding tasks such as quantum cryptography in future.

Bibliography

[1] *New Oxford American Dictionary.* Oxford University Press, 2015.

[2] W. Y. Zou, “Orthogonal frequency division multiplexing: a multi-carrier modulation scheme,” *IEEE Trans. Consum. Electron.*, vol. 41, no. 3, pp. 392–399, 1995.

[3] W. Y. Zou and Y. Wu, “COFDM: an overview,” *IEEE Trans. Broadcast.*, vol. 41, no. 1, pp. 1–8, 1995.

[4] T. Jiang and Y. Wu, “An overview: peak-to-average power ratio reduction techniques for OFDM signals,” *Broadcast. IEEE Trans.*, vol. 54, no. 2, pp. 257–268, 2008.

[5] J. A. Davis and J. Jedwab, “Peak-to-mean power control in OFDM, Golay complementary sequences and Reed-Muller codes,” *IEEE Int. Symp. Inf. Theory - Proc.*, vol. 45, no. 7, p. 190, 1998.

[6] T. Jiang and G. Zhu, “Complement block coding for reduction in peak-to-average power ratio of OFDM signals,” *Commun. Mag. IEEE*, vol. 43, no. 9, pp. 17–22, 2005.

[7] G. Wu and M. G. Parker, “A complementary construction using mutually unbiased bases,” in *Cryptography and Communications*, 2013, vol. 6, no. 1, pp. 3–25.

[8] K. G. Paterson, “Generalized Reed-Muller codes and power control in OFDM modulation,” *Inf. Theory, IEEE Trans.*, vol. 46, no. 1, pp. 104–120, 2000.

[9] K.-U. Schmidt, “On cosets of the generalized first-order reed-muller code with low PMEPR,” *IEEE Trans. Inf. Theory*, vol. 52, no. 7, pp. 3220–3232, 2006.

[10] K.-U. Schmidt, "Complementary Sets, Generalized Reed-Muller Codes, and Power Control for OFDM," *IEEE Trans. Inf. Theory*, vol. 53, no. 2, pp. 808–814, 2007.

[11] F. Fiedler, J. Jedwab, and M. G. Parker, "A multi-dimensional approach to the construction and enumeration of Golay complementary sequences," *J. Comb. Theory, Ser. A*, vol. 115, no. 5, pp. 753–776, 2007.

[12] J. Jedwab and M. G. Parker, "Golay complementary array pairs," *Des. Codes Cryptogr.*, vol. 44, no. 1–3, pp. 209–216, 2007.

[13] S. Matsufuji, R. Shigemitsu, Y. Tanada, and N. Kuroyanagi, "Construction of Complementary Arrays," in *IEEE Symposium on Trends in Communications*, 2004, pp. 78–81.

[14] M. G. Parker and C. Tellambura, "Golay-Davis-Jedwab Complementary Sequences and Rudin-Shapiro Constructions," *IEEE Transactions on Information Theory.* pp. 1–44, 2001.

[15] M. G. Parker and C. Tellambura, "A Construction for Binary Sequence Sets with Low," *IEEE International Symposium on Information Theory, Lausanne, Switzerland*, June 30- July 5, 2002, no. 7, p. 239.

[16] M. G. Parker and C. Riera, "Generalised Complementary Arrays," in *13th IMA International Conference on Cryptography and Coding*, 12-15 Dec. ,2011, Oxford, UK, Lecture Notes in Computer Science, LNCS, 2011.

[17] I. D. Ivonovic, "Geometrical description of quantal state determination," *J. Phys. A. Math. Gen.*, vol. 14, no. 12, pp. 3241–3245, 1999.

[18] M. B. Ruskai, "Some Connections between Frames, Mutually Unbiased Bases, and POVM's in Quantum Information Theory," *Acta Appl. Math.*, vol. 108, no. 3, pp. 709–719, 2009.

[19] J. Schwinger, "Unitary Operator Bases.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 46, no. 4, pp. 570–579, 1960.

[20] M. Viswanathan, *Simulation of Digital Communication Systems Using Matlab[eBook]-Second Edition.* 2013.

[21] S. Shepherd, J. Orriss, and S. Barton, “Asymptotic limits in peak envelope power reduction by redundant coding in orthogonal frequency-division multiplex modulation,” *IEEE Trans. Commun.*, vol. 46, no. 1, pp. 5–10, 1998.

[22] J. Tao, M. Guizani, C. Hsiao-Hwa, X. Weidong, and W. Yiyan, “Derivation of PAPR Distribution for OFDM Wireless Systems Based on Extreme Value Theory,” *Wirel. Commun. IEEE Trans.*, vol. 7, no. 4, pp. 1298–1305, 2008.

[23] A. E. Jones, T. A. Wilkinson, and S. K. Barton, “Block coding scheme for reduction of peak to mean envelope power ratio of multicarrier transmission schemes,” *Electron. Lett.*, vol. 30, no. 25, pp. 2098–2099, 1994.

[24] S. Ben Slimane, “Reducing the peak-to-average power ratio of OFDM signals through precoding,” *IEEE Trans. Veh. Technol.*, vol. 56, no. 2, pp. 686–695, 2007.

[25] R. O’Neill and L. B. Lopes, “Envelope variations and spectral splatter in clipped multicarrier signals,” *Pers. Indoor Mob. Radio Commun. 1995. PIMRC’95. “Wireless Merging onto Inf. Superhighway”., Sixth IEEE Int. Symp.*, vol. 1, pp. 71–75 vol.1, 1995.

[26] R. Guangliang, Z. Hui, and C. Yilin, “A complementary clipping transform technique for the reduction of peak-to-average power ratio of OFDM system,” *Consum. Electron. IEEE Trans.*, vol. 49, no. 4, pp. 922–926, 2003.

[27] H. Nikookar and K. S. Lidsheim, “Random phase updating algorithm for OFDM transmission with low PAPR,” *Broadcast. IEEE Trans.*, vol. 48, no. 2, pp. 123–128, 2002.

[28] S. Yoo, S. Yoon, S. Y. Kim, and I. Song, “A novel PAPR reduction scheme for OFDM systems: selective mapping of partial tones (SMOPT),” *IEEE Trans. Consum. Electron.*, vol. 52, no. 1, pp. 40–43, 2006.

[29] R. W. Bäuml, R. F. H. Fischer, and J. B. Huber, "Reducing the peak-to-average power ratio of multicarrier modulation by selected mapping," *Electron. Lett.*, vol. 32, no. 22, p. 2056, 1996.

[30] S. H. Müller and J. B. Huber, "OFDM with reduced peak-to-average power ratio by optimum combination of partial transmit sequences," *Electron. Lett.*, vol. 33, no. 5, p. 368, 1997.

[31] T. A. Wilkinson and A. E. Jones, "Minimisation of the Peak To Mean Transmission Schemes By Block Coding," *IEEE 45th Veh. Technol. Conf.*, no. 1, pp. 825–829, 1995.

[32] I. Bengtsson, "Three ways to look at mutually unbiased bases," in *AIP Conference Proceeding 889- Vaxjo Conference on Foundations of Probability and Physics*, 2007, no. October, pp. 1–18.

[33] W. K. Wootters, "A Wigner-Function Formulation of Quantum Mechanics," *Ann. Phys. (N. Y).*, vol. 176, pp. 1–21, 1987.

[34] N. J. Cerf, M. Bourennane, A. Karlsson, and N. Gisin, "Security of quantum key distribution using d-level systems," *Phys. Rev. Lett.*, vol. 88, no. 12, pp. 1–4, 2001.

[35] W. K. Wootters and B. D. Fields, "Optimal state-determination by mutually unbiased measurements," *Ann. Phys.*, vol. 191, pp. 363–381, 1989.

[36] T. Durt, B.-G. Englert, I. Bengtsson, and K. Życzkowski, "On mutually unbiased bases," *Int. J. Quantum Inf.*, vol. 8, pp. 535–640, 2010.

[37] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. Van den Nest, and H.-J. Briegel, "Entanglement in Graph States and its Applications," in *Quantum Computers, Algorithms and Chaos*, 2006, pp. 1–99.

