1994

Linear codes and error-correction

Karen Brown
University of Northern Iowa

LINEAR CODING AND ERROR-CORRECTION

KAREN BROWN

Presidential Scholar's Senior Thesis
Presented April 13, 1994
Submitted May 9, 1994

TABLE OF CONTENTS

ABSTRACT

The process of encoding information for transmission from one source to another is a vital process in many areas of science and technology. Whenever coded information is sent, there arises a certain possibility that an error will occur, either during transmisson or in decoding. Therefore, it is imperative to develop methods to detect and correct errors in a code. The study of coding theory is a "new" area of mathematics which is relatively undeveloped.

This paper focuses on the properties of linear codes and their corresponding methods of error-correction. To simplify the issue, only binary block codes are studied; hence all digits are either 0 or 1. The operation of addition is defined over modulo 2. In the decoding process, the principle of "maximum likelihood decoding" is used. This principle assumes that a minimum number of errors will occur in each codeword, since the overall probability of error decreases exponentially with the total number of errors.

Whenever a string of digits is encoded, the digits are multiplied by a generator matrix, which returns the original digits and a specified number of check digits. The check digits are helpful in detecting and correcting errors. The parity check matrix, **H**, is also determined from the generator matrix. If the product of **H** and the transpose of the received word is 0, then the received word is indeed a codeword. If the product is not 0, but is in fact the i^{th} column of **H**, then an error occurs in the i^{th} digit of the received string. The Hamming codes are specific linear codes which contain a maximum number of distinguishable columns. Therefore, the Hamming codes are ideal for error-correction, provided that only one error occurs in each codeword.

It has been speculated that Hadamard matrices are ideal for the coding process, due to the mutual distinguishability of every row and column of the matrix. An Hadamard matrix is a square matrix of order n whose entries are 1 and -1, and which satisfies the equation $HH^T = nI$, where I is the identity matrix of order n. It is known that Hadamard matrices exist only for orders n = 1, n = 2 or $n \equiv 0 \pmod 4$. The rows and columns of the Hadamard matrix are orthogonal and linearly independent, which makes them ideal generator matrices. Hadamard matrices can be constructed using several different methods.

In his paper Hadamard Matrices and Doubly Even Self-Dual Error-Correcting Codes, Michio Ozeki proposed that if the rows of the generator matrix for a binary [n, k] code C all have weights divisible by 4 and are also orthogonal, then C is a doubly even self-dual code. Furthermore, when C is generated by a Hadamard matrix, the result is a doubly even self-dual linear [2n, n] code.

It is now necessary to determine whether two codes will be equivalent if their corresponding Hadamard matrices are equivalent. The remainder of the paper will be devoted finding unique [56, 28] Hadamard codes. It is not known how many different Hadamard matrices exist of order 28. The method of integral equivalence will be used to determine the relationship between two distinct Hadamard matrices. A computer program will generate all of the individual codes.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Professor Tim Hardy for all the help he has given me for the past three years in researching this topic. I would also like to thank the College of Natural Sciences for the research grant I received this year, as well as the University of Northern Iowa for allowing me to be part of the Presidential Scholar's Program. To all those who have supported me throughout the last four years, I am deeply grateful. THANK YOU!

LINEAR CODING AND ERROR-CORRECTION

KAREN S. BROWN

Introduction:

In many areas of science and technology, it becomes necessary to transmit information from one source to another, as in the transfer of data from one computer to another. Such was the case with the photographs taken of Mars by Mariner 9. In order to transmit information, it must first be converted into some type of code, which can easily be decoded by the receiver. Upon transmission, however, the code may possibly be modified due to human or random error. Usually, if an error occurs a code can be re-transmitted, and the error corrected. However, many times messages cannot be sent again. Therefore, it is necessary to determine a process for detecting and correcting errors in a code. In 1950, Robert W. Hamming published a paper on error-correction for linear codes, which pioneered the further study of coding theory. The purpose of this paper will be to study the general properties of codes, discuss simple linear codes and their corresponding methods of error-correction, and analyze a specific type of linear code, namely that generated by Hadamard matrices.

This paper will focus on binary block codes, in which all information is transmitted as a string of zeros and ones. A codeword is such a string of n 0's and 1's, which consists of k ($k<n$) message digits and r ($r = n-k$) check digits. The total number of possible combinations of strings of length n using only 0's and 1's as digits is 2^n . For example, the total number of strings of length 6 is 2^6, or 64. Of these 64 strings, not all are codewords, but only a certain few. Suppose that the previous string contained only 3 message digits and 3 check digits. Since the check digits are determined by the message digits, the actual number of codewords will only be 2^3, or 8. Hence, not all possible strings of

number of codewords will only be 2^3, or 8. Hence, not all possible strings of length 6 are codewords, and therefore a method of error-detection must be found that will determine actual codewords from random strings of digits. It must also be noted that binary coding is defined over addition modulo 2. Hence, $1 + 1 = 0$.

Properties of Codes:

Before studying specific types of codes, a general overview of basic properties of codes and error correction must be reviewed. The Hamming distance - named after R. W. Hamming - is defined as the number of digits that are different between two strings. For example, if d(x,y) is the symbol denoting the Hamming distance between x and y, then:

$$d(011010, 000110) = 3$$

because the second, third and fourth digits differ. In the first string, these digits are 1,1 and 0 respectively; in the second string, they are 0,0, and 1 respectively. The first, fifth and sixth digits in both sets do not differ; they are 0, 1 and 0 respectively in both strings. Therefore, since three digits differ, the Hamming distance is three. This can be rewritten as

$$d(x,y) = \Sigma_{i=1}^{n} |x_i - y_i|,$$

for all strings x and y of length n. This is obvious, because the only time the sum is incremented is if the digits between x and y differ.

The Hamming distance is in fact a metric, in that it satisfies the four basic properties of metrics.

PROOF:

Property 1: $d(x,y) \geq 0$ for all x, y.

Since d(x,y) is defined in terms of absolute value, d(x,y) is always positive.

Property 2: $d(x,y) = 0$ if and only if $x = y$.

If $d(x,y) = 0$, then $x_i = y_i$ for all i. Thus, x must equal y.

Property 3: $d(x,y) = d(y,x)$ for all x,y.

Now $d(x,y) = \Sigma_{i=1}^{n} |x_i - y_i|$. By properties of absolute value, this is equivalent to $\Sigma_{i=1}^{n} |y_i - x_i|$. Therefore, by definition of distance, this becomes $d(y,x)$. Hence, $d(x,y) = d(y,x)$.

Property 4: $d(x,z) \leq d(x,y) + d(y,z)$ for all x,y,z.

$d(x, z) = \Sigma_{i=1}^{n}|x_i - z_i|$

$= \Sigma_{i=1}^{n}|x_i - y_i + y_i - z_i|$

$\leq \Sigma_{i=1}^{n}(|x_i - y_i| + |y_i + z_i|)$ by the Triangle Inequality.

$\leq \Sigma_{i=1}^{n}|x_i - y_i| + \Sigma_{i=1}^{n}|y_i - z_i|$

But then, $\Sigma_{i=1}^{n}|x_i - z_i| \leq \Sigma_{i=1}^{n}|x_i - y_i| + \Sigma_{i=1}^{n}|y_i - z_i|$.

Hence, $d(x,z) \leq d(x,y) + d(y,z)$.

Therefore, since d satisfies all properties of metrics, the Hamming distance is a metric.

Q.E.D.

Now in order to detect an error, the error must convert a codeword to a non-codeword. Therefore, there must be a minimum number of digits that are different between each individual codeword. This is called the minimum Hamming distance, or d. If d = 1, then codewords only differ in one digit, so errors would be impossible to detect. For example, suppose that the digit string 001010 was sent, and the string 001011 was received. If d = 1, then 001011 would also be a codeword, and the error would not be discovered. The greatest number of errors that can be detected in a code is (d-1). This is obvious, because if there are d number of errors in a codeword, then the original word would be received as a different codeword.

Detecting errors and correcting them are two very different matters. Although (d-1) errors can be detected, even fewer can be corrected. The type of decoding used in this paper is called "maximum likelihood decoding," which minimizes the probability of an error occurring. For example, when using strings of length 6, assume that each of the six digits has an equal probability of error p. Suppose p = 0.01, so that the probability of each digit being correct is 0.99. Since addition is defined modulo 2, each digit can be thought of as an independent set of Bernoulli trials. Therefore, the probability of the string being entirely correct would be $\binom{6}{0}(0.99)^6$, or 0.9415. Similarly, the probability for one error would be $\binom{6}{1}(0.01)(0.99)^5$, or 0.057. The probability for two errors would be $\binom{6}{2}(0.01)^2(0.99)^4$, or 0.00144. Since the probability decreased with the number of errors, it is assumed that fewer errors are made. Using a code with two words, namely 000000 and 111111, d = 6, since the minimum Hamming distance between the two "words" is 6. Therefore, up to 5 errors can be detected. However, if the string 010111 is transmitted, errors can be detected, but cannot be corrected, because it cannot be determined which of the codewords was meant to be sent. By using maximum likelihood decoding, it can be assumed that since d(000000, 010111) = 4 and d(111111, 010111) = 2, the string to be sent was 111111.

