Jennifer Schulz and Alysse Haferman
Abstract Algebra
Professor Westlund
December 10, 2012

LINEAR CODES

Linear codes are used frequently in many applications by today's computing society. They are primarily used to correct and detect errors when digital data needs to be transmitted. This paper will attempt to present a brief overview of linear codes, providing various definitions, proofs and examples, and then will delve into a more specific linear code, the Hamming code. Please note that our primary source is *Contemporary Abstract Algebra* by Gallian [1], and so the majority of the information within this paper stems from this source.

We begin with a definition of a linear code:

> Definition [1]: An (n, k) **linear code** over a finite field F is a k-dimensional subspace V of the vector space $F^n = F \oplus F \oplus \cdots \oplus F$ (where there are n copies of F) over F. The members of V are called the **code words**.

Since F is a finite field, we can think of F as $\mathbb{Z}_p$ where p is prime. When F is $\mathbb{Z}_2$, the linear code is called binary, and when F is $\mathbb{Z}_3$, the linear code is called ternary.

We note that the vectors that make up the k-dimensional subspace V form a basis of the subspace V, such that if $\vec{v_i}$ is a basis vector, then $V = Span\{\vec{v_1}, \vec{v_2}, \cdots, \vec{v_k}\}$. Furthermore, if we let S be a nonempty subset of V such that $S = \{\vec{v_1}, \cdots, \vec{v_k}\}$, we can say the span of $S = \langle S \rangle = \{\lambda_1\vec{v_1} + \cdots + \lambda_k\vec{v_k} \mid \lambda_i \in F\}$ [2]. Each vector (code word) in V is the result of a linear combination of the basis vectors spanning V.

The vectors in V are n-tuples consisting of two parts: the message part, which is k digits long, and the redundancy part, which is $n - k$ digits long. As a result, the notation (n, k) represents the length of the code words and the length of the message inside the code word. Thus, if we have the vector $(x_1, x_2, ..., x_k, x_{k+1}, ..., x_n)$, then $(x_1, ..., x_k)$ constitutes our message part, while $(x_{k+1}, ..., x_n)$ is the redundancy part. Because the message part of the code is so obvious, linear codes are not used for security purposes, but rather for error detection and correction.

We note that we are also able to determine the number of code words present in a linear code by use of the expression q^k, where q is the order of the finite field F and k is the length of the messages we want to encode. This gives us the total number of legitimate, sent code words because there are q possible digits for each of k positions in the message part of our code word. On the other hand, we are also able to determine the total number of code words (including those with errors) in the linear code by use of the expression q^n, where n is the length of the code word. This gives us the total number of possible received code words because there are q possible digits for each of the n positions, since any of the n spots is prone to have errors.

Consider the linear code consisting of code words:

$$0000000, 1110101, 0111001, 0010011, 1001100, 0101010, 1011111, 1100110.$$

We want to write this code using the notation (n, k). Using the above terminology, we note that because there are eight code words in this linear code, then $q^k = 8$. In addition, we see that $|F| = 2$, since our code words only consist of zeros and ones. Thus, $q = 2$ and so, $2^k = 8 \implies k = 3$. Since there are seven digits in each code word, we see that $n = 7$. As a result, we can write this linear code as $(7, 3)$. This tells us that our messages are three digits long, while our redundancy term is four digits long.

Now we look at a few more important definitions:

> Definition [8]: The **code rate** (or efficiency) of a linear code is $R = \frac{k}{n}$, for k, n defined above.

The code rate helps us determine the dimensions that our code needs to be in order for us to have the optimal efficiency. Ultimately, we desire to find the shortest length of n for code words that still allows us to detect/correct the desired number of errors. The closer R is to 1, the better the efficiency of our code.

> Definition [1]: The **Hamming distance** between two vectors in F^n is the number of components in which they differ.

For example, if we have two vectors, $\vec{s}$ and $\vec{t}$, where $\vec{s} = 0010110$ and $\vec{t} = 1001011$, then to find the distance between them, we simply count up the number of digits in which they are different. So in this case, $d(\vec{s}, \vec{t}) = 5$.

> Definition [1]: The **Hamming weight** of a vector is the number of nonzero components of the vector. The **Hamming weight** of a linear code is the minimum weight of any nonzero vector in the code.