[38] F. R. Kschischang, B. J. Frey, and H. a. Loeliger, "Factor graphs and the sum-

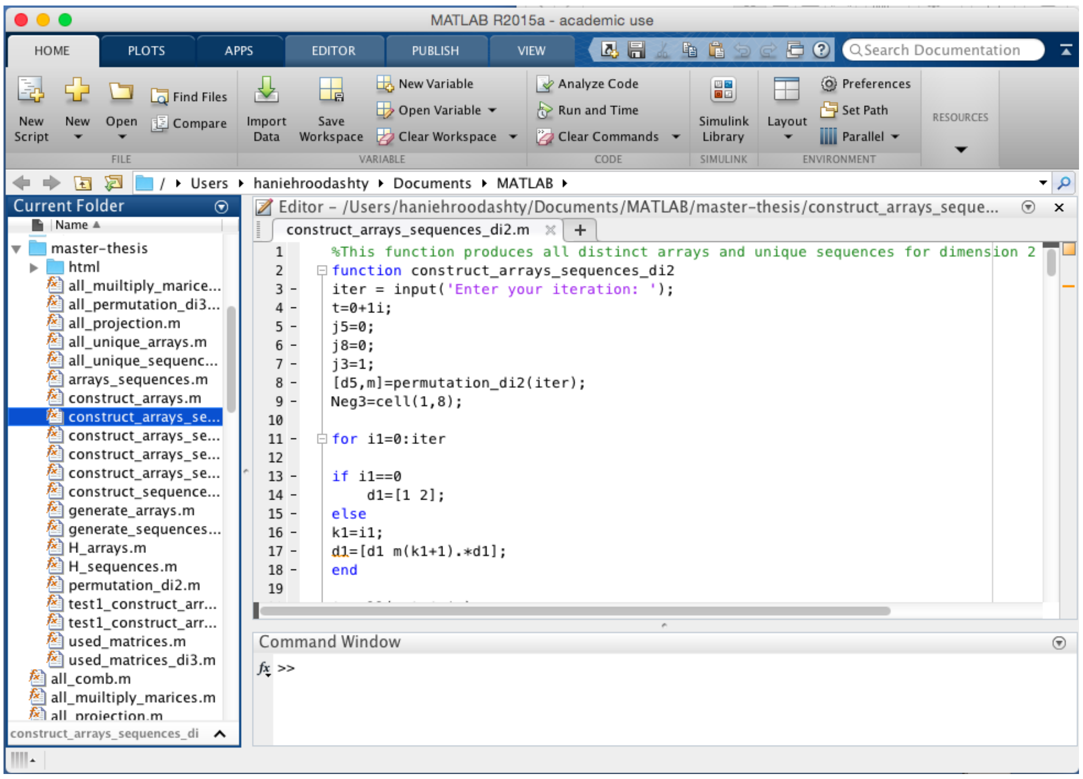
product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[39] C. Riera, S. Jacob, and M. G. Parker, "From Graph States to Two-Graph States," *Des. Codes Cryptogr.*, vol. 48, no. 2, pp. 179–206, 2008.

Appendix 1: User Manual

The goal here is to construct distinct arrays and sequences seeding by MUBs for dimension 2 and 3. The generated code in this project is in MATLAB language and one would need this software with a valid license to run the code.

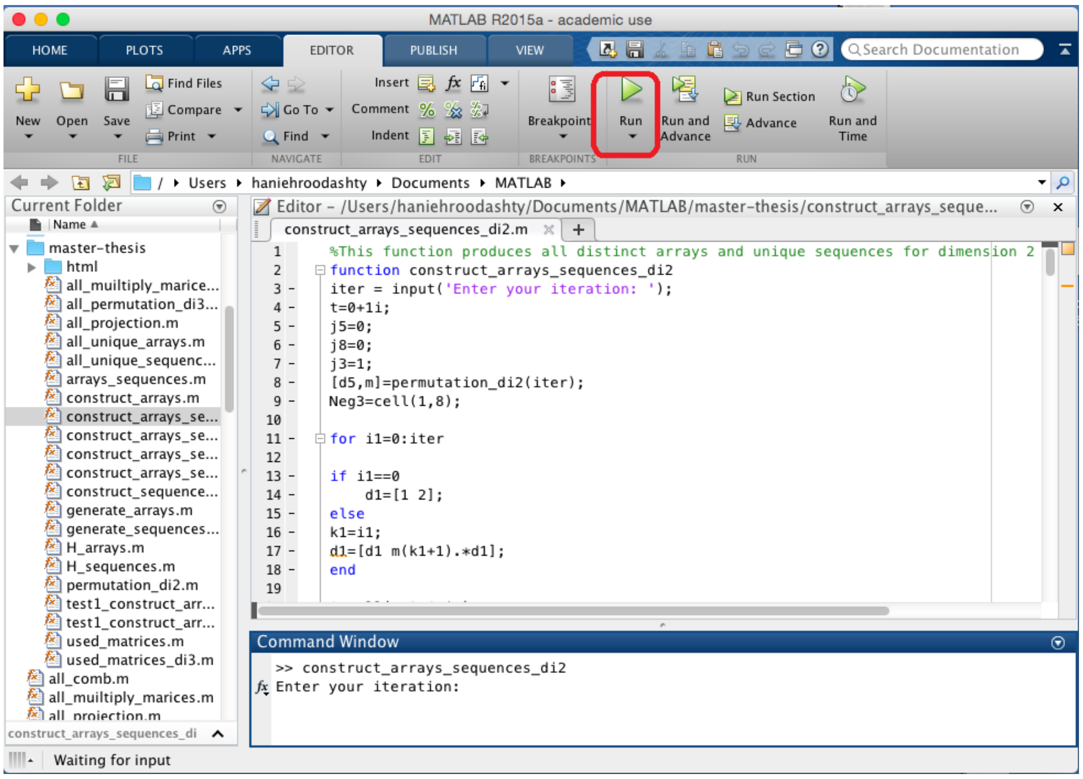
- Open the code in MATLAB environment.
- Add all functions and subfunctions (eleven items) in the same folder in MATLAB.
- Generating arrays and sequences for dimension 2: Run function construct_arrays_sequences_di2. You will be prompted with the message: “Enter your iteration:”. In the command window.
- Enter the number of iteration you are interested in. The iteration can be any integer but higher iterations need a huge memory and strong processor to run.
    - Iteration=0 is defined in the code by default and cannot be entered in this step.
- After entering the number and running the code, the output will be the number of distinct arrays and sequences for any iteration separately.
- When the code has completely run, all distinct arrays and sequences are stored on 'array-di2.txt' and 'sequence-di2.txt'files.
- Type 'array-di2.txt' or 'sequence-di2.txt' in the command window to retrieve the output.
- To generate arrays and sequences for dimension 2 without linear offset one can run the function construct_arrays_sequences_without_linearoffset.
- All the next steps are similar to the above with the same steps including the input to the code, output and the place for storage.
- To generate arrays and sequences for dimension 3: Run function construct_arrays_sequences_di3.
- Enter the number of iteration.
- The output will be stored in 'array-di3.txt'and 'sequence-di3.txt'files.
- To generate arrays and sequences for dimension 3 without linear offset one can run the function construct_arrays_sequences_without_linearoffset.
- All the next steps are similar to the above for dimension 3 with the same steps including the input to the code, output and the place for storage.



```
%This function produces all distinct arrays and unique sequences for dimension 2
function construct_arrays_sequences_di2
iter = input('Enter your iteration: ');
t=0+1i;
j5=0;
j8=0;
j3=1;
[d5,m]=permutation_di2(iter);
Neg3=cell(1,8);

for i1=0:iter

if i1==0
    d1=[1 2];
else
k1=i1;
d1=[d1 m(k1+1).*d1];
end

```

```
>>
```

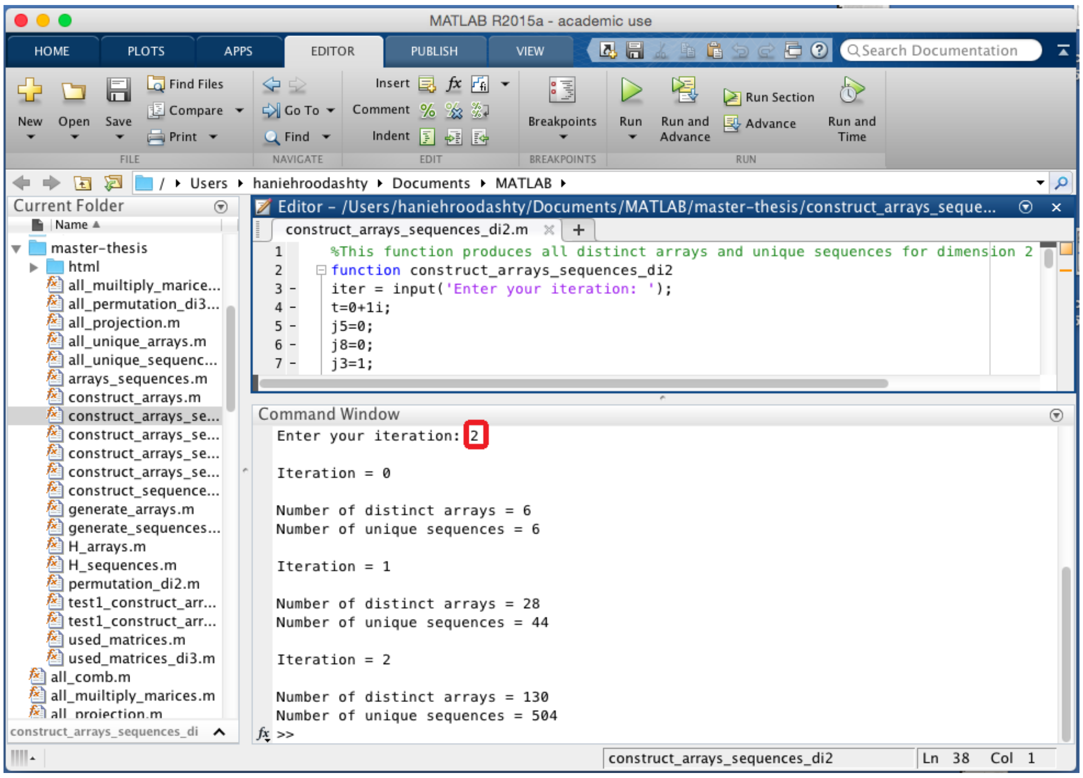
Figure A-1: Run the code in MATLAB.



```
%This function produces all distinct arrays and unique sequences for dimension 2
function construct_arrays_sequences_di2
iter = input('Enter your iteration: ');
t=0+1i;
j5=0;
j8=0;
j3=1;
[d5,m]=permutation_di2(iter);
Neg3=cell(1,8);

for i1=0:iter

if i1==0
    d1=[1 2];
else
k1=i1;
d1=[d1 m(k1+1).*d1];
end