This "error-correcting" method is called the nearest-neighbor rule. Using this rule, all errors which occur in fewer than (d/2) digits can be corrected. If fewer than (d/2) errors are made, then there is exactly one codeword to which the incorrect word is closest, and to which it can therefore be corrected. If there are (d/2) or more errors, many codewords are of equal distance from the incorrect word. Therefore, the string received cannot be corrected. Using the example above, it is assumed that 010111 has less than (d/2) errors that are detected; it can be corrected to 111111. Suppose instead that 010101 was received. This string has (d/2) or 3 detectable errors. It is clear that this cannot be corrected, since it is of equal distance from both 000000 and 111111. The number of errors which can be corrected from the nearest neighbor rule is:

$t = \lceil (d/2)-1 \rceil$, where $\lceil x \rceil$ is the least integer greater than or equal to x. From this principle comes the following theorem:

<u>THEOREM 1:</u> Suppose that d is the minimum Hamming distance between two codewords in the binary code C. Then no error-detecting rule can detect more than (d -1) errors and no error-correcting rule can correct more than $t = \lceil (d/2) - 1 \rceil$ errors. [13]

<u>PROOF:</u> From the discussion above, it is clear that no error-detecting rule can detect more than (d - 1) errors. However, the error-correction conclusion is a bit more difficult to explain.

Suppose an error-correction rule R exists, which can correct up to (t+1) errors. Let α and β be two codewords, with $d(\alpha, \beta) = d$, or the minimum Hamming distance. Now let γ be a received word with $d(\alpha, \gamma)$ less than or equal to t+1. Also, $d(\beta, \gamma)$ is less than or equal to t+1. Then $R(\gamma)$ must be either α or β. Now without loss of generality, let a codeword be transmitted as α and received as γ. Then this codeword might possibly be corrected to α or β, because there are t+1 or fewer errors. Therefore, the correction rule does not accurately correct errors, and is therefore not valid. Hence, there is no accurate rule that can correct more than $t = \lceil (d/2) - 1 \rceil$ errors.

Q.E.D

Now that the groundwork has been established, the paper can proceed to the discussion of different types of codes and their corresponding error-correction methods.

Types of Codes:

It has already been stated that the check digits are determined by the message digits; therefore there must be some rule for ascertaining what these will be. This is known as the encoding problem. Ideally, an encoding technique should send as many message digits as possible, while subsequently limiting the number of check digits. The information rate of a code is calculated by dividing the number of message digits by the total number of digits in the string. Obviously, a higher information rate is desired.

The first type of codes to be studied are repetition codes. The check digits are simply the message digits repeated a pre-determined number of times. For example, when $k = 1$ and $n = 6$, the two possible codewords would be 000000 and 111111. If $k = 2$ and $n = 6$, then the four possible codewords would be 000000, 010101, 101010, and 111111. These codes could generally be easily corrected using the nearest neighbor rule. However, repetition codes have a very low information rate which will never be greater than one-half.

A type of code which has an extremely high information rate is the single-parity-check code. To find the one parity check digit, the message digit string is added modulo 2, and the parity check digit is given the resulting sum. Hence, the sum of the digits in every codeword is 0. (This can also be done having the sum always equal 1.) Because of this trait, one error is extremely easy to detect, by simply adding the digits in the string. If there are two or any even number of errors, though, the errors would not be detected, and the string might pass for an intentionally transmitted word. Moreover, it would be impossible to find the error, as all of the digits have an equal probability of error. Despite the high information rate, the single-parity-check code has many disadvantages.

The compromising solution is to find a method of encoding which has both a moderate information rate and reasonable level of correctability. Suppose a message digit string of length k is encoded by multiplying it by a matrix, which consists of the (k x k) identity matrix augmented by a (k x (n-k)) matrix to generate parity check digits. This matrix will be known as the generator matrix, or **M**. For example, if 010 is a string of message digits, and **M** is:

$$\begin{bmatrix} 100110 \\ 010011 \\ 001101 \end{bmatrix}$$

the codeword received would be:

$$\begin{bmatrix} 010 \end{bmatrix} \cdot \begin{bmatrix} 100110 \\ 010011 \\ 001101 \end{bmatrix} = \begin{bmatrix} 010011 \end{bmatrix}$$

Obviously, when k message digits are multiplied by a generator matrix, the first k digits of the resulting string are the message digits, because of the presence of the identity matrix in the generator matrix. Hence, to determine the original string, the parity digits only need to be dropped. However, the parity check digits are extremely useful in locating and correcting errors.

<u>**Detecting Errors:**</u>

The generator matrix has already been shown to be $[\mathbf{I_k}\ \mathbf{G}]$ with **G** as an (n x k) matrix. Now let $\mathbf{G^T}$ be the transpose of the matrix **G**. Also, let **H** be called the parity check matrix where **H** is the transpose of **G** augmented by the (n-k) identity matrix, or $[\mathbf{G^T}\ \mathbf{I_{(n-k)}}]$. In the previous example, the transpose of **G** is:

$$\mathbf{G}^T = \begin{bmatrix} 101 \\ 110 \\ 011 \end{bmatrix}$$

which makes the parity check matrix **H**:

$$\mathbf{H} = \begin{bmatrix} 101100 \\ 110010 \\ 011001 \end{bmatrix}$$

From this comes the following definition:

A code is said to be a linear code or a group code if and only if its codewords are the set of vectors **C** which satisfy an equation $\mathbf{HC}^T = \mathbf{0}$. [4]

In fact, the repetition codes and single-parity-check codes are linear codes with corresponding parity check matrices. For the repetition code of length n, **H** = [11....1], where 1 is repeated an n number of times. The single-parity-check codes' parity check matrix is the k x n matrix consisting of a column of 1's augmented by a k x k identity matrix. For example, if n = 4, **H** is:

$$\begin{bmatrix} 1100 \\ 1010 \\ 1001 \end{bmatrix}$$

The parity check matrix can be used to identify codewords, and therefore determine if an error has been made.

THEOREM 2: In a linear code, a block $\mathbf{a} = a_1a_2,,,a_k$ is encoded as $\mathbf{x} = x_1x_2...x_n$ if and only if $a_i = x_i$ for all i less than or equal to k and $\mathbf{Hx}^T = \mathbf{0}$. (It must be noted that **0** is the bit string consisting of all zeros.) [13]

PROOF: By the definition of encoding, **a** is multiplied by the generator matrix to get **x**, so, as explained above, the first k digits of **x** will be the same as the first k digits of **a**. Therefore, $a_i = x_i$ for all i less than or equal to k.

Secondly, **H** times the transpose of **x**, or $\mathbf{Hx}^T$ is equal to $\mathbf{H(a[I_kG])}^T$, by definition.

Now,
$$\mathbf{H(a[I_kG])^T = H[I_kG]^Ta^T}.$$
$$= \mathbf{[G^TI_{(n-k)}]} \begin{bmatrix} I_k \\ G^T \end{bmatrix} \mathbf{a^T}.$$

Since $\mathbf{[G^TI_{(n-k)}]}$ is an (n-k) x (k+(n-k)), or an (n-k) x n matrix, and $\begin{bmatrix} I_k \\ G^T \end{bmatrix}$ is a (k x (n-k)) x k, or an n x k matrix, they can be multiplied together. Since both contain an identity matrix, it can be proven that their product results in $\mathbf{(G^T + G^T)}$. Therefore, $\mathbf{Hx^T} = \mathbf{(G^T + G^T)a^T}$. Since addition is modulo two, then the sum is $\mathbf{0}$, so $\mathbf{Hx^T = 0a^T = 0}$.

Conversely, suppose that $a_i = x_i$ for all i less than or equal to k, and $\mathbf{Hx^T = 0}$. Suppose further that $\mathbf{a}$ is encoded as $\mathbf{y} = y_1y_2 \ldots y_n$. Then as proven above, $a_i = y_i$ for all i less than or equal to k, and further $\mathbf{Hy^T = 0}$. But $\mathbf{Hx^T = 0}$ is also true.

It follows that $\mathbf{x = y}$.

Q.E.D.

Therefore, it has been proven that in order for a string of digits $\mathbf{x}$ to be a codeword, $\mathbf{Hx^T}$ must equal $\mathbf{0}$. Now the syndrome is defined as $\mathbf{s^T = HR^T}$, where $\mathbf{R}$ is a word that has been received. Therefore, $\mathbf{x}$ is a codeword if and only if the syndrome of $\mathbf{x}$ is $\mathbf{0}$. To exemplify this, we again turn to the previous example.

Case 1: Suppose that the string of digits received using the given generator matrix is 000101. This is $\mathbf{x}$. Then the transpose $\mathbf{x^T}$ is:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

When $\mathbf{x^T}$ is multiplied to $\mathbf{H}$, the syndrome is:

$$\begin{bmatrix} 101100 \\ 110010 \\ 011001 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Since this is not equal to **0**, then 000101 must not be a codeword.

Case 2: Now take **x** = 001101. The transpose of **x** is then:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

When multiplied by **H**, the syndrome is **0**, so the string is actually a codeword.

$$\begin{bmatrix} 101100 \\ 110010 \\ 011001 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

In fact, the set of strings of length n is actually an Abelian group with respect to addition, and the set of codewords is a subgroup of that group.

PROOF: (the set of strings of length n is an Abelian group)

(1) Closure:

Let a, b$\in$S (the set of strings of length n).

Since addition is defined over modulo two, a + b is also an element of S.

(2) Associativity:

Since addition is associative in modulo 2, then associativity holds over the elements of S.

(3) Identity:

The identity will be the string of 0's of length n which is an element of S.

(4) Inverse:

In modulo 2 addition, each element will be the inverse of itself.