Using our previous example with $\vec{s}$ and $\vec{t}$ defined above, $wt(\vec{s}) = 3$, because that is the number of nonzero digits in the vector. Similarly, $wt(\vec{t}) = 4$. Thus, we see that we are able to find the weight of a single vector. In order to find the weight of an entire linear code, you need to take the minimum of all the weights of the vectors in the linear code (excluding the zero vector). If we take the $(7, 3)$ linear code from above, we note that the weights of the code words are 3, 4, 5, or 6. Since $\min(3, 4, 5, 6) = 3$, then 3 is the weight of this linear code.

Now we will explore some properties of linear codes.

> Properties [1]: For any vectors $\vec{u}$, $\vec{v}$, and $\vec{w}$:
> (1) $d(\vec{u}, \vec{v}) \leq d(\vec{u}, \vec{w}) + d(\vec{w}, \vec{v})$
> (2) $d(\vec{u}, \vec{v}) = wt(\vec{u} - \vec{v})$.

Proof for (1): Note that if $\vec{u}$ and $\vec{v}$ are the same in the i^{th} position, then $d(u_i, v_i) = 0$ at that position, which is certainly less than or equal to whatever is on the right side of the equation (since distances are always ≥ 0). On the other hand, if $\vec{u}$ and $\vec{v}$ are different in the i^{th} position, and if $\vec{u}$ and $\vec{w}$ are the same in that i^{th} position, then $\vec{w}$ and $\vec{v}$ must differ in that position, so $1 \leq 0 + 1$ at the i^{th} position. Thus, by summing the distances at each position, we see that $d(\vec{u}, \vec{v}) \leq d(\vec{u}, \vec{w}) + d(\vec{w}, \vec{v})$.

Proof for (2): It is clear that both $d(\vec{u}, \vec{v})$ and $wt(\vec{u} - \vec{v})$ denote the number of positions in which $\vec{u}$ and$\vec{v}$ differ.

In order to see these properties in action, consider the following vectors: $\vec{u} = 1001011$, $\vec{v} = 0111101$, and $\vec{w} = 1110010$. We see $d(\vec{u}, \vec{v}) = 5$, $d(\vec{u}, \vec{w}) = 4$, and $d(\vec{w}, \vec{v}) = 5$, and so $d(\vec{u}, \vec{v}) \leq d(\vec{u}, \vec{w}) + d(\vec{w}, \vec{v}) \implies 5 \leq 4 + 5 = 9$. Thus, Property (1) holds true. Similarly, we see $wt(\vec{u} - \vec{v}) = wt(1110110) = 5$. Thus, $d(\vec{u}, \vec{v}) = 5 = wt(\vec{u} - \vec{v})$, so Property (2) also holds true.

These definitions and properties are important because they help us determine the number of errors that can be corrected and/or detected by our code.

> Theorem [1] (Correcting Capability of a Linear Code): If the Hamming weight of a linear code is at least $2t + 1$, then the code can *correct* any t or fewer errors. Alternatively, the same code can *detect* any $2t$ or fewer errors.

In order to prove this theorem, we will use a principle called the Nearest Neighbor Decoding Theory. In order to see how this theory applied, suppose that we have a sent vector $\vec{v}$ and a received vector $\vec{v'}$, where the sent vector is the original code word and the received vector is prone to error due to the transmission process. If we know that at least one error occured in transmission and we want to correct this error, we find $d(\vec{v'}, \vec{x})\ \forall \vec{x}$ where $\vec{x}$ is a code word. If $d(\vec{v'}, \vec{x})$ is a minimum for a specific $\vec{x}$, then we know, by the Nearest Neighbor Decoding Theory, that we can correct our received vector $\vec{v'}$ to $\vec{x}$. In our supposed case, $d(\vec{v}, \vec{v'})$ should give us a minimum value since it is the received vector's original vector, and so should result in a correction of our received vector $\vec{v'}$ to the original sent vector $\vec{v}$. Knowing this principle, we next look at a proof of this theorem.