```

```
>> construct_arrays_sequences_di2
Enter your iteration:
```

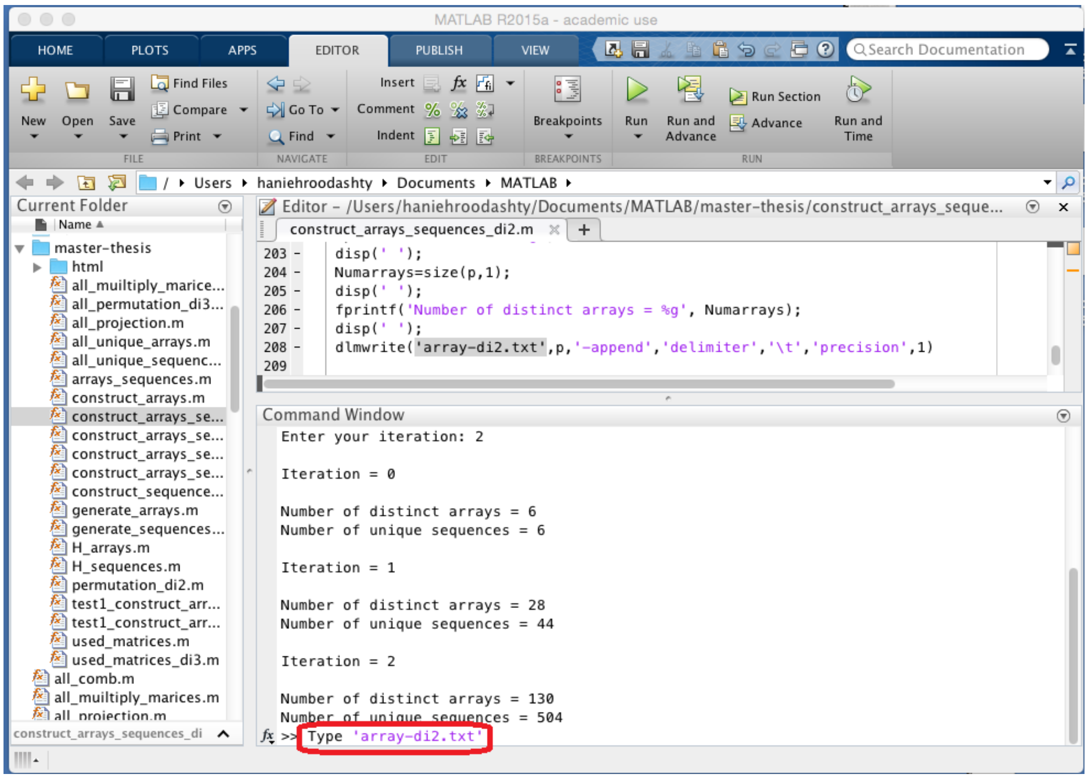
Figure A-2: Enter the iteration and press enter.



```
%This function produces all distinct arrays and unique sequences for dimension 2
function construct_arrays_sequences_di2
iter = input('Enter your iteration: ');
t=0+1i;
j5=0;
j8=0;
j3=1;
```

```
Enter your iteration: 2

Iteration = 0

Number of distinct arrays = 6
Number of unique sequences = 6

Iteration = 1

Number of distinct arrays = 28
Number of unique sequences = 44

Iteration = 2

Number of distinct arrays = 130
Number of unique sequences = 504
>>
```

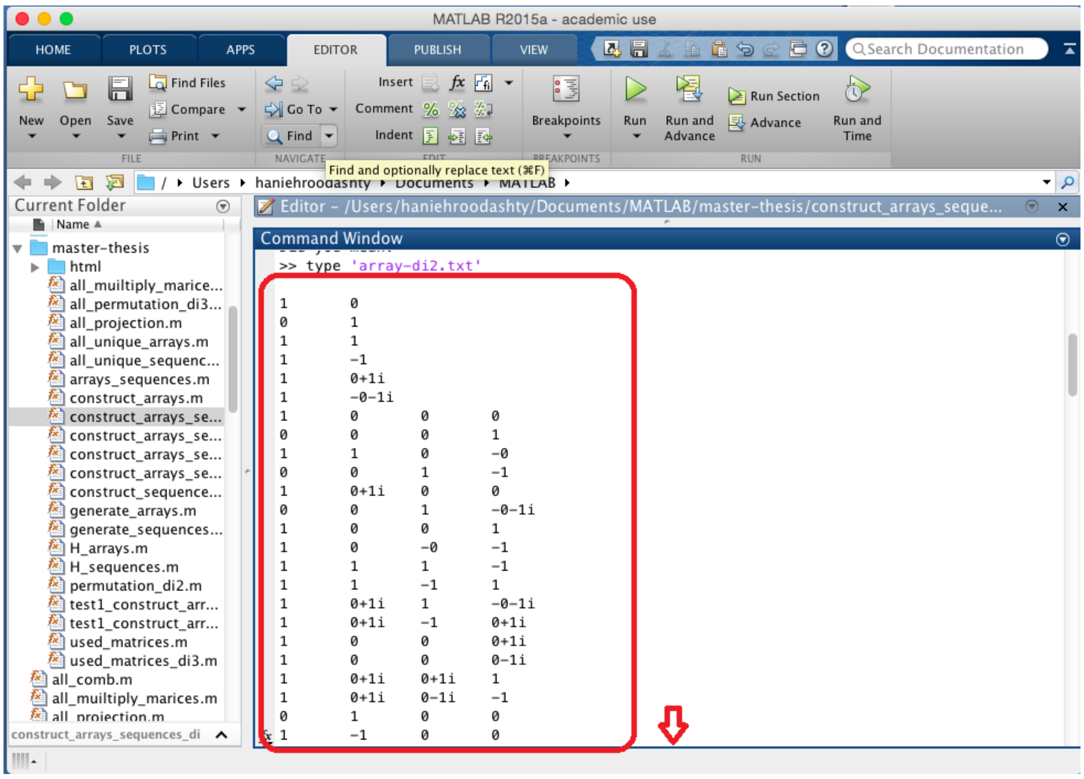
FIGURE A-3: The number of arrays and sequences are presented for any iterations.



```
disp(' ');
Numarrays=size(p,1);
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array-di2.txt',p,'-append','delimiter','\t','precision',1)
```

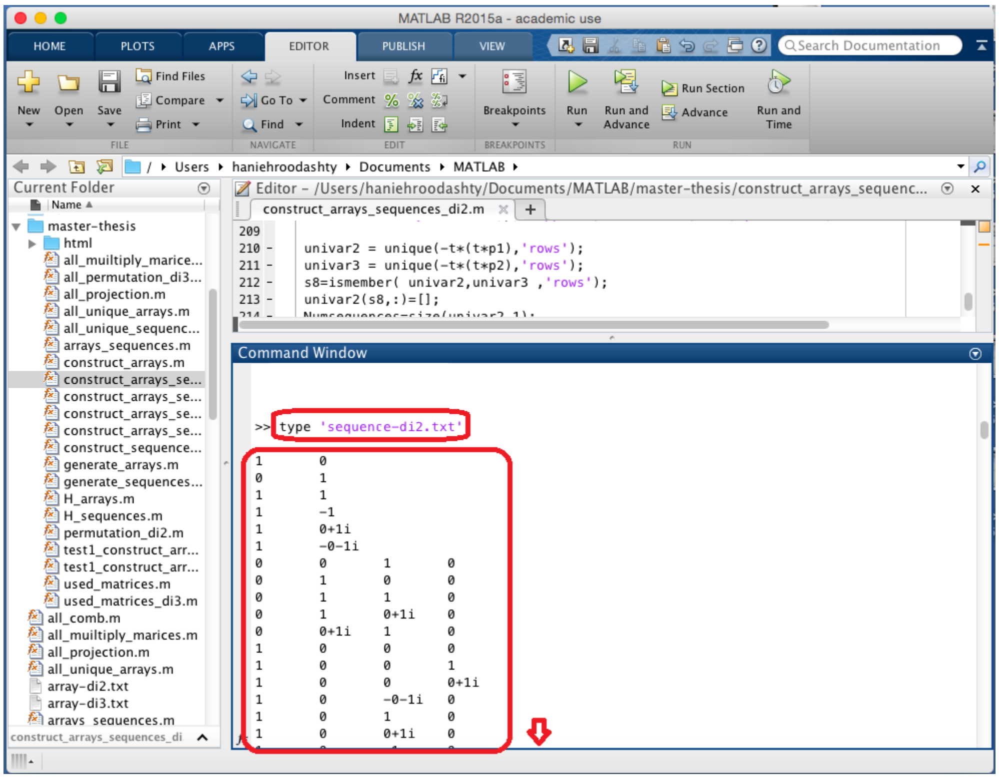
```
Enter your iteration: 2

Iteration = 0

Number of distinct arrays = 6
Number of unique sequences = 6

Iteration = 1

Number of distinct arrays = 28
Number of unique sequences = 44

Iteration = 2

Number of distinct arrays = 130
Number of unique sequences = 504
>> Type 'array-di2.txt'
```

FIGURE A-4: Type 'array-di2.txt' to obtain all the distinct arrays.