Hence S is a group.

(5) Commutativity (Abelian group):

Since addition is commutative in modulo 2, S is also commutative.

Hence S is an Abelian group.

PROOF: (the codewords are a subgroup.)

Let C denote the set of all codewords. Then $C \subseteq S$. Now C is associative, since S is associative. Also, the identity element is contained in C. Since each element of C is its own inverse, it suffices to show that C is closed.

Let $a, b \in C$. Then, as a and b are codewords,

$$\mathbf{H}a^{\mathbf{T}} = \mathbf{H}b^{\mathbf{T}} = \mathbf{0}.$$

$$\mathbf{H}a^{\mathbf{T}} - \mathbf{H}b^{\mathbf{T}} = \mathbf{0}$$

$$\mathbf{H}(a^{\mathbf{T}} - b^{\mathbf{T}}) = \mathbf{0}$$

$$\mathbf{H}(a - b)^{\mathbf{T}} = \mathbf{0}$$

By definition, $(a - b) \in C$. Since the subgroup is defined over addition modulo 2, $(a - b) = (a + b)$. Hence $(a + b) \in C$. Therefore, C is a subgroup of S.

Q.E.D.

In fact, the set of strings of length n can be partitioned into cosets with respect to their corresponding syndrome. The coset leader is the string in the group with the fewest number of 1's. The table for strings of length 6 having the generator matrix of the above example is provided below. The coset leaders are the first strings in each row.

TABLE 1:

Message:	000	001	010	100	101	110	011	111	Syndrome:
Code:	000000	001101	010011	100110	101011	110101	011110	111000	$(000)^T$
Coset 1:	100000	101101	110011	000110	001011	010101	111110	011000	$(110)^T$
Coset 2:	010000	011101	000011	110110	111011	100101	001110	101000	$(011)^T$
Coset 3:	001000	000101	011011	101110	100011	111101	010110	110000	$(101)^T$
Coset 4:	000100	001001	010111	100010	101111	110001	011010	111100	$(100)^T$
Coset 5:	000010	001111	010001	100100	101001	110111	011100	111010	$(010)^T$
Coset 6:	000001	001100	010010	100111	101010	110100	011111	111001	$(001)^T$
Coset 7:	001010	000111	011001	101100	100001	111111	010100	110010	$(111)^T$

Correcting an Error:

Now the following theorem can be explained and proven.

THEOREM 3: Suppose that the columns of the parity check matrix **H** are all nonzero and all distinct. Suppose that a codeword **y** is transmitted and **x** is received. If **x** differs from **y** only on the ith digit, then $\mathbf{Hx}^T$ is the ith column of **H**. [13]

PROOF: Since **y** is a codeword, then it follows that $\mathbf{Hy}^T = \mathbf{0}$. Since **x** differs from **y**, then there is a string **e**, such that $\mathbf{x} + \mathbf{y} = \mathbf{e}$. (Every digit in **e** is a zero, except on the digits where **x** and **y** differ.)

Then $\mathbf{Hx}^T = \mathbf{H}(\mathbf{y} + \mathbf{e})^T$

$$= \mathbf{H}(\mathbf{y}^T + \mathbf{e}^T)$$

$$= \mathbf{Hy}^T + \mathbf{He}^T$$

$$= \mathbf{0} + \mathbf{He}^T$$

$$= \mathbf{He}^T$$

Therefore, if exactly one error is made, then when the string is multiplied by the parity check matrix, the result must be one of the columns of the matrix. The number of the column corresponds to the digit which is incorrect in the string.

Q.E.D.

Using Case 1 of the example, since the result was $\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ then that matches with the third column of the matrix. Therefore, the third digit of the original string is incorrect and can be corrected to 001101, which as shown by Case 2 is indeed a codeword. Errors can also be corrected by using the coset table, such as is shown in Table 1. Again using Case 1, 000101 is found in the fourth row of the table; its corresponding coset leader is 001000. This means that one error has occurred, and it is in the third digit. The correct string appears at the top of the column. Although this method might appear easier than multiplying by the parity check matrix, when codes become large, these tables are extremely inefficient.

Suppose on the other hand that $\mathbf{Hx}^{\mathbf{T}}$ is not one of the i^{th} columns of $\mathbf{H}$. In this case, more than one error has occurred, and the string cannot be corrected using this code. Again using the example, suppose instead of receiving 001101, 000111 is received. This obviously has two errors. Then $\mathbf{x}^{\mathbf{T}}$ is:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

and $\mathbf{H}\mathbf{x}^T$ is:

$$\begin{bmatrix} 101100 \\ 110010 \\ 011001 \end{bmatrix} \bullet \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Since $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ is not one of the columns of **H**, then 000111 has more than two errors and cannot be corrected. Referring back to Table 1, the string 000111 has a coset leader of 001010, and there are two errors in the third and fifth digits.

Using the given generator matrix for codewords of length three with three check digits added, the minimum Hamming distance is also three. By Theorem 1, this means that up to $\lceil (d/2) - 1 \rceil$, or one digit can be corrected. Hence, the example satisfies Theorem 1 also. The information rate for the example used is 1/2, since the six digit string only had 3 message digits; however, this number will vary depending upon the size of the generator matrix used.

Hamming Codes:

One standard form of code which facilitates a generator matrix is the Hamming code. In the Hamming codes, there are k message digits and $2^k - 1$ digits in the string. Because of this, each column of the parity check matrix is non-zero and distinct, which means that each syndrome will correspond to exactly one of the columns of **H**. An example of a Hamming code with three message digits is the following:

$$\mathbf{H} = \begin{bmatrix} 1110100 \\ 1101010 \\ 1011001 \end{bmatrix}$$

Clearly, this uses maximum likelihood decoding, since it assumes that only one error occurs and can be corrected. With an individual probability of error of 0.01 for each digit, this code can correct errors in a string of length 7 (3 message digits) with a 97.5% accuracy. However, due to the large number of check digits, this percentage will decrease as the number of message digits increases. The information rate for these codes becomes extremely low as the codes become large.

Hadamard Matrices:

Another type of coding which has received more and more research in the past few years is coding generated by Hadamard matrices. Hadamard matrices are named for the French mathematician Jaques Hadamard (1865 - 1963), who was prominent during the late nineteenth and early twentieth centuries. [10] According to Solomon W. Golomb and Leonard D. Baumert of the Jet Propulsion Laboratory at the California Institute of Technology, "Several years ago, at the Jet Propulsion Laboratory of Caltech, we became interested in the problem of the optimum codes for communicating through space. The rows of an Hadamard matrix form an ideal set of 'code words' for this purpose, because of the high degree of mutual distinguishability (as many disagreements as agreements) between any two such rows." [8] It appears that Hadamard matrices may be the ideal solution to the problems of coding theory.

An Hadamard matrix H is a square matrix of order n whose entries are 1 and -1 and which satisfies the equation $HH^T = nI$, where I is the identity matrix of order n. In order for this to occur, Hadamard matrices only exist for n = 1, n = 2, and $n \equiv 0 (\text{mod } 4)$. It is not known whether Hadamard matrices exist for all multiples of 4. In 1933, R.E.A.C. Paley found Hadamard matrices for all possible orders less than or equal to 200, with the exception of 92, 116, 156, 172, 184, and 188. Since his work was published, they have been found for all orders less than 268. [11] Hadamard matrices have a variety of special properties, which make them perfect for coding. All of the rows and columns of an Hadamard matrix are orthogonal to one another; in other words, the entries in a row or column coincide in exactly half of their positions. An example of an Hadamard matrix of order 4 is the following: (To simplify notation, + will be used for +1, and - will be used for -1.)

$$\begin{bmatrix} + & + & + & + \\ + & + & - & - \\ + & - & + & - \\ + & - & - & + \end{bmatrix}$$

An Hadamard matrix is said to be in "normal form" if it has its first row and first column consisting entirely of +1's. The number of -1's in the remainder of the columns will be n/2, and the number of +1's will be (n/2-1).

The determinant of an Hadamard matrix H is $\pm\ n^{n/2}$. Since the determinant is non-zero, and henceforth the columns are linearly independent, and the columns (or rows) of the matrix span the n-dimensional space determined by the field $\mathbf{F_2}$, the matrix forms a basis for this space. By the self-orthogonality of the rows and columns, the Hadamard matrix is an orthogonal basis for the n-dimensional space determined by $\mathbf{F_2}$. This ensures that the product of an Hadamard matrix with a unique string will also be unique.

Two Hadamard matrices are said to be Hadamard equivalent (or H-equivalent) if one can be formed from the other by (1) exchanging two rows, (2) exchanging two columns, or (3) multiplying some rows or columns by -1. Hadamard-equivalence is in fact an equivalence relation, in that it is reflexive, symmetric and transitive. H-equivalence may also be helpful in determining whether or not two codes are equivalent.

An Hadamard matrix is not uniquely determined by its order. Although Hadamard matrices of orders 1,2,4 and 8 are all H-equivalent, order 16 produces five inequivalent Hadamard matrices, and order 20 yields three. According to Technical Report No. 32-761 from the Jet Propulsion Laboratory, "Certain theoretical considerations make it plausible to expect more classes of Hadamard matrices of order n when $n \equiv 0 \pmod{8}$ than when $n \equiv 4 \pmod{8}$." [9] With the case of 16 and 20, this appears to be true. However, not enough research has been done with higher orders which would provide proof to this hypothesis.