Proof: Assume that the Hamming weight of a linear code is at least $2t+1$. We must show that the code will correct any t or fewer errors or detect any $2t$ or fewer errors. We suppose that we have a sent vector $\vec{u}$ and a received vector $\vec{v}$. We also suppose that $d(\vec{u}, \vec{v}) \leq t$ (that at most t errors occurred in transmission). We assume that $\vec{w}$ is any code word other than $\vec{u}$ such that $\vec{w} - \vec{u}$ is a nonzero code word. We will prove this process using the Nearest Neighbor Decoding Theory, as explained above. We see that

$$2t + 1 \leq wt(\vec{w} - \vec{u}) = d(\vec{w}, \vec{u}) \leq d(\vec{w}, \vec{v}) + d(\vec{v}, \vec{u}) \leq d(\vec{w}, \vec{v}) + t.$$

Thus, we see that $2t + 1 \leq d(\vec{w}, \vec{v}) + t$, and consequently $t + 1 \leq d(\vec{w}, \vec{v})$. As a result, since $d(\vec{u}, \vec{v}) \leq t$ and $d(\vec{w}, \vec{v}) \geq t+1$, by the Nearest Neighbor Decoding Theory, $\vec{v}$ can be correctly corrected to $\vec{u}$. Therefore, we see that whenever the Hamming weight of a linear code is at least $2t + 1$, the code is able to correct t or fewer errors.

To prove the second part of the theorem, we suppose the same information as above. In addition, we also suppose that $2t \geq d(\vec{u}, \vec{v}) \geq 1$ (that at least one error and at most $2t$ errors were made in transmission). However, we see that in order for two code words in this linear code to be distinct, they must have a Hamming weight of at least $2t + 1$. Since $d(\vec{u}, \vec{v}) \leq 2t < 2t+1$, we know that $\vec{u}$ and $\vec{v}$ are not distinct code words. Instead, this shows that an error was made in transmission, implying that $\vec{v}$ needs to be corrected to its original vector. As a result, we see that whenever the Hamming weight of a linear code is at least

$2t+1$, the code is able to detect $2t$ or fewer errors.

In order to illustrate this theorem, we see that if we use the $(7,3)$ linear code from above, since the Hamming weight of the code is 3, $2t+1=3 \implies t=1$. This means that we can either correct one error or detect two or fewer errors.

It is important to note that this theorem can be used to either correct *or* detect errors. It cannot be used to do both at the same time. As a result, based on the size of the code and the efficiency resulting from the use of either process, you have to choose which process you will use in running the code – if you will use it to only correct errors, or if you will use it to only detect errors.

As a side note, however, if we write the Hamming weight of a linear code in the form $2t+s+1$, we can correct any t *and* detect any $t+s$ or fewer errors.

We see then that if we have a linear code with Hamming weight 5 and we want to correct and/or detect errors, we can use one of the following procedures:
Based on the first method ($2t+1=5$):
1. Correct 2 or fewer errors ($t=2$)
2. Detect 4 or fewer errors ($t=2$)

Based on the second method ($2t+s+1=5$):
1. Correct 2 or fewer AND detect 2 or fewer errors (with t=2 and s=0)
2. Correct 1 AND detect 3 or fewer errors (with t=1 and s=2)
3. Correct 0 AND detect 4 or fewer errors (with t=0 and s=4)

Thus, we see that we are able to define and create linear codes in such a way as to have the ability to correct and/or detect any desired number of errors.

Building on what we have covered so far, we now look at a few more important definitions:

Definition [3]: A **sphere of radius *r*** about $\vec{x} \in F_q^n$ is the set
$$S_r(\vec{x}) = \{\vec{y} \in F_q^n \ \mid \ d(\vec{x},\vec{y}) \le r\}.$$

In other words, a sphere has a unique, correct code word $\vec{x}$ at its center. The other code words that are enclosed in the sphere are those received code words (stemming from the sent code word at the center) that have a maximum of r errors in transmission, such that $d(\vec{x},\vec{y}) \le r$ [3]. As a result, based on the assumption above, the distance between each vector and its respective code word is $\le r$ [4]. The definition of a sphere allows us to better understand the definition of a Hamming bound:

Definition [3]: The **Hamming bound** of a t-error-correcting code C over F_q of length n with M codewords is
$$M(1+(q-1)\binom{n}{1}+\cdots+(q-1)^t\binom{n}{t}) \le q^n.$$