MATLAB R2015a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Find and optionally replace text (⌘F)
Editor – /Users/haniehroodashty/Documents/MATLAB/master-thesis/construct_arrays_seque...
Current Folder
Command Window

```
>> type 'array-di2.txt'

1       0
0       1
1       1
1       -1
1       0+1i
1       -0-1i
1       0       0       0
0       0       0       1
1       1       0       -0
0       0       1       -1
1       0+1i    0       0
0       0       1       -0-1i
1       0       0       1
1       0       -0      -1
1       1       1       -1
1       1       -1      1
1       0+1i    1       -0-1i
1       0+1i    -1      0+1i
1       0       0       0+1i
1       0       0       0-1i
1       0+1i    0+1i    1
1       0+1i    0-1i    -1
0       1       0       0
1       -1      0       0
```

FIGURE A-5: All distinct arrays will be presented. Scroll down to see entire list.


MATLAB R2015a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Editor – /Users/haniehroodashty/Documents/MATLAB/master-thesis/construct_arrays_sequenc...
construct_arrays_sequences_di2.m
Current Folder
Command Window

```
univar2 = unique(-t*(t*p1),'rows');
univar3 = unique(-t*(t*p2),'rows');
s8=ismember( univar2,univar3 ,'rows');
univar2(s8,:)=[];
```

```
>> type 'sequence-di2.txt'
1       0
0       1
1       1
1       -1
1       0+1i
1       -0-1i
0       0       1       0
0       1       0       0
0       1       1       0
0       1       0+1i    0
0       0+1i    1       0
1       0       0       0
1       0       0       1
1       0       0       0+1i
1       0       -0-1i   0
1       0       1       0
1       0       0+1i    0
```

FIGURE A-6: Type 'sequence-di2.txt' and scroll down to see all the distinct sequences

Appendix 2: Program Sources

A Dimension 2

A-1 Function construct_arrays_sequences_di2

```
%This function produces all distinct arrays and unique sequences
for dimension 2 with linear offset
function construct_arrays_sequences_di2
iter = input('Enter your iteration: ');
t=0+1i;
j5=0;
j8=0;
j3=1;
[d5,m]=permutation_di2(iter);
Neg3=cell(1,8);

for i1=0:iter

if i1==0
    d1=[1 2];
else
k1=i1;
d1=[d1 m(k1+1).*d1];
end

Q=cell(1,3*6^i1);
prev2=cell (1,3*6^i1);
n=i1+1;
w1=used_matrices(n);
sizw=size(w1,1);

if i1==0
        prev{1}=[1 0;0 1];
        prev{2}=[1 1;1 -1];
        prev{3}=[1 t;1 -t];
        prev{4}=[0 1;1 0];
        prev{5}=[1,-1;1 1];
        prev{6}=[1 -t;1 t];
        p(1:2,1:2)=prev{1};
        p(3:4,1:2)=prev{2};
        p(5:6,1:2)=prev{3};
        disp(' ');
        fprintf('Iteration = %g',i1);
        disp(' ');
        disp(' ');
        fprintf('Number of distinct arrays = %g',6);
        disp(' ');
        fprintf('Number of unique sequences = %g',6);
        disp(' ');
        dlmwrite('array-di2.txt',p,'delimiter','\t','precision',1)
        dlmwrite('sequence-di2.txt',p,'delimiter','\t','precision',1)
    else
        k=1;

        for j=1:6^i1
           if j>1
               if mod(j-1,3^i1)==0
                    k=k+2*3^i1;
               end
           end
Q{k}=[prev{j}(1,:) 0*prev{j}(2,:);0*prev{j}(1,:) prev{j}(2,:)];
prev2{k}=Q{k}([2 1],:);

Q{k+3^i1}=[prev{j}(1,:) prev{j}(2,:);prev{j}(1,:) -prev{j}(2,:)];
prev2{k+3^i1}= Q{k+3^i1}([2 1],:);

Q{k+2*3^i1}=[prev{j}(1,:) t*prev{j}(2,:);prev{j}(1,:) -t*prev{j}(2,:)];
prev2{k+2*3^i1}= Q{k+2*3^i1}([2 1],:);
k=k+1;
        end

         k=(k+2*3^i1)-1;
         prev=[Q,prev2];

p1=zeros(k*factorial(i1+1),2^(i1+1));
p2=zeros(k*factorial(i1+1),2^(i1+1));
j3 =j3+factorial(i1);
s=size(Q,2);
p=zeros(2*s,2^(i1+1));
r=-1;
k1=0;

  for jj=1:k
      k1=k1+1;

      if jj>1
         if mod(jj-1,3^(i1+1))==0
            k1=1;
         end
      end
      if Q{jj}==zeros(2,2^(i1+1))
      else
      if w1(k1,:)=='0'
           Q{jj}=zeros(2,2^(i1+1));
      else

     d2{1}=d1(Q{jj}(1,:)==-1);
       d2{2}=d1(Q{jj}(1,:)==t);
        d2{3}=d1(Q{jj}(1,:)==1);
         d2{4}=d1(Q{jj}(1,:)==-t);

      d2{5}=d1(Q{jj}(2,:)==-1);
       d2{6}=d1(Q{jj}(2,:)==t);
        d2{7}=d1(Q{jj}(2,:)==1);
         d2{8}=d1(Q{jj}(2,:)==-t);

         j4=0;
         j6=j5+1;
         j7=j8+1;

             for j2=j3:((factorial(i1+1)+j3)-1)
                 j5=j5+1;
                 j8=j8+1;
                 if d1 == d5{j2}
                      p1(j5,:)=Q{jj}(1,:);
                      p2(j8,:)=Q{jj}(2,:);
                 else
                  for ll=1:8
                   j4=j4+1;
                   Neg3{j4}=find (ismember(d5{j2},d2{ll}));
                  end
                  p1(j5,[Neg3{1}])=-1;
                  p1(j5,[Neg3{2}])=t;
                  p1(j5,[Neg3{3}])=1;
                  p1(j5,[Neg3{4}])=-t;

                  p2(j8,[Neg3{5}])=-1;
                  p2(j8,[Neg3{6}])=t;
                  p2(j8,[Neg3{7}])=1;
                  p2(j8,[Neg3{8}])=-t;

                  j4=0;

                  for s=jj+size(w1,1):size(w1,1):k
                    s1= ismember(Q{s},[p1(j5,:);p2(j8,:)],'rows');
                    Q{s}(s1,:)=0;

                     if p1(j5,1)==0 || p2(j8,1)==0
s2=ismember(Q{s},-1*[p1(j5,:);p2(j8,:)],'rows');
s3=ismember((-t*(t*Q{s})),(t*[p1(j5,:);p2(j8,:)]),'rows');
s4=ismember((-t*(t*Q{s})),(-(t*[p1(j5,:);p2(j8,:)])),'rows');
                      Q{s}(s2,:)=0;
                      Q{s}(s3,:)=0;
                      Q{s}(s4,:)=0;
                     end
                  end

                   if jj>sizw

                  if all(p1(j5,:)==0) || all(p2(j8,:)==0)
                  else

                      p3=[p2(j8,:);p1(j5,:)];
                  if Q{jj}==p3(1:2,:)
                     Q{jj}(2,:)=0;
                    p2(j8,:)=0;
                  else

                  if p1(j5,1)==0

                      if Q{jj}(2,:)==-1*p1(j5,:)
                          Q{jj}(2,:)=0;
                           p2(j8,:)=0;
                      else

                      if (-t*(t*Q{jj}(2,:)))==(t*p1(j5,:))
                        Q{jj}(2,:)=0;
                         p2(j8,:)=0;
                      else

                     if (-t*(t*Q{jj}(2,:)))==(-(t*p1(j5,:)));
                         Q{jj}(2,:)=0;
                          p2(j8,:)=0;
                     end
                      end
                      end
                  end
                  end

                  end
                  end
                 end
            if jj>sizw
             if j2 > j3
               if p1(j5,1)==0 || p2(j8,1)==0
s5=ismember([p1(j6:j5-1,:);p2(j7:j8-1,:)],-1*[p1(j5,:);p2(j8,:)],'rows');
s6=ismember((-t*(t*[p1(j6:j5-1,:);p2(j7:j8-1,:)])),(t*[p1(j5,:);p2(j8,:)]),'rows');
s7=ismember((-t*(t*[p1(j6:j5-1,:);p2(j7:j8-1,:)])),(-(t*[p1(j5,:);p2(j8,:)])),'rows');

             k2=[p1(j6:j5-1,:);p2(j7:j8-1,:)];
             k2(s5,:)=0;
             k2(s6,:)=0;
             k2(s7,:)=0;
             si1=size(p1(j6:j5-1,:),1);
             p1(j6:j5-1,:)=k2( 1:si1,:);
             p2(j7:j8-1,:)=k2(si1+1:end,:);
               end
             end
            end
             end
             r=r+2;
             p(r:r+1,:)=Q{jj};
      end
      end
  end
p = p(any(p,2),:);
disp(' ');
fprintf('Iteration = %g',i1);
disp(' ');
Numarrays=size(p,1);
```

```
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array-di2.txt',p,'-
append','delimiter','\t','precision',1)

univar2 = unique(-t*(t*p1),'rows');
univar3 = unique(-t*(t*p2),'rows');
s8=ismember( univar2,univar3 ,'rows');
univar2(s8,:)=[];
Numsequences=size(univar2,1);
Numsequences1=size(univar3,1);
totalsize= Numsequences+Numsequences1-1;

fprintf('Number of unique sequences = %g',totalsize);
disp(' ');
dlmwrite('sequence-di2.txt',univar2,'-
append','delimiter','\t','precision',1)
dlmwrite('sequence-di2.txt',univar3,'-
append','delimiter','\t','precision',1)
end
j5=0;
j8=0;
end
end
```