Constructing Hadamard Matrices:

There are several different methods for generating Hadamard matrices, a few of which will be discussed in this paper. Some methods yield Hadamard matrices which are H-equivalent to others, while some produce unique examples. The first method is to derive higher order matrices from smaller ones by using the tensor product (*). The 2 x 2 elementary Hadamard matrix is substituted into an n x n Hadamard matrix for each +1, and the negative of the 2 x 2 for each -1. This will yield a 2n x 2n Hadamard matrix. For example, when substituting the 2 x 2 matrix into a 4 x 4 matrix, the result would be an 8 x 8 Hadamard matrix.

$$\begin{bmatrix} + & + \\ + & - \end{bmatrix} * \begin{bmatrix} + & + & + & + \\ + & + & - & - \\ + & - & + & - \\ + & - & - & + \end{bmatrix} =$$

$$\begin{bmatrix} + & + & + & + & + & + & + & + \\ + & - & + & - & + & - & + & - \\ + & + & + & + & - & - & - & - \\ + & - & + & - & - & + & - & + \\ + & + & - & - & + & + & - & - \\ + & - & - & + & + & - & - & + \\ + & + & - & - & - & - & + & + \\ + & - & - & + & - & + & + & - \end{bmatrix}$$

This tensor product could also be used by substituting 4 x 4 matrices, 8 x 8 matrices, or any order. Thus, the set of Hadamard matrices is closed under the operation of the tensor product.

The Sylvester matrices are formed in a way similar to the tensor product. Given an Hadamard matrix H_n of order n, the Sylvester matrix is the H_{2n} matrix of order 2n formed from the original matrix by the rule

$$H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}$$

This type of construction will produce Hadamard matrices of all powers of 2.

A third type of Hadamard matrix is the Williamson type, which was first used to find Hadamard matrices of order 116, 156 and 172. Williamson discovered that if four symmetric circulant t x t matrices A, B, C, and D can be found, then there will exist an Hadamard matrix of order 4t in the form

$$H = \begin{bmatrix} A & B & C & D \\ -B & A & -D & C \\ -C & D & A & -B \\ -D & -C & B & A \end{bmatrix}$$

provided that $A^2 + B^2 + C^2 + D^2 = 4tI$, where I is the 4t x 4t identity matrix, and $BA - AB + DC - CD = 0$, $CA - AC + BD - DB = 0$, and $DA - AD + CB - BC = 0$. A symmetric circulant matrix is a matrix in which each row is a cyclic permutation of the previous row. An elementary example of the Williamson type is the 12 x 12 Hadamard matrix where

$$A = \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \quad \text{and} \quad B = C = D = \begin{bmatrix} + & - & - \\ - & + & - \\ - & - & + \end{bmatrix}$$

The resulting matrix would be:

$$\begin{bmatrix}
+ & + & + & + & - & - & + & - & - & + & - & - \\
+ & + & + & - & + & - & - & + & - & - & + & - \\
+ & + & + & - & - & + & - & - & + & - & - & + \\
- & + & + & + & + & + & - & + & + & + & - & - \\
+ & - & + & + & + & + & + & - & + & - & + & - \\
+ & + & - & + & + & + & + & + & - & - & - & + \\
- & + & + & + & - & - & + & + & + & - & + & + \\
+ & - & + & - & + & - & + & + & + & + & - & + \\
- & - & + & - & - & + & + & + & + & + & + & - \\
- & + & + & - & + & + & + & - & - & + & + & + \\
+ & - & + & + & - & + & - & + & - & + & + & + \\
+ & + & - & + & + & - & - & - & + & + & + & +
\end{bmatrix}$$

It can also be shown that $A^2 + B^2 + C^2 + D^2$:

$$= \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix}^2 + \begin{bmatrix} + & - & - \\ - & + & - \\ - & - & + \end{bmatrix}^2 + \begin{bmatrix} + & - & - \\ - & + & - \\ - & - & + \end{bmatrix}^2 + \begin{bmatrix} + & - & - \\ - & + & - \\ - & - & + \end{bmatrix}^2$$

$$= \begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix} + \begin{bmatrix} 3 & - & - \\ - & 3 & - \\ - & - & 3 \end{bmatrix} + \begin{bmatrix} 3 & - & - \\ - & 3 & - \\ - & - & 3 \end{bmatrix} + \begin{bmatrix} 3 & - & - \\ - & 3 & - \\ - & - & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 12 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 12 \end{bmatrix}$$

$$= 12 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 12\,I$$

Similarly, A, B, C, and D are commutative in pairs. This Williamson type Hadamard matrix is not in normal form, but can be changed into normal form by applying the same operations used in the section on Hadamard equivalence.

The final method of construction exemplified in this paper is the Paley construction. This method facilitates the use of quadratic residues to form the rows of the Hadamard matrix.

<u>Definition:</u>

> Let p be an odd prime. The nonzero squares modulo p, i.e., the numbers $1^2, 2^2, 3^2, \ldots$ reduced mod p, are called the quadratic residues mod p, or simply the residues mod p. [11]

Those integers which are not quadratic residues are called nonresidues. It is sufficient to consider the integers 1 to (p-1) to find the quadratic residues, since any other integer can be reduced (mod p) to an integer a, such that $0 \leq a \leq (p-1)$. Additionally, let $a \leq (p-1)/2$. Then $(p-a) \geq (p-1)/2$. Using properties of congruence modulo p, $(p-a)^2 \equiv (-a)^2 \equiv a^2 \pmod p$. Hence, it is sufficient to consider only the integers between 1 and (p-1)/2 to find the quadratic residues. For example, let p = 13. The quadratic residues are:

$1^2 = 1$, $2^2 = 4$, $3^2 = 9$, $4^2 = 16$ 3, $5^2 = 25$ 12, $6^2 = 36$ 10.

Hence, the nonresidues are 2,5,6,7,8 and 11.

Three properties of quadratic residues must also be explained.

(Q1) The product of two quadratic residues or of two nonresidues is a quadratic residue, and the product of a quadratic residue and a nonresidue is a nonresidue.

(Q2) If p is of the form 4k + 1, -1 is a quadratic residue mod p. If p is of the form 4k + 3, -1 is a nonresidue mod p.

(Q3) Let p be an odd prime. The function χ, called the Legendre symbol, is defined on the integers by

$\chi(i) = 0$	if i is a multiple of p,
$\chi(i) = 1$	if the remainder when i is divided by p is a quadratic residue mod p, and
$\chi(i) = -1$	if the remainder is a nonresidue.

The Legendre symbol will be used to form the Paley matrices.

The following Theorem must be stated and proved before the Paley matrices can be formed.

<u>THEOREM 4:</u>

For any $c \not\equiv 0 \pmod p$, $\Sigma_{b=0}^{p-1} \chi(b)\chi(b + c) = -1$. [11]

<u>PROOF:</u>

From (Q1), it can be shown that $\chi(xy) = \chi(x)\chi(y)$ for $0 \leq x,y \leq p-1$.

If b = 0, then the sum is not incremented, since zero is a multiple of every number. Therefore, the sum can start with b = 1. Now when $b \neq 0$, then let $z \equiv (b + c)/b \pmod p$, where z is a unique integer between 0 and (p - 1) and varies between all possible values of b where $1 \leq b \leq (p-1)$. Clearly, $z \neq 1$, because then $c \equiv 0 \pmod p$ which contradicts the hypothesis. Then

$$\begin{aligned}\Sigma_{b=0} \chi(b)\chi(b + c) &= \Sigma_{b=1}^{p-1} \chi(b)\chi(bz) \\ &= \Sigma_{b=1}^{p-1} \chi(b)^2\chi(z) \\ &= \Sigma_{\substack{z=0 \\ z \neq 1}}^{p-1} \chi(z) = 0 - \chi(1) = -1.\end{aligned}$$

Q.E.D.

Now the Paley construction can be described. It will yield an Hadamard matrix of order n = p + 1, where p is an odd prime and n is divisible by 4. First, the Jacobsthal matrix $Q = (q_{ij})$ must be formed. This p x p matrix has rows and columns labeled 0, 1,...p-1, where the corresponding entries are determined as follows: $q_{ij} = \chi(i - j)$. An example for the case of p = 7 is given below.

$$\begin{array}{c|ccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
0 & 0 & 1 & 1 & - & 1 & - & - \\
1 & - & 0 & 1 & 1 & - & 1 & - \\
2 & - & - & 0 & 1 & 1 & - & 1 \\
3 & 1 & - & - & 0 & 1 & 1 & - \\
4 & - & 1 & - & - & 0 & 1 & 1 \\
5 & 1 & - & 1 & - & - & 0 & 1 \\
6 & 1 & 1 & - & 1 & - & - & 0
\end{array}$$

Note that Q is skew-symmetric, so $Q^T = -Q$.

<u>LEMMA 1:</u> $QQ^T = pI - J$, and $QJ = JQ = 0$, where J is the matrix all of whose entries are 1. [11]

<u>PROOF:</u>

Let $P = (p_{ij}) = QQ^T$. Then

$$(p_{ii}) = \Sigma_{k=0}^{p-1} q_{ik}{}^2 = p-1,$$

$$(p_{ij}) = \Sigma_{k=0}^{p-1} q_{ik}q_{jk} = \Sigma_{k=0}^{p-1} \chi(k-i)\chi(k-j), \text{ for } i \neq j,$$

$$= \Sigma_{b=0}^{p-1} \chi(b)\chi(b+c), \text{ where } b = k - i \text{ and } c = i - j,$$

$$= -1 \text{ by the previous theorem.}$$

The diagonal of QQ^T consists of the integer $(p - 1)$, and all of the other entries are -1. Hence, $QQ^T = pI - J$. Also, since each row and column of Q contains the same number of positive and negative 1's, $QJ = JQ = 0$.