This inequality shows the maximum number of possible code words in a code with certain parameters. The summation expression of the inequality $(1+(q-1)\binom{n}{1}+\cdots+(q-1)^t\binom{n}{t})$ represents the total number of vectors (code words) contained in each sphere. The greater the capacity for error correction, the more code words will be included in each sphere. Since there are M possible codewords, the total number of codewords when adding up all the codewords in every disjoint sphere is $M(1+(q-1)\binom{n}{1}+\cdots+(q-1)^t\binom{n}{t})$. Since there must be q^n possible received code words in a code, then the total number of code words included in every disjoint sphere must be $\leq q^n$. If, on the other hand, there is equality in the above equation, we call the code **perfect** [3]. This is another attribute that we can observe in linear codes in order to better classify and categorize them.

Now that we know how to determine various attributes and qualities in linear codes, we move on to the generation and creation process of such codes. In order to encode our messages, we use generator matrices [1]; these matrices build in the redundancy part into our code words. Generator matrices are of the form:

$$[I|A] = \begin{bmatrix} 1 & 0 & \cdots & 0 & a_{1,1} & \cdots & a_{1,n-k} \\ 0 & 1 & \cdots & 0 & a_{2,1} & \cdots & a_{2,n-k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & a_{k,1} & \cdots & a_{k,n-k} \end{bmatrix}$$

where $a_{i,j} \in F$. So I is the $k \times k$ identity matrix, and A is a $k \times (n-k)$ matrix, altogether giving us a $k \times n$ generator matrix. Remembering back to the definition of linear code, we remember that there were k vectors in the k-dimensional subspace V that could form a basis spanning all of V, such that every code word was a linear combination of those k basis vectors. We see that our encoding process incorporates just such a principle. In order to encode our messages (of length k), we multiply each message vector by our generator matrix on the right and get our code words of length n. Multiplying by the identity part of the matrix keeps our original message intact, while multiplying by the A part builds the redundacy onto the end of the message. We note that this means that the rows of the generator matrix form the basis vectors of the k-dimensional subspace V that contains all the code words of the linear code.

Suppose for a $(6,3)$ binary linear code we have the generator matrix $G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$. If we are trying to encode the message 010, we perform the multiplication

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

Thus, our code word corresponding to our message is 010101 and this is the vector that we would transmit.

Among the many different methods of decoding linear codes, one method to decode received vectors is to use a parity-check matrix, which is related to the generator matrix. If we have a generator matrix $G = \begin{bmatrix} I_k | A \end{bmatrix}$, then our parity check matrix is $H = \begin{bmatrix} -A \\ \hline I_{n-k} \end{bmatrix}$, where $-A$ is the additive inverse of A, and I_{n-k} is the $(n-k) \times (n-k)$ identity matrix. This tells us that H is a $n \times (n-k)$ matrix. In order to decode a received vector $\vec{v}$, we perform the following steps [1]:

1. Calculate $\vec{v}H$.
2. If $\vec{v}H = \vec{0}$, assume that no errors occurred during transmission.

3a. (If our linear code is binary): If $\vec{v}H$ is the i^{th} row of H for exactly one i, assume that an error was made in the i^{th} component of $\vec{v}$. However, if $\vec{v}H$ is more than one row of H, we do not decode.
3b. (If our linear code is not binary): If there is exactly one instance of a nonzero element $s \in F$ and a row i of H such that $\vec{v}H$ is s times row i, assume that the sent word was $\vec{v} - (0...s...0)$, where s occurs in the i^{th} component. If there is more than one such instance, we do not decode.
4. If $\vec{v}H$ does not fit into either number 2 or 3, we know that at least two errors occurred, and so we do not attempt to decode.

As we see by the following theorem, we are able to use this fairly simplistic decoding method when there is at most one error; if there is more than one error, this decoding method fails.

Theorem [1] (Parity-Check Matrix Decoding): Parity-check matrix decoding will correct any single error if and only if the rows of the parity-check matrix are nonzero and no one row is a scalar multiple of any other row.