A-2 Function permutation_di2

```
%This function produces 1, Z0, Z1, Z0Z1...by numeric values for dimension 2
function [d5,m]=permutation_di2(iter)
j1=1;
d5=cell(1,2^(iter+1));
d3=cell(1,2^(iter+1));
for i2=0:iter
    if i2==0
        d3{1}=[1 2];
        d5{1}=d3{1};
    else
        pri=primes(60);
        m=pri(1:i2+1);
        p=perms(m);

        for i=1:factorial((i2+1))
             k=p(i,1);
              d3{1}=[1 k];
           for j=1:(i2)
              k=p(i,j+1);
              d3{j+1}=[d3{j} k.*d3{j}];
           end
              j1=j1+1;
               d5{j1}=d3{j+1};
        end
    end
end
end
```

A-3 Function used_matrices

```
1   % All possible combinations of 3 matrices(I,H,N) in each iteration
    without flip combinations
2   % is found by this function.
3   % n is equal to iteration +1 (n=i+1)
4   %I found line 7-19 from internet. It is wrote by Abdulrahman Ikram
    Siddiq.
5   % for useing this function we must write two lines:
6   % n=3 (give value to n)
7   % and
8   % w1=used_matrices(n)
9   function m=used_matrices(n)
10  alphabet=['I' 'H' 'N'];
11  L=length(alphabet);
12  for i=n:-1:1
13      v=[];
14      for j=1:L
15          v=[v alphabet(j)*ones(1,L^(i-1))];
16      end
17      cv=[];
18      Lv=length(v);
19      for k=1:(L^n)/Lv
20          cv=[cv v];
21      end
22       m(1:L^n,n-i+1)=cv';
23  end
24  m=char(m);
25
26  for e1=1:3^n
27      if m(e1,n)~='I'
28     N = fliplr(m(e1,:));
29     N3=N;
30     N2 = find(N =='I');
31     for e2=1:length(N2)
32       N3(N2(e2))=N3(N2(e2)-1);
33       N3(N2(e2)-1)='I';
34     end
35     if N(1,n)~='I'
36     N1 =find(ismember(m,N,'rows'));
37
38      if N1~=e1
39     m(N1,:)='0';
40      end
41     end
42     N4=find(ismember(m,N3,'rows'));
43
44     if N4~=e1
45     m(N4,:)='0';
46     end
47      end
48  end
49  %disp(m)
50  end
```

A-4 Function construct_arrays_sequences_without_linearoffset

```
%This function produces all distinct arrays and unique sequences
for dimension 2 without linear offset
function construct_arrays_sequences_without_linearoffset
iter = input('Enter your iteration: ');
t=0+1i;
j5=0;
j3=1;
[d5,m]=permutation_di2(iter);
Neg3=cell(1,8);
for i1=0:iter
if i1==0
  d1=[1 2];
else
k1=i1;
d1=[d1 m(k1+1).*d1];
end
Q=cell(1,3*3^i1);
n=i1+1;
w1=used_matrices(n);

if i1==0
        prev{1}=[1 0;0 1];
        prev{2}=[1 1;1 -1];
        prev{3}=[1 t;1 -t];
        p(1:2,1:2)=prev{1};
        p(3:4,1:2)=prev{2};
        p(5:6,1:2)=prev{3};
        disp(' ');
        fprintf('Iteration = %g',i1);
        disp(' ');
        disp(' ');
        fprintf('Number of distinct arrays = %g',6);
        disp(' ');
        fprintf('Number of unique sequences = %g',6);
        disp(' ');
        dlmwrite('array-di2.txt',p,'delimiter','\t','precision',1)
        dlmwrite('sequence-
di2.txt',p,'delimiter','\t','precision',1)
    else
        k=1;
        for j=1:3^i1
Q{k}=[prev{j}(1,:) 0*prev{j}(2,:);0*prev{j}(1,:) prev{j}(2,:)];

Q{k+3^i1}=[prev{j}(1,:) prev{j}(2,:);prev{j}(1,:) -prev{j}(2,:)];

Q{k+2*3^i1}=[prev{j}(1,:) t*prev{j}(2,:);prev{j}(1,:) -
t*prev{j}(2,:)];
k=k+1;
        end
k=(k+2*3^i1)-1;
prev=Q;

p1=zeros(k*factorial(i1+1),2^(i1+1));
p2=zeros(k*factorial(i1+1),2^(i1+1));
```

```
j3 =j3+factorial(i1);
s=size(Q,2);
p=zeros(s,2^(i1+1));
r=-1;
k1=0;
  for jj=1:k
        k1=k1+1;

        if w1(k1,:)=='0'
             Q{jj}=zeros(2,2^(i1+1));
        else

       d2{1}=d1(Q{jj}(1,:)==-1);
         d2{2}=d1(Q{jj}(1,:)==t);
          d2{3}=d1(Q{jj}(1,:)==1);
           d2{4}=d1(Q{jj}(1,:)==-t);

        d2{5}=d1(Q{jj}(2,:)==-1);
         d2{6}=d1(Q{jj}(2,:)==t);
          d2{7}=d1(Q{jj}(2,:)==1);
           d2{8}=d1(Q{jj}(2,:)==-t);
           j4=0;
               for j2=j3:((factorial(i1+1)+j3)-1) %j2=2:3
                      j5=j5+1;
                   if d1 == d5{j2}
                      p1(j5,:)=Q{jj}(1,:);
                      p2(j5,:)=Q{jj}(2,:);
                   else
                    for ll=1:8
                        j4=j4+1;
                        Neg3{j4}=find (ismember(d5{j2},d2{ll}));
                    end
                    p1(j5,[Neg3{1}])=-1;
                    p1(j5,[Neg3{2}])=t;
                    p1(j5,[Neg3{3}])=1;
                    p1(j5,[Neg3{4}])=-t;

                    p2(j5,[Neg3{5}])=-1;
                    p2(j5,[Neg3{6}])=t;
                    p2(j5,[Neg3{7}])=1;
                    p2(j5,[Neg3{8}])=-t;
                    j4=0;
                  end
               end
               r=r+2;
               p(r:r+1,:)=Q{jj};
        end
  end
disp(' ');
fprintf('Iteration = %g',i1);
disp(' ');
Numarrays=size(p,1);
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array-di2.txt',p,'-
append','delimiter','\t','precision',1)

univar2 = unique(-t*(t*p1),'rows');
univar3 = unique(-t*(t*p2),'rows');
s8=ismember( univar2,univar3 ,'rows');
univar2(s8,:)=[];

Numsequences=size(univar2,1);
Numsequences1=size(univar3,1);
totalsize= Numsequences+Numsequences1-1;
fprintf('Number of unique sequences = %g',totalsize);
disp(' ');

dlmwrite('sequence-di2.txt',univar2,'-
append','delimiter','\t','precision',1)
dlmwrite('sequence-di2.txt',univar3,'-
append','delimiter','\t','precision',1)
end
j5=0;
end
end
```