Q.E.D

Now let

$$H = \begin{pmatrix} 1 & \mathbf{1} \\ \mathbf{1}^T & Q - I \end{pmatrix}.$$

Then

$$HH^T = \begin{pmatrix} 1 & \mathbf{1} \\ \mathbf{1}^T & Q-I \end{pmatrix}\begin{pmatrix} 1 & \mathbf{1} \\ \mathbf{1}^T & Q^T-I \end{pmatrix} = \begin{pmatrix} p+1 & 0 \\ 0 & J + (Q-I)(Q^T-I) \end{pmatrix}$$

From the previous lemma and fact that Q is skew symmetric,

$$J + (Q-I)(Q^T-I) = J + (pI - J) - Q - Q^T + I = (p + 1)I.$$

Therefore, $HH^T = (p + 1)I$. Hence H is a normalized Hadamard matrix of order (p+1). The Hadamard matrix formed is called a Paley matrix. An example of a Paley matrix of order 8 is given below.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & - & 1 & 1 & - & 1 & - & - \\ 1 & - & - & 1 & 1 & - & 1 & - \\ 1 & - & - & - & 1 & 1 & - & 1 \\ 1 & 1 & - & - & - & 1 & 1 & - \\ 1 & - & 1 & - & - & - & 1 & 1 \\ 1 & 1 & - & 1 & - & - & - & 1 \\ 1 & 1 & 1 & - & 1 & - & - & - \end{bmatrix}$$

Now that several different constructions have been explored, the theory of Hadamard codes can be discussed.

Hadamard Codes:

In 1986, Michio Ozeki published a paper in the *Journal of Combinatorial Theory, Series A*, which provided much insight into Hadamard matrices and linear codes. For simplification of his own methods, Ozeki uses Hadamard matrices in which the first entry is -1 while the remaining entries in the first row and column are +1. He calls this normal form, in contradiction with the previous definition provided. From now on, matrices of Ozeki type will be said to be in Ozeki-normal form, while the traditional matrices will be in Klaessy-normal form. A matrix in Klaessy form may be transformed into one of Ozeki form by (*i*) multiplying the first row by -1 and then (*ii*) multiplying all columns except the first by -1. Similarly, an Ozeki-normal Hadamard matrix may be transformed into a Klaessy-normal matrix through the same process.

A code is said to be an [n, k] binary linear code if it is a vector subspace of V_n of dimension k over the field $\mathbf{F_2}$. Essentially, this means that the code will have words of length n with k message digits. Some of the following definitions will prove to be useful later in the paper. The Hamming weight wt(**x**) of the vector **x** is the regular Hamming distance between **x** and **0**, or d(**x**, **0**). A linear code is even if the weight of every element **x** is divisible by 2; if it is divisible by 4, then the code is called doubly even. For any two elements **x** and **y** in a code, the inner product (**x**, **y**) is the following:

$$(\mathbf{x}, \mathbf{y}) = \Sigma_{i=1}^{n} x_i y_i \,.$$

The dual code of the linear code [n, k] is $[n, k]^{\perp}$, where $[n, k]^{\perp} = \{\mathbf{y} \in Vn : (\mathbf{x}, \mathbf{y}) = 0$ for all $\mathbf{x} \in [n, k]\}$. A linear code is self-orthogonal if $[n, k] \subseteq [n, k]^{\perp}$. If $[n, k] = [n, k]^{\perp}$, then the code is said to be self-dual.

An Ozeki-normal Hadamard matrix $NH_n = (s_{ij})$ of order n is of the form

$$NH_n = \begin{bmatrix} -1 & 1 & 1 & 1 \; \ldots\ldots 1 \\ 1 & & & \\ 1 & & * & \\ . & & & \\ 1 & & & \end{bmatrix}$$

Let ξ_i be the ith row vector of NH_n, and let $v_1(i)$ be the number of 1's in the last n -1 entries of ξ_i. Let $v_2(i)$ be the number of -1's in the last n -1 entries of ξ_i. Now $v_1(1)$ is n - 1, since all the entries following the first -1 are 1. Similarly, $v_2(1)$ is 0. Therefore, for each i, $v_1(i) + v_2(i) = n - 1$. Now each of the rows of an Hadamard matrix differ in exactly half of their digits. In comparing any row or column with the first row or column, the first digits in each will differ. Therefore, of the remaining digits in the strings, n/2 of these will be the same, while (n/2 -1) will differ. Since the first row and column consist entirely of 1's, except for the first digit, it follows that there must be n/2 1's and (n/2 -1) -1's. Hence for any row or column i, $v_1(i) = n/2$ and $v_2(i) = (n/2 - 1)$.

Ozeki now forms his type of code. Let $NH_n = (s_{ij})$ be an Ozeki-normalized Hadamard matrix of order n, and let J_n be the square matrix of order n consisting entirely of 1's. The matrix K_n is now formed in the following manner.

$$K_n = 1/2(NH_n + J_n)$$

It follows directly that K_n is a matrix consisting entirely of 0's and 1's. We now let $C_n = (I_n\ K_n)$, so that C_n is an n x 2n matrix, and let $\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n}$ be the row vectors of C_n. From this, the vector subspace $\mathcal{C}(NH_n)$ of the vector space V_{2n} can be formed from the $\mathbf{x_i}$'s over $\mathbf{F_2}$. By the definition of an Hadamard matrix, all of the rows are linearly independent, which makes all of the $\mathbf{x_i}$'s linearly independent, so the dimension of $\mathcal{C}(NH_n)$ is n. The rows $\mathbf{x_i}$ will be denoted as $\mathbf{x_i} = (\mathbf{e_i}, \mathbf{y_i})$, where $\mathbf{e_i}$ is the ith row of the identity matrix and $\mathbf{y_i}$ is the ith row of the converted Hadamard matrix. The weight of each row $\mathbf{x_i}$, $wt(\mathbf{x_i})$, is equal to $1 + wt(\mathbf{y_i})$, since the identity matrix adds only 1 to the weight of each row of K_n. It is clear that $y_{ij} = 1$ if and only if $s_{ij} = 1$ in the original Hadamard matrix. Similarly, $y_{ij} = 0$ if and only if $s_{ij} = -1$. Hence, $wt(\mathbf{x_1}) = n$, since $\mathbf{y_1}$ is $(n - 1)$. Also, $wt(\mathbf{x_i}) = (n/2 + 2)$ for each i, $2<i<n$. This comes from the fact that there are n/2 1's in the last n -1 entries of the Hadamard matrix, a one as the first digit, and a one in the ith digit of the identity matrix.

Since the rows of the identity matrix are themselves self-orthogonal, the inner product $(\mathbf{x_i}, \mathbf{x_h})$ can be defined as

$$(\mathbf{x_i}, \mathbf{x_h}) = \Sigma_{i=1}^{n} y_{ij}y_{hj}.$$

Now for each i, h, $(\mathbf{x_i}, \mathbf{x_h}) = 0$. Trivially, the inner product of any two distinct rows of an identity matrix is zero. Additionally, any two rows of an Hadamard matrix differ in exactly one-half of their digits. If either the ith or hth row is the first row, then it is easy to see that the inner product will be 0. Now suppose that neither the ith or hth row is the first. The number of +1's in the last n -1 digits of each row is n/2. Since the number of digits differ in exactly half of their places, then the number of corresponding +1's in the ith and hth rows is n/4. The sum of the inner product will only be incremented when the +1's have corresponding digits in the two rows. Hence, the inner product will be (n/4 + 1) because the first digits will also correspond. Now it is known that

$n \equiv 4 \pmod 8$. Then $n/4 \equiv 1 \pmod 2$, so $(n/4 + 1) \equiv 0 \pmod 2$. Therefore, in binary addition, the inner product of any two rows will always be zero. From this fact Ozeki provided the following proposition.

PROPOSITION 1:

If the rows of a generator matrix C_n for a binary [n, k] code C have weights divisible by 4 and are orthogonal to each other, then C is self-orthogonal and all weights in C are divisible by 4. [12]

Ozeki proceeded with the next theorem.

THEOREM 5:

Let the notations be as above. When $n \equiv 4 \pmod 8$, then $\mathcal{C}(NH_n)$ is a doubly even self-dual linear [2n,n] code. [12]

PROOF:

There are two parts to this proof. First, it will be shown that $\mathcal{C}(NH_n)$ is doubly even; then it will be shown that it is self-dual.

Part 1: It is given that $n \equiv 4 \pmod 8$. It has been shown that $wt(\mathbf{x}_1) = n$. Also, for all i, $2 \leq i \leq n$, $wt(\mathbf{x}_i) = (n/2 + 2)$. But since $n \equiv 4 \pmod 8$, then $n/2 \equiv 2 \pmod 4$ and furthermore $(n/2 + 2) \equiv 0 \pmod 4$. Hence, for each i, $wt(\mathbf{x}_i)$ is divisible by 4. Therefore $\mathcal{C}(NH_n)$ is doubly even.

Part 2: For any i, h $(1<i,h<n)$, it has been proven that $(\mathbf{x}_i, \mathbf{x}_h) = 0$. Therefore, the rows of the generator matrix are orthogonal to each other. By the aforementioned proposition, this implies that $\mathcal{C}(NH_n)$ is self-orthogonal.