Because of the way that the parity-check matrix is defined, we are now able to define a linear code in terms of the parity-check matrix. The following definition gives us a more well-rounded understanding of a linear code (note that in this definition, the H referred to is the transpose of the H from our previous definition):

Definition [3]: Let H be an $(n-k) \times n$ matrix with elements in F_q. The set of all n-dimensional vectors $\vec{x}$ satisfying $H\vec{x}^T = 0$ over F_q is called a **linear (n, k) code** C over F_q of block length n, where $\vec{x}$ is a code word in C, and the matrix H is called the parity-check matrix of the code C.

One thing is made very clear through this definition. Since we know that C forms an additive group [1], and since $\forall \vec{x} \in C, H\vec{x}^T = 0$, then C is the kernel (nullspace) of H [3].

Though H is defined differently in this definition, throughout this paper we will use the form of H that we defined in our first, general definition of a parity-check matrix.

In order to see how parity-check matrices work, we will look at an example. Suppose we take the $(6, 3)$ linear code from above, in which our generator matrix $G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$.

We determine $H = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. If we take our code word 010101 and multiply it by H, we get:

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} = \vec{0}.$$

As a result, we see that since our vector times H was the zero vector, then our vector is a valid code word. If, on the other hand, we receive the vector 010100 (the last bit of the vector has an error), when we multiply by H, we get:

$$\begin{bmatrix}0 & 1 & 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0 & 1\end{bmatrix} \neq \vec{0}.$$

Therefore, we know that an error occurred, since we did not get the zero vector. So now we look to try and match our resulting vector with a row in H, and see that it matches the 6^{th} (last) row in H. Thus, we deduce that an error occurred in the 6^{th} position of our received vector, and we would decode the vector as 010101.

Using all of the information that has been presented so far, we will do a large example incorporating most of the aspects of linear codes learned up to this point.

Suppose that we want to create a (9,4) linear code that will have a Hamming weight of 4. We see that using this code, if we choose to do error correction and detection as separate processes, because $2t + 1 = 4$, we see that we will either be able to correct $t = \frac{3}{2}$, which we round to 1, or detect $2t = 3$ errors. If we choose to do error correction and detection simultaneously, since $2t + s + 1 = 4 \implies t = 1, s = 1$; or $t = 0, s = 3$; or $t = \frac{3}{2}, s = 0$. In addition, we are able to calculate the code rate (efficiency) of this code by noting that $R = \frac{4}{9} \approx 0.444$. Hence, we see that our efficiency is 0.444. This is not the best efficiency that could be obtained, and could be improved by changing some of the dimensions of the code. For our purposes, we will keep these dimensions in order to provide a cohesive example.

In order to encode our code words, we will need to create a 4×9 generator matrix. We note that the rows of the generator matrix will be the basis vectors of the 4-dimensional subspace V and also span V. As a result, we define our generator matrix as:

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

This results in the following code words:

$$000000000, 000101101, 001010110, 001111011, 010001110, 010100011, 011011000, 011110101,$$
$$100010011, 100111110, 101000101, 101101000, 110011101, 110110000, 111001011, 111100110.$$

We are now able to calculate our parity-check matrix H from the rows of the generator matrix:

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

By using the parity-check matrix method of decoding code words, we will be able to correct any one error in any of our code words.

In order to determine if our code is perfect or not, we see use the following formula:

$$M(1 + (q-1)\binom{n}{1} + \cdots + (q-1)^t\binom{n}{t}) \leq q^n.$$

Since $|F| = q = 2$ and there are $q^k = 2^4 = 16 = M$ possible sent codewords and since we have chosen to only correct 1 error, we see that $16(1 + (2-1)\binom{9}{1}) = 160 \leq 512 = q^n$. As a result, since $160 \neq 512$, our $(9, 4)$ linear code is not perfect. Intuitively, these results signify that there are quite a few code words with more than one error that could be received. If we had chosen to correct more errors, we would have had a greater chance of the code being perfect.

We wish to transmit the message: "HI." In order to use the linear coding procedure outlined above, we must transform the letters of the message into binary vectors. We respresent H as 8 and I as 9, and so in binary form, H is denoted as 1000, whereas I is denoted as 1001. When we multiply the message 1000 by G and we get 100010011 as the associated code word. In addition, when multiplying the message 1001 by G, we get 100111110 as the associated code word [5].