A-5 Function construct_arrays

```
%This function produces all distinct arrays for dimension 2 with
linear offset
function construct_arrays
 iter = input('Enter your iteration: ');
t=0+1i;
j5=-1;
j3=1;
[d5,m]=permutation_di2(iter);
Neg3=cell(1,8);

for i1=0:iter

if i1==0
    d1=[1 2];
else
k1=i1;
d1=[d1 m(k1+1).*d1];
end
Q=cell(1,3*6^i1);
prev2=cell (1,3*6^i1);

n=i1+1;
w1=used_matrices(n);

if i1==0
        prev{1}=[1 0;0 1];
        prev{2}=[1 1;1 -1];
        prev{3}=[1 t;1 -t];
        prev{4}=[0 1;1 0];
        prev{5}=[1,-1;1 1];
        prev{6}=[1 -t;1 t];
        p(1:2,1:2)=prev{1};
        p(3:4,1:2)=prev{2};
        p(5:6,1:2)=prev{3};
    else
        k=1;
        for j=1:6^i1
           if j>1
               if mod(j-1,3^i1)==0
                    k=k+2*3^i1;
               end
           end
Q{k}=[prev{j}(1,:) 0*prev{j}(2,:);0*prev{j}(1,:) prev{j}(2,:)];
prev2{k}=Q{k}([2 1],:);

Q{k+3^i1}=[prev{j}(1,:) prev{j}(2,:);prev{j}(1,:) -prev{j}(2,:)];
prev2{k+3^i1}= Q{k+3^i1}([2 1],:);

Q{k+2*3^i1}=[prev{j}(1,:) t*prev{j}(2,:);prev{j}(1,:) -
t*prev{j}(2,:)];
prev2{k+2*3^i1}= Q{k+2*3^i1}([2 1],:);
k=k+1;
        end
k=(k+2*3^i1)-1;
```

```
prev=[Q,prev2];
p1(1:2*k*factorial(i1+1),1:2^(i1+1))=t;
j3 =j3+factorial(i1);
s=size(Q,2);
p=zeros(2*s,2^(i1+1));
r=-1;
k1=0;
  for jj=1:k
      k1=k1+1;

      if jj>1
         if mod(jj-1,3^(i1+1))==0
            k1=1;
         end
      end
      if Q{jj}==zeros(2,2^(i1+1))
      else
      if w1(k1,:)=='0'
          Q{jj}=zeros(2,2^(i1+1));
      else

     d2{1}=d1(Q{jj}(1,:)==-1);
       d2{2}=d1(Q{jj}(1,:)==0);
        d2{3}=d1(Q{jj}(1,:)==1);
         d2{4}=d1(Q{jj}(1,:)==-t);

      d2{5}=d1(Q{jj}(2,:)==-1);
       d2{6}=d1(Q{jj}(2,:)==0);
        d2{7}=d1(Q{jj}(2,:)==1);
         d2{8}=d1(Q{jj}(2,:)==-t);

         j4=0;

             for j2=j3:((factorial(i1+1)+j3)-1) %j2=2:3
                 j5=j5+2;
                 if d1 == d5{j2}
                    j5=j5-2;
                 else
                  for ll=1:8
                   j4=j4+1;
                   Neg3{j4}=find (ismember(d5{j2},d2{ll}));
                  end
                  p1(j5,[Neg3{1}])=-1;
                  p1(j5,[Neg3{2}])=0;
                  p1(j5,[Neg3{3}])=1;
                  p1(j5,[Neg3{4}])=-t;
                  p1(j5+1,[Neg3{5}])=-1;
                  p1(j5+1,[Neg3{6}])=0;
                  p1(j5+1,[Neg3{7}])=1;
                  p1(j5+1,[Neg3{8}])=-t;
                  j4=0;

                  for s=jj+size(w1,1):size(w1,1):k
                    s1= ismember(Q{s},p1(j5:j5+1,:),'rows');
                    Q{s}(s1,:)=0;

```

```
                     if p1(j5,1)==0 || p1(j5+1,1)==0
           s2=ismember(Q{s},-1*p1(j5:j5+1,:),'rows');
           s3=ismember((-t*(t*Q{s})),(t*p1(j5:j5+1,:)),'rows');
           s4=ismember((-t*(t*Q{s})),(-(t*p1(j5:j5+1,:))),'rows');
                      Q{s}(s2,:)=0;
                      Q{s}(s3,:)=0;
                      Q{s}(s4,:)=0;
                     end
                  end
                  if all(p1(j5,:)==0) || all(p1(j5+1,:)==0)
                  else
                  if Q{jj}==p1([j5+1 j5],:)
                         Q{jj}(2,:)=0;
                  end
                  if p1(j5,1)==0

                      if Q{jj}(2,:)==-1*p1(j5,:)
                          Q{jj}(2,:)=0;
                      end

                   if (-t*(t*Q{jj}(2,:)))==(t*p1(j5,:))
                          Q{jj}(2,:)=0;
                   end

                     if (-t*(t*Q{jj}(2,:)))==(-(t*p1(j5,:)));
                          Q{jj}(2,:)=0;
                     end

                  end
                  end
                 end
             end
             r=r+2;
             p(r:r+1,:)=Q{jj};
      end
      end
  end
p = p(any(p,2),:);
end
disp(' ');
fprintf('Iteration = %g',i1);
disp(' ');
Numarrays=size(p,1);
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array-di2.txt',p,'-append','delimiter','\t','precision',1)
end
end
```

A-6 Function H_arrays

```matlab
%This function produces all distinct arrays for U={H}, P={I,X} in
dimension 2
function H_arrays
iter = input('Enter your iteration: ');
rows=2;
prev=cell (rows,2^iter);
for i=0:iter
j=0;
k=0;
columns=2^(i+1);
matrix=cell (rows,columns);
if i==0
prev{1}=[1;1];
end
for jj=1:2^(i+1)

   if jj<=2^i
       j=j+1;
       matrix{jj}(1,:)=[prev{j}(1,:) prev{j}(2,:)] ;
       matrix{jj}(2,:)=[prev{j}(1,:) -prev{j}(2,:)];
   else
       k=k+1;
       matrix{jj}(1,:)=[prev{k}(1,:) -prev{k}(2,:)];
       matrix{jj}(2,:)=[prev{k}(1,:) prev{k}(2,:)] ;

   end
end
prev=matrix;
fprintf('Iteration = %g',i);
disp(' ');
Numarrays=size(matrix,2);
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
for jj=1:2^(i)
     disp (matrix{jj});
end
end
```

A-7 Function H_sequences

```
%This function produces all unique arrays and sequences for
U={H}, P={I,X} in dimension 2
function H_sequences
iter = input('Enter your iteration: ');
rows=2;
prev=cell (rows,2^iter);
j3=1;
j5=-1;
[d5,m]=permutation_di2(iter);

for i1=0:iter
j=0;
k=0;
columns=2^(i1+1);
matrix=cell (rows,columns);

if i1==0
    d1=[1 2];
else
k1=i1;
d1=[d1 m(k1+1).*d1];
end

if i1==0
prev{1}=[1;1];
p=[1 1;1 -1];
p1=[1 1;1 -1];
end

for jj=1:2^(i1+1)

   if jj<=2^i1
       j=j+1;
       matrix{jj}(1,:)=[prev{j}(1,:) prev{j}(2,:)] ;
       matrix{jj}(2,:)=[prev{j}(1,:) -prev{j}(2,:)];

   else
       k=k+1;
       matrix{jj}(1,:)=[prev{k}(1,:) -prev{k}(2,:)];
       matrix{jj}(2,:)=[prev{k}(1,:) prev{k}(2,:)] ;

   end

end

prev=matrix;
if i1~=0
p=zeros(2^i1,2^(i1+1));
p1(1:2^(i1+1)*factorial(i1+1),1:2^(i1+1))=1;
j3 =j3+factorial(i1);
r=-1;

for jj=1:2^i1
```

```

     d2{1}=d1(matrix{jj}(1,:)==-1);
       d2{2}=d1(matrix{jj}(2,:)==-1);
         j4=0;
             for j2=j3:((factorial(i1+1)+j3)-1)
                 j5=j5+2;
                 if d1 == d5{j2}
                    p1(j5:j5+1,:)=matrix{jj};
                 else
                  for ll=1:2
                   j4=j4+1;
                   Neg3{j4}=find (ismember(d5{j2},d2{ll}));
                  end
                  p1(j5,[Neg3{1}])=-1;
                  p1(j5+1,[Neg3{2}])=-1;
                  j4=0;
                 end
             end
r=r+2;
p(r:r+1,:)=matrix{jj};
end
end
fprintf('Iteration = %g',i1);
disp(' ');
Numarrays=size(p,1);
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array.txt',p,'-append','delimiter','\t','precision',2)
disp(' ');

univar =unique(p1,'rows');
Numarrays2=size( univar,1);
fprintf('Number of unique sequences = %g', Numarrays2);
disp(' ');
dlmwrite('sequence.txt',univar,'-
append','delimiter','\t','precision',2)

j5=-1;
end
```