Hence, $\mathcal{C}(NH_n)$ is a doubly even self-dual linear [2n,n] code.

Q.E.D.

Determining Equivalent Codes:

Ozeki now has a basis for constructing special types of linear codes. However, due to the various numbers of Hadamard matrices of different orders, it becomes necessary to find a method of determining whether or not two codes are equivalent. Ozeki makes a broad statement.

THEOREM 6:

We assume that $n \equiv 4 \pmod 8$. Suppose $NH_n^{(1)}$ and $NH_n^{(2)}$ are two normalized and H-equivalent Hadamard matrices of order n; then the codes $\mathcal{C}(NH_n^{(1)})$ and $\mathcal{C}(NH_n^{(2)})$ are equivalent codes. [13]

Ozeki's proof of this theorem is extremely long; unfortunately, his findings are disputed by Vladimir D. Tonchev in a paper in the *Journal of Combinatorial Theory, Series A*. Tonchev states, "An interesting theorem from [Ozeki] states that designs arising from equivalent Hadamard matrices yield equivalent codes. Exploring the concept of a self-orthogonal design, we generalize the construction of self-dual codes based on Hadamard designs to a construction using (0, 1) - Hadamard matrices. The general construction can produce inequivalent codes from equivalent Hadamard matrices." [17]

If Ozeki's theorem is indeed false, then there exists no specific manner for determining whether two matrices are H-equivalent, short of applying all possible permutations of the three operations defined above. This process, although exhaustive, will consume an extreme amount of time and resources. Therefore, it is necessary to find another method which will signify the equivalence or inequivalence of two Hadamard matrices. It has been found that a test for integral equivalence will help differentiate between inequivalent matrices. Two matrices are said to be integrally equivalent if one can be obtained from the other by (1) adding an integer multiple of one row or column to another, (2) negating a row or column, or (3) permuting the rows and/or columns. Since H-equivalence merely implies conditions (2) and (3), then H-equivalent matrices will also be integrally equivalent.

In order to determine whether two matrices are integrally equivalent, the profile of each matrix must be computed. The profile is calculated by finding the absolute value of the generalised inner product of a combination of four distinct rows i, j, k and l, or

$$P_{ijkl} = \left|\Sigma_{x=1}^{n} h_{ix}h_{jx}h_{kx}h_{lx}\right|$$

where h_{ij} is an entry of the Hadamard matrix H of order 4n. It can be shown that $P_{ijkl} \equiv 4n(\text{mod}8)$, since each of the four rows will correspond in exactly half of their digits. The profile of the Hadamard matrix H, or $\pi(m)$, is the number of sets $\{ i, j, k, l \}$ of four distinct rows such that $P_{ijkl} = m$, where $m \equiv 4n(\text{mod } 8)$.

<u>THEOREM 7:</u>

Equivalent Hadamard matrices have the same profile.

<u>PROOF:</u>

It must be shown that the three operations which may be applied to H-equivalent Hadamard matrices will not change the profile of the matrix. Since the profile is determined by computing absolute value, row and column negations do not affect the profile. Also, interchanging columns will clearly not alter the value of the profile. Now suppose that row i of matrix A is exchanged with row r to form matrix B. Then P_{ijkl} of A is P_{rjkl} of B, and P_{rjkl} of A is P_{ijkl} of B.

Therefore, the value of the profile will not be changed, since it is computed from all combinations of four distinct rows.

Q.E.D.

Therefore, if two Hadamard matrices have different profiles, then they are not H-equivalent. However, the converse is not necessarily true; if two Hadamard matrices have the same profile, then they might be H-equivalent or they might not be. The test for integral equivalence is the only "simple" test to determine whether two matrices are inequivalent.

Hadamard Matrices of Order 28:

The remainder of this paper consists of individual data collected from Hadamard matrices of order 28. This number was chosen so as to represent the English alphabet, a blank space, and a period. The only applicable method for construction of 28x28 Hadamard matrices is the Williamson construction. Appendix A contains a computer program written in VAX Pascal which finds all 7x7 symmetric circulant matrices, the results of which are printed in Appendix B. There are 16 possible combinations of 7x7 matrices. The program which finds appropriate matrices to form a Williamson Hadamard matrix is shown in Appendix C. According to the results, there are 52 28x28 Hadamard matrices of the Williamson type, as is shown in Appendix D. From this data, all 52 matrices were computed and tested for integral equivalence. Appendices E and F contain these programs. The results were compatible with those predicted by Baumert and Hall. [2] Of the 52 combinations, there existed exactly two distinct profiles.

TYPE 1:	TYPE 2:
$\pi(4) = 18200$	$\pi(4) = 18032$
$\pi(12) = 2184$	$\pi(12) = 2436$
$\pi(20) = 91$	$\pi(20) = 7$
$\pi(28) = 0$	$\pi(28) = 0$

Exactly 35 of the 52 Hadamard matrices had a profile of TYPE 1; the remaining 17 had a profile of TYPE 2. Therefore, there are at least two distinct inequivalent Hadamard matrices of order 28. Whether more inequivalent Hadamard matrices exist is unknown.

<u>Conclusion:</u>

Binary linear coding can be used in many different areas for a variety of purposes. Due to the availability of methods of error-correction, linear coding is applicable where transmission of information can occur only once, such as in the field of space exploration. Hadamard matrices appear to be the key to a new world of possibilites in the realm of coding theory. Because of the mutual distinguishability of each of the rows and columns, Hadamard matrices are perfect for certain types of error-correction methods. Since so little is known about Hadamard matrices, their study is sure to play a large role in contemporary mathematics.

WORKS CITED

1. Baumert, L.D., *Hadamard Matrices of Orders 116 and 232*, Bulletin of the American Mathematical Society, v. 72:2, (March, 1966), 237.

2. Baumert, L.D., and Hall, Marshall Jr., *Hadamard Matrices of the Williamson Type,* Mathematics of Computation, v. 19, (1965), 442-7.

3. Baumert, L.D. and Hall, Marshall Jr., *A New Construction for Hadamard Matrices*, Bulletin of the American Mathematical Society, v. 71:1, (Jan., 1965), 169-170.

4. Berlekamp, Elwyn R., *Algebraic Coding Theory*, McGraw-Hill Series in Systems Science, McGraw-Hill Book Company, New York, 1968.

5. Burn, Bob, *Messages*, Mathematics Teaching (June, 1979), 52-57.

6. Cooper, Joan, Milas, James and Wallis, W.D., *Hadamard Equivalence*, Lecture Notes in Mathematics, v. 686, (1978), 126 - 135.

7. Goldberg, K., *Hadamard Matrices of Order Cube Plus One*, Proceedings of the American Mathematical Society, v. 17:3, (June, 1966), 744 - 746.

8. Golomb, Solomon W. and Baumert, Leonard D., *The Search for Hadamard Matrices*, The American Mathematical Monthly, v. 70:1, (Jan., 1963), 12 - 17.

9. Hall, Marshall Jr., *Hadamard Matrices of Order 20*, Technical Report No. 32-761, Jet Propulsion Laboratory, California Institute of Technology, National Aeronautics and Space Administration, (Nov. 1, 1965), 1 - 41.

10. Klaessy, Ann, *Hadamard Matrices*, unpublished paper, (April 21, 1993).

11. MacWilliams, F.J. and Sloane, N.J.A., *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, North-Holland Publishing Company, New York, 1977.

12. Ozeki, Michio, *Hadamard Matrices and Doubly Even Self-Dual Error-Correcting Codes*, Journal of Combinatorial Theory, Series A, v. 44, (1987), 274 - 287.

13. Robert, Fred S., *Applied Combinatorics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

14. Rudin, Walter, *Principles of Mathematical Analysis*, McGraw-Hill Book Company, New York, 1964.

15. Schmidt, K.W. and Wang, Edward T.H., *The Weights of Hadamard Matrices*, Journal of Combinatorial Theory, Series A, v. 23, (1977), 257 - 263.

16. Sloane, N.J.A., *Is There a (72,36) d = 16 Self-Dual Code?*, IEEE Transactions of Information Theory, (1973), 251.

17. Tonchev, Vladimir D., *Self-Orthogonal Designs and Extremal Doubly Even Codes*, Journal of Combinatorial Theory, Series A, v. 52, (1989), 197 - 205.

18. Wallis, Jennifer Seberry, *Hadamard Matrices*, Lecture Notes in Mathematics, Combinatorics: Room Squares, Sum-Free Sets, Hadamard Matrices, Springer-Verlag, Heidelberg, 1972.