In order to decode these code words and check for errors, we run the code words through the parity-check matrix. We see that when we multiply the two received code words by H, we get $\vec{0}$ since no errors occurred in the transmission process. However, suppose that the sent code words were the two defined above, but the received code words were 100011011 and 110011110, respectively. Since we know what the original code words were, we see that the first received codeword has one error that occured in the sixth position, while the second received code word has two errors that occurred in the second and fourth positions.

In order to check for errors, we multiply the received codewords by the parity-check matrix. We see that

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

Since this vector is the same as the sixth row of the parity-check matrix, we see that there was an error made in the sixth position of the codeword during transmission. Therefore, we can correct our received codeword from 100011011 to 100010011.

Similarly, we see that for the second received vector:

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

Since this vector cannot be seen in any of the rows of the parity-check matrix, we know that more than one error occurred during transmission. As a result, we will not decode this vector. In actual implementation, we would probably request a retransmission.

In the end, after obtaining the correct codewords (100010011 and 100111110), we would be be able to strip the codewords of their redundancy terms, and so obtain the message. After converting the binary messages (1000 and 1001) back into messages in base ten, we would receive the digits 8 and 9, which correspond to letters "H" and "I," accordingly. Thus, in the end, we receive the message: "HI."

There are many different types of linear codes, including cyclic, Reed-Solomon, and Reed-Mueller. One specific type of linear code that we will look at is the Hamming code.

One important fact about Hamming codes that makes them distinct from other kinds of linear codes is their length. Hamming codes are of length $2^r - 1$ $(r \geq 2)$ where $k = n - r$ and r represents the number of parity bits in each code word [3]. In addition, in order for a linear code to be considered a Hamming code, there needs to be at least 2 ones in each row of the A part of the generator matrix [5]. Also, each Hamming code has a minimum weight of 3, making Hamming codes single-error correcting codes [6]. As a result, we can denote Hamming codes to be (n, k, d) where d is the Hamming weight of the code [3]. Another important property is that Hamming codes are always perfect [6].

Sometimes, it is desirable to add another parity bit in order to improve the error-correction and error-detection ability of a code. With Hamming codes, we are able to create an "extended Hamming code" by adding an extra parity column and row to our parity-check matrix. So if $H = \begin{bmatrix} a_{1,1} & \cdots & a_{k,1} \\ \vdots & \ddots & \vdots \\ a_{1,n-k} & \cdots & a_{k,n-k} \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$, then our new $H^* = \begin{bmatrix} a_{1,1} & \cdots & a_{k,1} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{1,n-k} & \cdots & a_{k,n-k} & 1 \\ 1 & \cdots & 0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 1 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$.

This is important because it increases our Hamming weight to a minimum of 4, which gives us an ability to correct/detect more errors.

One example of a Hamming code is the $(7, 4, 3)$ code found by using the generator matrix $G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$. The general $(7, 4, 3)$ Hamming code is important because it was the code that Richard Hamming introduced in 1950. It is also widely used as an example because it is one of the simplest Hamming codes.

This code can in fact be modeled with a Venn diagram, in order to help us illustrate the error detecting/correcting process. Recall that a code word consists of two parts, the message part of length k and the redundancy part of length $n - k$. In our case, we can write each code word as $d_1 d_2 d_3 d_4 p_1 p_2 p_3$, since messages are four digits long and the code words are seven digits long. We can then put each of these bits into our Venn diagram:

In this diagram, there are an even number of ones in each of A, B, and C. If we receive a vector and see that there are an odd number of ones in A, B, and/or C, then we know that an error occurred during transmission.

For example, say we receive the vector 110100. When we put this vector into our Venn diagram, we get:

and we see that there are two ones in each of A, B, and C, so we know that 110100 is a valid codeword, and we assume that no errors occurred during transmission. If, on the other hand, we received the vector 1101010 (noting that our second-to-last bit was flipped), then when we put it into the Venn diagram it looks like this:

and we see that while there are still two ones in A and also in C, in B there are three ones. This shows us that there is an error in our code word (since three is not an even number). If we wanted to try to correct the error, we would note that only B contained an odd number of ones. Because of this, we would assume that the error occurred in p_2, which is the second parity-check bit. Therefore, we would decode the vector to be 1101000, which we know to be the correct code word. However, this process will not work if multiple errors occurred during transmission, and we may end up wrongly decoding the vector if several errors have occurred. Therefore, more often than not, we will simply detect the error(s) and request a retransmission of the vector, rather than trying to correct the errors that have occurred.