B Dimension 3

B-1 Function construct_arrays_sequences_di3

```
%This function produces all distinct arrays and sequences for
dimension 3 with linear offset
function construct_arrays_sequences_di3
iter = input('Enter your iteration: ');
j5=0;
j3=1;
a=exp(2*pi*1i/3);
[d5,m]=all_permutation_di3(iter);
Neg3=cell(1,8);

for i1=0:iter
    if i1==0
    d1=[1 2 4];
    else
    k1=i1;
    d1=[d1 m(k1+1).*d1 (m(k1+1))^2.*d1];
    end
    Q=cell(1,4*24^i1);
    prev2=cell(1,4*24^i1);
    prev3=cell(1,4*24^i1);
    prev4=cell(1,4*24^i1);
    prev5=cell(1,4*24^i1);
    prev6=cell(1,4*24^i1);

n=i1+1;
w1=used_matrices_di3(n);
sizw=size(w1,1);

        if i1==0

            prev{1}=[1 0 0;0 1 0;0 0 1];
                 prev{5}=prev{1}([2 3 1],:);
                 prev{9}=prev{1}([1 3 2],:);
                 prev{13}=prev{1}([2 1 3],:);
                 prev{17}=prev{1}([3 2 1],:);
                 prev{21}=prev{1}([3 1 2],:);

            prev{2}=[1 1 1;1 a a^2;1 a^2 a];
                 prev{6}=prev{2}([2 3 1],:);
                 prev{10}=prev{2}([1 3 2],:);
                 prev{14}=prev{2}([2 1 3],:);
                 prev{18}=prev{2}([3 2 1],:);
                 prev{22}=prev{2}([3 1 2],:);

            prev{3}=[1 a a;1 a^2 1;1 1 a^2];
                 prev{7}=prev{3}([2 3 1],:);
                 prev{11}=prev{3}([1 3 2],:);
                 prev{15}=prev{3}([2 1 3],:);
                 prev{19}=prev{3}([3 2 1],:);
                 prev{23}=prev{3}([3 1 2],:);

            prev{4}=[1 a^2 a^2;1 1 a;1 a 1];
                 prev{8}=prev{4}([2 3 1],:);
                 prev{12}=prev{4}([1 3 2],:);
                 prev{16}=prev{4}([2 1 3],:);
                 prev{20}=prev{4}([3 2 1],:);
                 prev{24}=prev{4}([3 1 2],:);

                 p(1:3,:)=prev{1};
                 p(4:6,:)=prev{2};
                 p(7:9,:)=prev{3};
                 p(10:11,:)=prev{4}(2:3,:);
        disp(' ');
        fprintf('Iteration = %g',i1);
        disp(' ');
        disp(' ');
        fprintf('Number of distinct arrays = %g',11);
        disp(' ');
        fprintf('Number of unique sequences = %g',11);
        disp(' ');
        dlmwrite('array-di3.txt',p,'delimiter','\t','precision',2)
        dlmwrite('sequence-di3.txt',p,'delimiter','\t','precision',2)
        else
            k=1;
               for j=1:24^(i1)
                    if j>1
                       if mod(j-1,4^i1)==0
                        k=k+3*4^i1;
                       end
                    end
Q{k}=[prev{j}(1,:) 0*prev{j}(2,:) 0*prev{j}(3,:);0*prev{j}(1,:) prev{j}(2,:) 0*prev{j}(3,:);0*prev{j}(1,:) 0*prev{j}(2,:) prev{j}(3,:)];
                 prev2{k}=Q{k}([2 3 1],:);
                 prev3{k}=Q{k}([1 3 2],:);
                 prev4{k}=Q{k}([2 1 3],:);
                 prev5{k}=Q{k}([3 2 1],:);
                 prev6{k}=Q{k}([3 1 2],:);


Q{k+4^i1}=[prev{j}(1,:) prev{j}(2,:) prev{j}(3,:);prev{j}(1,:) a*prev{j}(2,:) a^2*prev{j}(3,:);prev{j}(1,:) a^2*prev{j}(2,:) a*prev{j}(3,:)];
                 prev2{k+4^i1}=Q{k+4^i1}([2 3 1],:);
                 prev3{k+4^i1}=Q{k+4^i1}([1 3 2],:);
                 prev4{k+4^i1}=Q{k+4^i1}([2 1 3],:);
                 prev5{k+4^i1}=Q{k+4^i1}([3 2 1],:);
                 prev6{k+4^i1}=Q{k+4^i1}([3 1 2],:);


Q{k+2*4^i1}=[prev{j}(1,:) a*prev{j}(2,:) a*prev{j}(3,:);prev{j}(1,:) a^2*prev{j}(2,:) prev{j}(3,:);prev{j}(1,:) prev{j}(2,:) a^2*prev{j}(3,:)];
                 prev2{k+2*4^i1}=Q{k+2*4^i1}([2 3 1],:);
                 prev3{k+2*4^i1}=Q{k+2*4^i1}([1 3 2],:);
```

```
                      prev4{k+2*4^i1}=Q{k+2*4^i1}([2 1 3],:);
                      prev5{k+2*4^i1}=Q{k+2*4^i1}([3 2 1],:);
                      prev6{k+2*4^i1}=Q{k+2*4^i1}([3 1 2],:);


Q{k+3*4^i1}=[prev{j}(1,:) a^2*prev{j}(2,:)
a^2*prev{j}(3,:);prev{j}(1,:) prev{j}(2,:)
a*prev{j}(3,:);prev{j}(1,:) a*prev{j}(2,:) prev{j}(3,:)];
                      prev2{k+3*4^i1}=Q{k+3*4^i1}([2 3 1],:);
                      prev3{k+3*4^i1}=Q{k+3*4^i1}([1 3 2],:);
                      prev4{k+3*4^i1}=Q{k+3*4^i1}([2 1 3],:);
                      prev5{k+3*4^i1}=Q{k+3*4^i1}([3 2 1],:);
                      prev6{k+3*4^i1}=Q{k+3*4^i1}([3 1 2],:);

                      k=k+1;
                    end
k=(k+3*4^i1)-1;
prev=[Q,prev2,prev3,prev4,prev5,prev6];

p1(1:k*factorial(i1+1),1:3^(i1+1))=a;
p2(1:k*factorial(i1+1),1:3^(i1+1))=a;
p3(1:k*factorial(i1+1),1:3^(i1+1))=a;

j3 =j3+factorial(i1);
s=size(Q,2);
p=zeros(3*s,3^(i1+1));
r=-2;
k1=0;
      for jj=1:k
           k1=k1+1;

          if jj>1
             if mod(jj-1,4^(i1+1))==0
              k1=1;
             end
          end
          if Q{jj}==zeros(3,3^(i1+1))
          else
          if w1(k1,:)=='0'
           Q{jj}=zeros(3,3^(i1+1));
          else
      d2{1}=d1(Q{jj}(1,:)==1);
       d2{2}=d1(Q{jj}(1,:)==0);
        d2{3}=d1(Q{jj}(1,:)==a^2);

      d2{4}=d1(Q{jj}(2,:)==1);
       d2{5}=d1(Q{jj}(2,:)==0);
        d2{6}=d1(Q{jj}(2,:)==a^2);

      d2{7}=d1(Q{jj}(3,:)==1);
       d2{8}=d1(Q{jj}(3,:)==0);
        d2{9}=d1(Q{jj}(3,:)==a^2);

          j4=0;

             for j2=j3:((factorial(i1+1)+j3)-1)
```

```
                     j5=j5+1;
                        if d1 == d5{j2}

                         p1(j5,:)=Q{jj}(1,:);
                         p2(j5,:)=Q{jj}(2,:);
                         p3(j5,:)=Q{jj}(3,:);

                        else
                         for ll=1:9
                           j4=j4+1;
                           Neg3{j4}=find (ismember(d5{j2},d2{ll}));
                         end
                      p1(j5,[Neg3{1}])=1;
                      p1(j5,[Neg3{2}])=0;
                      p1(j5,[Neg3{3}])=a^2;

                      p2(j5,[Neg3{4}])=1;
                      p2(j5,[Neg3{5}])=0;
                      p2(j5,[Neg3{6}])=a^2;

                      p3(j5,[Neg3{7}])=1;
                      p3(j5,[Neg3{8}])=0;
                      p3(j5,[Neg3{9}])=a^2;
                      j4=0;
                      for s=jj+size(w1,1):size(w1,1):k

                        p7=[p1(j5,:);p2(j5,:);p3(j5,:)];
                        s1= ismember(Q{s},p7(1:3,:),'rows');
                        Q{s}(s1,:)=0;
                        s4= ismember(Q{s},Q{jj},'rows');
                        Q{s}(s4,:)=0;

                      end
                     if jj>sizw
                         p4=[p2(j5,:);p1(j5,:)];
                      if Q{jj}(1:2,:)==p4(1:2,:)
                         Q{jj}(2,:)=0;
                         p2(j5,:)=0;

                      end
                         p5=[p3(j5,:);p1(j5,:)];
                      if Q{jj}([1 3],:)==p5(1:2,:)
                         Q{jj}(1,:)=0;
                         p1(j5,:)=0;
                      end
                         p6=[p3(j5,:);p2(j5,:)];
                      if Q{jj}([2 3],:)==p6(1:2,:)
                         Q{jj}(3,:)=0;
                         p3(j5,:)=0;
                      end
                     end
                        end
                 end
                 r=r+3;
                 p(r:r+2,:)=Q{jj};
              end
```

```
          end
      end
p = p(any(p,2),:);

disp(' ');
fprintf('Iteration = %g',i1);
disp(' ');
Numarrays=size(p,1);
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array-di3.txt',p,'-append','delimiter','\t','precision',1)

univar2 = unique(p1,'rows');
univar3 = unique(p2,'rows');
univar4 = unique(p3,'rows');

s8=ismember( univar2,univar3 ,'rows');
univar2(s8,:)=[];
s9=ismember( univar2,univar4 ,'rows');
univar2(s9,:)=[];
s10=ismember( univar3,univar4 ,'rows');
univar3(s10,:)=[];

Numsequences1=size(univar2,1);
Numsequences2=size(univar3,1);
Numsequences3=size(univar4,1);
totalsize=Numsequences1+Numsequences2+ Numsequences3-2;

fprintf('Number of unique sequences = %g', totalsize);
disp(' ');
dlmwrite('sequence-di3.txt',univar2,'-append','delimiter','\t','precision',2)
dlmwrite('sequence-di3.txt',univar3,'-append','delimiter','\t','precision',2)
dlmwrite('sequence-di3.txt',univar4,'-append','delimiter','\t','precision',2)
        end
j5=0;
end
end
```