APPENDIX A

```
program Symmetric(Thesis);
        (*  This program computes the 7x7 symmetric matrices.  *)


const t=7;

type
        Matrix=array[1..t,1..t] of integer;
        Row=array[1..t] of integer;
        List=array[1..20] of Row;

var
        V, A2, A2T, Z:Matrix;
        R:Row;
        I,J,K,L,M,N,P, Counter:integer;
        Sym:List;
        Thesis:text;

(****************************************************************************)



procedure Form(R:Row; var M:Matrix);
        (*  Forms the symmetric circulant matrix.  *)

var Q, I, J:integer;

begin
        for Q:=1 to t do
             M[1,Q]:=R[Q];
        for I:=2 to t do
             for J:=1 to t do begin
                  if (J=1) then
                        M[I,J]:=M[I-1,t]
                  else
                        M[I,J]:=M[I-1,J-1];
             end;
end;

(****************************************************************************)



procedure MMult(X,Y:Matrix; var Z:Matrix);
        (*  Computes the product of two matrices.  *)

var I,J,K:integer;

begin
        for I:=1 to t do
             for J:=1 to t do begin
                  Z[I,J]:=0;
                  for K:=1 to t do
```

```
                    Z[I,J]:=Z[I,J]+X[I,K]*Y[K,J];
          end;
end;

(****************************************************************************)


procedure Transpose(A:Matrix; var AT:Matrix);
      (*  Computes the transpose of a matrix.  *)

var I,J:integer;

begin
      for I:=1 to t do
          for J:=1 to t do
             AT[I,J]:=A[J,I];
end;

(****************************************************************************)


procedure Result(A,B:Matrix; var C:Matrix);
      (*  Subtracts one matrix from anothes.  *)

var I,J:integer;

begin
      for I:=1 to t do
          for J:=1 to t do
              C[I,J]:=A[I,J]-B[I,J];
end;

(****************************************************************************)


function IsZero(M:Matrix):boolean;
      (*  Determines whether a matrix is the zero matrix.  *)

var I,J:integer;
    Temp:boolean;

begin
      Temp:=true;
      I:=1;
      J:=1;
      while Temp and (I<=t) do begin
          while Temp and(j<=t) do begin
              Temp:=(M[I,J] = 0);
              J:=J+1;
          end;
          I:=I+1;
      end;
```

```
        IsZero:=Temp;
end;


(******************************************************************************)


procedure PrintList(L:List; Num:integer);
        (*  Prints the results to the file Thesis.  *)

var I,J:integer;

begin
        open(Thesis, 'Thesis.dat;1', new);
        rewrite(Thesis);
        for I:=1 to Num do begin
             for J:=1 to t do
                  if (L[I][J]=1) then
                      write(Thesis, L[I][J]:3)
                  else
                      write(Thesis, L[I][J]-1:3);
             writeln(Thesis);
        end;
end;


(******************************************************************************)
(******************************************************************************)


(*                              MAIN PROGRAM                                  *)


begin
    Counter:=1;
    for I:=0 to 1 do
         for J:=0 to 1 do
              for K:=0 to 1 do
                   for L:=0 to 1 do
                        for M:=0 to 1 do
                             for N:=0 to 1 do
                                  for P:=0 to 1 do begin
                                       R[1]:=I;
                                       R[2]:=J;
                                       R[3]:=K;
                                       R[4]:=L;
                                       R[5]:=M;
                                       R[6]:=N;
                                       R[7]:=P;
                                       Form(R, V);
                                       MMult(V, V, A2);
                                       Transpose(A2, A2T);
                                       Result(A2, A2T, Z);
                                       if IsZero(Z) then begin
                                            Sym[Counter]:=R;
                                            Counter:=Counter+1;
                                        end;
                                  end;
    PrintList(Sym, Counter-1);
end.
```

APPENDIX B

```
-1 -1 -1 -1 -1 -1 -1
-1 -1 -1  1  1 -1 -1
-1 -1  1 -1 -1  1 -1
-1 -1  1  1  1  1 -1
-1  1 -1 -1 -1 -1  1
-1  1 -1  1  1 -1  1
-1  1  1 -1 -1  1  1
-1  1  1  1  1  1  1
 1 -1 -1 -1 -1 -1 -1
 1 -1 -1  1  1 -1 -1
 1 -1  1 -1 -1  1 -1
 1 -1  1  1  1  1 -1
 1  1 -1 -1 -1 -1  1
 1  1 -1  1  1 -1  1
 1  1  1 -1 -1  1  1
 1  1  1  1  1  1  1
```

APPENDIX C

```
program Williamson(Data, output);

const
        t = 7;
        Num = 16;

type
        Matrix = array[1..t, 1..t] of integer;
        Matrixrow = array[1..t] of integer;
        Holder = array[1..20] of Matrixrow;
        Storer = record
             A, B, C, D:integer;
        end;
        Resulttype = array[1..100] of Storer;


var ABCD: Holder;
    Results:Resulttype;
    HMN: integer;
    Data:text;


(***************************************************************************)


procedure SkipBlanks(var F:text);
        (*  Skips blanks in the data, so that eof won't be read.  *)

var Finished:boolean;

begin
        Finished:=false;
        repeat
             if eof(F) then
                Finished:=true
             else if F^ = ' ' then
                get(F)
             else
                Finished:=true;
        until Finished;
end;

(***************************************************************************)


procedure Form(Row:Matrixrow; var S:Matrix);
        (* Forms the 7x7 symmetric circulant matrices.  *)

var I, J, K, L:integer;

begin
```

```
        for J:=1 to t do
                S[1,J]:=Row[J];
        for K:=2 to t do
             for L:= 1 to t do
                 if L-1<>0 then
                       S[K,L]:=S[K-1,L-1]
                  else
                       S[K,L]:=S[K-1, t];
end;


(**************************************************************************)


procedure Sum (A, B, C, D:Matrix;  var S:Matrix);
        (*  Computes the sum of four matrices.  *)

var G,H:integer;

begin
        for G:=1 to t do
             for H:=1 to t do
                  S[G,H]:=A[G,H] + B[G,H] + C[G,H] +D[G,H];
end;

(**************************************************************************)


procedure MMult(A,B:matrix;  var C:Matrix);
        (*  Computes the product of two matrices.  *)

var I,J,K: integer;

begin
        for I:=1 to t do
             for J:=1 to t do begin
                  C[I,J]:=0;
                  for K:=1 to t do
                        C[I,J]:=C[I,J] +A[I,K] * B[K,J];
             end;
end;

(**************************************************************************)


procedure Sum2(W, X, Y, Z:Matrix;  var Res:Matrix);
        (*  Computes the sum of four matrices, with two negations.  *)

var I, J:integer;
```

```
begin
        for I:=1 to t do
             for J:=1 to t do
                  Res[I,J]:=W[I,J] - X[I,J] + Y[I,J] - Z[I,J];
end;


(****************************************************************************)


function IsZero(M:Matrix):boolean;
        (*  Determines whether a matrix is the zero matrix.   *)

var I,J:integer;
    Temp:boolean;

begin
        Temp:=true;
        I:=1;
        J:=1;
        while Temp and (I<=t) do begin
             while Temp and (J<=t) do begin
                  Temp:=(M[I,J]=0);
                  J:=J+1;
             end;
             I:=I+1;
        end;
        IsZero:=Temp;
end;

(****************************************************************************)


function Check(Temp:Matrix):boolean;
     (*  Determines whether the matrix is a multiple of the identity matrix. *)

var I,J:integer;
    Same:boolean;

begin
        Same:=true;
        I:=1;
        J:=1;
        while Same and (I<=t) do begin
              while Same and (J<=t) do begin
                  if (I=J) then
                       Same:=(Temp[I,J] = 4*t)
                  else
                       Same:=(Temp[I,J] = 0);
                  J:=J+1;
             end;
             I:=I+1;
        end;
        Check:=Same;
end;

(*****************************************************************************)


procedure PrintMat(M:Matrix);
        (*  Prints a 7x7 matrix.  (This was used as a check.)  *)

var Q,S:integer;

begin
        for Q:=1 to t do begin
             for S:= 1 to t do
                  write(M[Q,S]:3);
             writeln;
        end;
        writeln;
end;

(***********************************************************************)


procedure ReadMatrices(var Data:text;  var ABCD:Holder);
        (*  Reads and stores the first row of each matrix.  *)

var Counter, K, L, J, A:integer;

begin
        reset(Data);
        Counter:=1;
        SkipBlanks(Data);
        while not eof(Data) do begin
             for J:=1 to 7 do begin
                  read(Data, A);
                  ABCD[Counter][J]:=A;
                  Skipblanks(Data);
             end;
             Counter:=Counter+1;
             Skipblanks(Data);
        end;
end;

(***********************************************************************)


procedure Compute(var Results:Resulttype; ABCD:Holder;  var HMN:integer);
        (*  Determines whether a set of four matrices satisfy  *)
        (*  the qualifications for a Williamson Hadamard matrix.  *)
```

APPENDIX C

```
var Temp, A2, B2, C2, D2:Matrix;
    A, B, C, D:Matrix;
    One, Two, Three, Four, Com1, Com2, Com3:Matrix;
    Counter, I, J, K, L:integer;


begin
    Counter:=1;
    for I:=1 to Num do
         for J:=I to Num do
              for K:=J to Num do
                   for L:=K to Num do begin
                        Form(ABCD[I], A);
                        Form(ABCD[J], B);
                        Form(ABCD[K], C);
                        Form(ABCD[L], D);
                        MMult(A, A, A2);
                        MMult(B, B, B2);
                        MMult(C, C, C2);
                        MMult(D, D, D2);
                        Sum(A2, B2, C2, D2, Temp);
                   (*   PrintMat(Temp); *)
                        if Check(Temp) then begin
                   (*   Checks whether A2+B2+C2+D2=4tI   *)

                             MMult(B, A, One);
                             MMult(A, B, Two);
                             MMult(D, C, Three);
                             MMult(C, D, Four);
                             Sum2(One, Two, Three, Four, Com1);
                             if IsZero(Com1) then begin
                   (*  Checks to see if BA-AB+DC-CD=0.  *)

                                MMult(C, A, One);
                                MMult(A, C, Two);
                                MMult(B, D, Three);
                                MMult(D, B, Four);
                                Sum2(One, Two, Three, Four, Com2);
                                if IsZero(Com2) then begin
                   (*  Checks to see if CA-AC+BD-DB=0.  *)

                                   MMult(D, A, One);
                                   MMult(A, D, Two);
                                   MMult(C, B, Three);
                                   MMult(B, C, Four);
                                   Sum2(One, Two, Three, Four, Com3);
                                   if  IsZero(Com3) then begin
                   (*  Checks to see if DA-AD+CB-BC=0.  *)

                                        Results[Counter].A:=I;
                                        Results[Counter].B:=J;
                                        Results[Counter].C:=K;
                                        Results[Counter].D:=L;
                                        Counter:=Counter+1;
                                end;
```