While there are various methods to decode Hamming codes including decoding by parity-check matrices, we will look at a different method that uses checking numbers that will tell us in what position the error occurred. First, we will look at the different binary representations of different base ten numbers:

$$\begin{aligned} 1 &= 1 \\ 2 &= 10 \\ 3 &= 11 \\ 4 &= 100 \\ 5 &= 101 \\ 6 &= 110 \\ 7 &= 111 \\ 8 &= 1000 \\ 9 &= 1001 \\ 10 &= 1010 \\ 11 &= 1011 \\ 12 &= 1100 \\ &\vdots \end{aligned}$$

We see that with 1,3,5,7,9,and 11, there is a 1 in the last position. Similarly, we see that with 2,3,6,7,10, and 11, there is a 1 in the second to last position; with 4,5,6, and 7, there is a 1 in the third-to-last position, etc. Using this knowledge, we will define a way to decode Hamming codes using a *checking number*.

In order to understand this error-correcting procedure, we will go through an example. For this example, we will be using a $(7, 4, 3)$ Hamming code (note that the last three digits in each code word represent the redundacy part), and we will send the codeword 0111100. However, let's assume that an error occurs in the 3^{rd} position during the transmission process, which means we will receive the vector 0101100. In order to decode, we will create a checking number that will tell us where the error occurred (if any error occurred). We perform the following procedure [7]:

1. Add up the digits corresponding to positions 1,3,5,7 to get a checking number digit c_1.
2. Add up the digits corresponding to positions 2,3,6,7 to get a checking number digit c_2.
3. Add up the digits corresponding to positions 4,5,6,7 to get a checking number digit c_3.

4. The resulting checking number $c_3c_2c_1$ will tell us the placement of the error in the vector.

We see in our example that $c_1 = 0 + 0 + 1 + 0 = 1$, $c_2 = 1 + 0 + 0 + 0 = 1$, and $c_3 = 1 + 1 + 0 + 0 = 0$, since we are in binary. As a result, the checking number is $c_3c_2c_1 = 011 = 3$. This means that an error occurred in the 3^{rd} position of our received vector. This matches our observation. Thus, we correct the received vector from 0101100 to 0111100 [7].

As was noted before, there are many different kinds of linear codes. These codes are implemented frequently today as they can be used in many applications, such as in telecommunications, CD players, and disc drives. Because they are error-correcting and -detecting codes, they have the ability to be used in many different situations involving intense computing. Richard Hamming created Hamming codes for the specific purpose of error-correction and -detection. Because he did a lot of long computations on his computer, he wanted to be able to create an encoding system that would detect and correct errors automatically. With this code, he could run a program over the weekend or over an extended period of time and not have to work through errors by hand. Hamming codes were actually one of the first error-correction and -detection codes created. Because they are simple and very versatile, they are used in many different applications today as well.

[1] Gallian, Joseph. *Contemporary Abstract Algebra.* 7^{th} ed. Belmont, CA: Brooks/Cole, 2006. Print.

[2] Xing, Chaoping and San Ling. *Coding Theory: A First Course.* West Nyack, NY: Cambridge University Press, 2004. Web.

[3] Lidl, Rudolf and Günter Pilz. *Applied Abstract Algebra.* New York: Springer-Verlag, 1984. Print.

[4] Grossman, Jay. "Coding Theory: Introduction to Linear Codes and Applications." *Rivier Academic Journal* 4.2 (2008). Web.

[5] Johnston, William and Alex McAllister. *A Transition to Advanced Mathematics.* New York: Oxford University Press, 2009. Print.

[6] <http://www.mth.msu.edu/ jhall/classes/codenotes/Hamming.pdf>.

[7] Lemmermeyer, Franz. *Error-correcting Codes.* 2005. <http://www.fen.bilkent.edu.tr/ franz/lect/codes.pdf>.

[8] Neubauer, Andre, Jurgen Freudenberger, and Volker Kuhn. *Coding Theory: Algorithms, Architectures and Applications.* Hoboken, NJ: Wiley, 2008. Web.