B-2 Function all_permutation_di3

```
%This function produces all permutation of the set [1 z0 z0^2 z1
z0z1 z0^2z1...] for dimension 3 by numeric values
function [d5,m]=all_permutation_di3(iter)
j1=1;
d5=cell(1,2^(iter+1));
d3=cell(1,3^(iter+1));

for i2=0:iter
    if i2==0
        d3{1}=[1 2 4];
        d5{1}=d3{1};
    else
        pri=primes(60);
        m=pri(1:i2+1);
         p=perms(m);

        for i=1:factorial((i2+1))
             k=p(i,1);
             d3{1}=[1 k k^2];
           for j=1:(i2)
              k=p(i,j+1);
              d3{j+1}=[d3{j} k.*d3{j} k^2.*d3{j}];
           end
               j1=j1+1;
               d5{j1}=d3{j+1};
        end
    end
end
end
```

B-3 Function used_matrices_di3

```
% This function produces all possible combinations of the four matrices (I, F, FD, FD^2)
%through different iterations without symmetric combinations.
% n=i+1
% Line 9-20 have been originally written by Abdulrahman Ikram Siddiq.
% Two lines must be added to use this function: assigning value to n and w1=used_matrices(n)
function m=used_matrices_di3(n)
alphabet=['I' 'F' 'Q' 'R'];
L=length(alphabet);
for i=n:-1:1
    v=[];
    for j=1:L
        v=[v alphabet(j)*ones(1,L^(i-1))];
    end
    cv=[];
    Lv=length(v);
    for k=1:(L^n)/Lv
        cv=[cv v];
    end
     m(1:L^n,n-i+1)=cv';
end
m=char(m);

for e1=1:4^n
    if m(e1,n)~='I'
   N = fliplr(m(e1,:));
   N3=N;
   N2 = find(N =='I');
   for e2=1:length(N2)
     N3(N2(e2))=N3(N2(e2)-1);
     N3(N2(e2)-1)='I';
   end
   if N(1,n)~='I'
   N1 =find(ismember(m,N,'rows'));
    if N1~=e1
   m(N1,:)='0';
    end
   end
   N4=find(ismember(m,N3,'rows'));
   if N4~=e1
   m(N4,:)='0';
   end
    end
end
end
```

B-4 Function construct_arrays_sequences_without_linearoffset_di3

```
%This function produces all distinct arrays and sequences for
dimension 3 without linear offset
function construct_arrays_sequences_without_linearoffset_di3
iter = input('Enter your iteration: ');
j5=0;
j3=1;
a=exp(2*pi*1i/3);
[d5,m]=all_permutation_di3(iter);
Neg3=cell(1,8);

for i1=0:iter
    if i1==0
    d1=[1 2 4];
    else
    k1=i1;
    d1=[d1 m(k1+1).*d1 (m(k1+1))^2.*d1];
    end
Q=cell(1,4*24^i1);
n=i1+1;
w1=used_matrices_di3(n);

        if i1==0

            prev{1}=[1 0 0;0 1 0;0 0 1];

            prev{2}=[1 1 1;1 a a^2;1 a^2 a];

            prev{3}=[1 a a;1 a^2 1;1 1 a^2];

            prev{4}=[1 a^2 a^2;1 1 a;1 a 1];

                 p(1:3,:)=prev{1};
                 p(4:6,:)=prev{2};
                 p(7:9,:)=prev{3};
                 p(10:11,:)=prev{4}(2:3,:);
        disp(' ');
        fprintf('Iteration = %g',i1);
        disp(' ');
        disp(' ');
        fprintf('Number of distinct arrays = %g',11);
        disp(' ');
        fprintf('Number of unique sequences = %g',11);
        disp(' ');
        dlmwrite('array-
di3.txt',p,'delimiter','\t','precision',2)
        dlmwrite('sequence-
di3.txt',p,'delimiter','\t','precision',2)

        else
            k=1;

               for j=1:4^(i1)

```

```
Q{k}=[prev{j}(1,:) 0*prev{j}(2,:) 0*prev{j}(3,:);0*prev{j}(1,:) prev{j}(2,:) 0*prev{j}(3,:);0*prev{j}(1,:) 0*prev{j}(2,:) prev{j}(3,:)];

Q{k+4^i1}=[prev{j}(1,:) prev{j}(2,:) prev{j}(3,:);prev{j}(1,:) a*prev{j}(2,:) a^2*prev{j}(3,:);prev{j}(1,:) a^2*prev{j}(2,:) a*prev{j}(3,:)];

Q{k+2*4^i1}=[prev{j}(1,:) a*prev{j}(2,:) a*prev{j}(3,:);prev{j}(1,:) a^2*prev{j}(2,:) prev{j}(3,:);prev{j}(1,:) prev{j}(2,:) a^2*prev{j}(3,:)];

Q{k+3*4^i1}=[prev{j}(1,:) a^2*prev{j}(2,:) a^2*prev{j}(3,:);prev{j}(1,:) prev{j}(2,:) a*prev{j}(3,:);prev{j}(1,:) a*prev{j}(2,:) prev{j}(3,:)];

k=k+1;
               end
k=(k+3*4^i1)-1;
prev=Q;
p1(1:k*factorial(i1+1),1:3^(i1+1))=a;
p2(1:k*factorial(i1+1),1:3^(i1+1))=a;
p3(1:k*factorial(i1+1),1:3^(i1+1))=a;
j3 =j3+factorial(i1);
s=size(Q,2);
p=zeros(s/4,3^(i1+1));
r=-2;
k1=0;
      for jj=1:k
           k1=k1+1;

          if w1(k1,:)=='0'
           Q{jj}=zeros(3,3^(i1+1));
          else
      d2{1}=d1(Q{jj}(1,:)==1);
       d2{2}=d1(Q{jj}(1,:)==0);
        d2{3}=d1(Q{jj}(1,:)==a^2);

      d2{4}=d1(Q{jj}(2,:)==1);
       d2{5}=d1(Q{jj}(2,:)==0);
        d2{6}=d1(Q{jj}(2,:)==a^2);

      d2{7}=d1(Q{jj}(3,:)==1);
       d2{8}=d1(Q{jj}(3,:)==0);
        d2{9}=d1(Q{jj}(3,:)==a^2);

          j4=0;

             for j2=j3:((factorial(i1+1)+j3)-1)
                 j5=j5+1;
                    if d1 == d5{j2}
                     p1(j5,:)=Q{jj}(1,:);
                     p2(j5,:)=Q{jj}(2,:);
                     p3(j5,:)=Q{jj}(3,:);
                    else
                     for ll=1:9
```

```
                       j4=j4+1;
                       Neg3{j4}=find (ismember(d5{j2},d2{l1}));
                     end
                  p1(j5,[Neg3{1}])=1;
                  p1(j5,[Neg3{2}])=0;
                  p1(j5,[Neg3{3}])=a^2;

                  p2(j5,[Neg3{4}])=1;
                  p2(j5,[Neg3{5}])=0;
                  p2(j5,[Neg3{6}])=a^2;

                  p3(j5,[Neg3{7}])=1;
                  p3(j5,[Neg3{8}])=0;
                  p3(j5,[Neg3{9}])=a^2;
                  j4=0;
                    end
             end
               r=r+3;
             p(r:r+2,:)=Q{jj};
          end
      end
p = p(any(p,2),:);
disp(' ');
fprintf('Iteration = %g',i1);
disp(' ');
Numarrays=size(p,1);
disp(' ');
fprintf('Number of distinct arrays = %g', Numarrays);
disp(' ');
dlmwrite('array-di3.txt',p,'-append','delimiter','\t','precision',2)

univar2 = unique(p1,'rows');
univar3 = unique(p2,'rows');
univar4 = unique(p3,'rows');
s8=ismember( univar2,univar3 ,'rows');
univar2(s8,:)=[];
s9=ismember( univar2,univar4 ,'rows');
univar2(s9,:)=[];
s10=ismember( univar3,univar4 ,'rows');
univar3(s10,:)=[];

Numsequences1=size(univar2,1);
Numsequences2=size(univar3,1);
Numsequences3=size(univar4,1);
totalsize=Numsequences1+Numsequences2+Numsequences3-2;

fprintf('Number of unique sequences = %g', totalsize);
disp(' ');
dlmwrite('sequence-di3.txt',univar2,'-append','delimiter','\t','precision',2)
dlmwrite('sequence-di3.txt',univar3,'-append','delimiter','\t','precision',2)
dlmwrite('sequence-di3.txt',univar4,'-append','delimiter','\t','precision',2)
        end
```

```
j5=0;
end
end
```