APPENDIX D

```
The number of Williamson Hadamard matrices is  52.


The sets of four matrices are:
A =  2        B =  2        C =  3        D =  6
A =  2        B =  2        C =  3        D = 11
A =  2        B =  2        C =  6        D = 14
A =  2        B =  2        C = 11        D = 14
A =  2        B =  3        C =  6        D = 15
A =  2        B =  3        C = 11        D = 15
A =  2        B =  5        C =  5        D =  7
A =  2        B =  5        C =  5        D = 10
A =  2        B =  5        C =  7        D = 12
A =  2        B =  5        C = 10        D = 12
A =  2        B =  6        C = 14        D = 15
A =  2        B =  7        C = 12        D = 12
A =  2        B = 10        C = 12        D = 12
A =  2        B = 11        C = 14        D = 15
A =  3        B =  3        C =  4        D =  5
A =  3        B =  3        C =  4        D = 12
A =  3        B =  3        C =  5        D = 13
A =  3        B =  3        C = 12        D = 13
A =  3        B =  4        C =  5        D = 14
A =  3        B =  4        C = 12        D = 14
A =  3        B =  5        C = 13        D = 14
A =  3        B =  6        C = 15        D = 15
A =  3        B = 11        C = 15        D = 15
A =  3        B = 12        C = 13        D = 14
A =  4        B =  5        C = 14        D = 14
A =  4        B =  6        C =  7        D =  8
A =  4        B =  6        C =  7        D =  9
A =  4        B =  6        C =  8        D = 10
A =  4        B =  6        C =  9        D = 10
A =  4        B =  7        C =  8        D = 11
A =  4        B =  7        C =  9        D = 11
A =  4        B =  8        C = 10        D = 11
A =  4        B =  9        C = 10        D = 11
A =  4        B = 12        C = 14        D = 14
A =  5        B =  5        C =  7        D = 15
A =  5        B =  5        C = 10        D = 15
A =  5        B =  7        C = 12        D = 15
A =  5        B = 10        C = 12        D = 15
A =  5        B = 13        C = 14        D = 14
A =  6        B =  7        C =  8        D = 13
A =  6        B =  7        C =  9        D = 13
A =  6        B =  8        C = 10        D = 13
A =  6        B =  9        C = 10        D = 13
A =  6        B = 14        C = 15        D = 15
A =  7        B =  8        C = 11        D = 13
A =  7        B =  9        C = 11        D = 13
A =  7        B = 12        C = 12        D = 15
A =  8        B = 10        C = 11        D = 13
A =  9        B = 10        C = 11        D = 13
A = 10        B = 12        C = 12        D = 15
A = 11        B = 14        C = 15        D = 15
A = 12        B = 13        C = 14        D = 14
```

APPENDIX E

```
program ComputeHadamard(Data, Thesis);
        (*  This program form the Williamson type Hadamard matrix.  *)

const t=7;

type HMat = array[1..4*t, 1..4*t] of integer;
     Matrix = array[1..t, 1..t] of integer;
     Sorter = array[1..4] of Matrix;

var
        ABCD:Sorter;
        Hadamard:HMat;
        Data, Thesis:text;


(***************************************************************************)


procedure ReadMat(var Data:text;  var H:Sorter);
        (*  Reads and stores four 7x7 symmetric circulant matrices.  *)

var I, J, Counter:integer;

begin
        for Counter:=1 to 4 do begin
             for I:=1 to t do begin
                  for J:=1 to t do
                         read(Data, H[Counter][I,J]);
                  readln(Data);
             end;
        end;
end;


(***************************************************************************)


procedure Compute(A:Sorter; var H:HMat);
        (*  Computes the Hadamard matrix.  *)

var I, J, K, L, M:integer;

begin
        for I:=1 to t do begin
             for J:=1 to t do
                  H[I,J]:=A[1][I,J];
             for K:=(t+1) to 2*t do
                  H[I,K]:=A[2][I, K-t];
             for L:=(2*t+1) to 3*t do
                  H[I,L]:=A[3][I, L-2*t];
             for M:=(3*t+1) to 4*t do
                  H[I,M]:=A[4][I,M-3*t];
             end;
        for I:=t+1 to 2*t do begin
             for J:=1 to t do
```

```
                H[I,J]:=-(A[2][I-t,J]);
            for K:=(t+1) to 2*t do
                H[I,K]:=A[1][I-t, K-t];
            for L:=(2*t+1) to 3*t do
                H[I,L]:=-(A[4][I-t, L-2*t]);
            for M:=(3*t+1) to 4*t do
                H[I,M]:=A[3][I-t,M-3*t];
            end;
        for I:=(2*t+1) to 3*t do begin
            for J:=1 to t do
                H[I,J]:=-(A[3][I-2*t,J]);
            for K:=(t+1) to 2*t do
                H[I,K]:=A[4][I-2*t, K-t];
            for L:=(2*t+1) to 3*t do
                H[I,L]:=A[1][I-2*t, L-2*t];
            for M:=(3*t+1) to 4*t do
                H[I,M]:=-(A[2][I-2*t, M-3*t]);
            end;
        for I:=(3*t+1) to 4*t do begin
            for J:=1 to t do
                H[I,J]:=-(A[4][I-3*t,J]);
            for K:=(t+1) to 2*t do
                H[I,K]:=-(A[3][I-3*t, K-t]);
            for L:=(2*t+1) to 3*t do
                H[I,L]:=A[2][I-3*t, L-2*t];
            for M:=(3*t+1) to 4*t do
                H[I,M]:=A[1][I-3*t,M-3*t];
            end;
end;

(***************************************************************************)


procedure Print(H:HMat; var Thesis:text);
        (*  Prints the Hadamard matrix to a specified file.  *)

var I, J:integer;

begin
        for I:=1 to 4*t do begin
            for J:=1 to 4*t do
                write(Thesis, H[I,J]:3);
            writeln(Thesis);
        end;
end;

(***************************************************************************)
(***************************************************************************)


(*                              MAIN PROGRAM                               *)
```

```
begin
        open(Data, 'Data.Dat;1', old);
        reset(Data);
        ReadMat(Data, ABCD);
        Compute(ABCD, Hadamard);
        open(Thesis, 'HM.dat;1', new);
        rewrite(Thesis);
        Print(Hadamard, Thesis);
end.
```

APPENDIX F

```
program integral(Data, output);
        (*  Computes the profile of a 28x28 Hadamard matrix.  *)

const t = 7;

type Matrix = array[1..4*t, 1..4*t] of integer;
     Profile = array[0..10] of integer;

var HMat:Matrix;
    Pi:Profile;
    Data:text;

(*****************************************************************************)


procedure ReadMatrix(var M:Matrix);
        (*  Reads in a 28x28 Hadamard matrix.  *)

var I, J: integer;

begin
        for I:=1 to 4*t do begin
             for J:= 1 to 4*t do
                  read(Data, M[I, J]);
             readln(Data);
        end;
end;

(*****************************************************************************)


procedure InitializeProfile(var Prof:Profile);
        (*  Initializes the profile to zero.  *)

var Num, I: integer;

begin
        Num:=(4*t)div 8;
        for I:=0 to Num do
             Prof[I]:=0;
end;

(*****************************************************************************)


procedure ComputeProfile(M:Matrix;  var Prof:Profile);
        (*  Computes the profile of the matrix.  *)

var I,J,K,L:integer;
    X:integer;
    P, Temp:integer;
```

```
begin
    InitializeProfile(Prof);
    for I:=1 to (4*t-3) do
        for J:=(I+1) to (4*t-2) do
             for K:=(J+1) to (4*t-1) do
                  for L:=(K+1) to 4*t do begin
                        P:=0;
                        for X:= 1 to 4*t do begin
                             Temp:=M[I,X]*M[J,X]*M[K,X]*M[L,X];
                             P:=P + Temp;
                        end;
                        P:=abs(P);
                        Prof[P div 8]:=Prof[P div 8] + 1;
                  end;
end;

(****************************************************************************)


procedure PrintProfile(Prof:Profile);
        (*  Prints the profile to the screen.  *)

var Num, Q, Count:integer;

begin
        Num:=(4*t) div 8;
        if (((4*t) mod 8) = 0)  then
             Count:=0
        else
              Count:=4;
        for Q:=0 to Num do begin
             writeln('Pi(', Count:2, ') = ', Prof[Q]:7, '.');
             Count:=Count+8;
        end;
end;

(******************************************************************************)
(******************************************************************************)


(*                              MAIN PROGRAM                                 *)


begin
        open(Data, 'HM.dat;1', old);
        reset(Data);
        ReadMatrix(HMat);
        ComputeProfile(HMat, Pi);
        PrintProfile(Pi);
end.
```