1 Introduction and Motivation

Over the last 70 years, algebraic coding has become one of the most important and widely applied aspects of abstract algebra. Coding theory forms the basis of all modern communication systems, and is the key to another area of study, Information Theory, which lies in the intersection of probability and coding theory. Algebraic codes are now used in essentially all hardware-level implementations of smart and intelligent machines, such as scanners, optical devices, and telecom equipment. It is only with algebraic codes that we are able to communicate over long distances, or are able to achieve megabit bandwidth over a wireless channel.

Algebraic coding is most prevalent in communication systems, and has been developed and engineered because of one inescapable fact of communication: noise. Noise will always be a part of communications, and has the potential to corrupt data and voice due to its presence. Noise comes from practically an infinite number of sources, from cosmic background radiation (affecting space based communication), from an inductive motor in a vending machine down the hall, and can even be generated by the user themselves by induced signal reflections in the environment. The implications of destructive interference in communications is obvious: mission critical communiqué potentially couldn’t be trusted and decisions based on those communications could not be made.

Consider these basic applications of algebraic codes. Suppose that in the case of two warring nations, a binary message is to be sent indicating an intention of surrender or an intention of war. If a binary 1 is sent, the nation surrenders. If a binary 0 is sent, then war it is. In this time of such rudimentary communication, there is no concept of noise or error correction, and so it is possible, if not likely that due to noise a transmitted 0 to be received as a 1, or vice versa. To make this system substantially more robust, a party can transmit five bits, and the receiver then infers a message based on the majority contents. For instance, if 00000 meant surrender and was sent, though due to noise a 00100 was received, the message remains intact and the white flag is raised. Based on this sender-receiver agreement, up to three errors can occur before the messages intent is
reversed and ultimately lost. The probability of three bit errors occurring can be shown to be lower than a single error, and so the addition of this decoding makes the system more robust. This decision process is called the maximum-likelihood decoding procedure, and will be discussed further.

A more involved yet more pertinent example is the method in which current generation cell phones prevent cross-talk and ensure user security. This example will be explained further once the requisite background is established, though an overview will suffice for now. Code Division Multiple Access, or CDMA, is a dominant cell phone standard in North America, and operates on the idea of orthogonal codes. In a cell phone environment, there are multiple users talking on the same network, at the same time using the same space in a finite frequency band. How do the users remain independent of each other and avoid cross talk? Each user is assigned a unique spreading sequence, or code, and then network identifies and routes traffic based on these unique sequences. The code is an $M$ length binary vector that is multiplied onto the user’s signal. The key here is each of the codes are orthogonal and of high dimensionality relative to the number of users, so that the user’s conversations do not literally collapse into each other.

Two main branches of coding theory are source coding and channel coding. They are so named because the former manipulates the source to allow more efficient transmission (ie smaller size messages) while the latter addresses the errors that may be introduced in the transmission channel. The fundamental theorem of source coding was given by Claude Shannon in 1948, widely considered the father of Information Theory. Shannon’s Theorem describes the best possible error-correction of a code given certain parameters. Source coding is more within the computer science and engineering discipline, with main applications being compression of data prior to transmission. Our textbook focuses on error-correcting codes, since these find algebra more applicable.

One of the most fundamental concepts to understand about channel coding is that it is only possible to catch errors if there are some restrictions on what constitutes a proper message. The receiver needs to have an idea of the shape of what it will be receiving. If the message space is entirely composed of legitimate codewords then any error will change one code word into another, and the receiver will not be able to discern that the seemingly legitimate code word that was received is not the same as what was sent. Note that it is difficult to discuss encoding without including the corresponding decoding method. The most fundamental concept in encoding is to build redundancy into the message.

## 2 Mathematical Discussion

### 2.1 Code Words

There are several different kinds of codes, however one of the most common is the linear code.
**Definition 1.** A \((n, k)\) linear code over a finite field \(F\) is a \(k\)-dimensional subspace \(V\) of the vector space 
\[ F^n = F \oplus F \oplus \cdots \oplus F \] 
over \(F\). The vectors \(\alpha \in V\) are called the code words. When \(F = \mathbb{Z}_2\), we refer to working with binary codes.

A \((n, k)\) linear code over a field \(F\) can be thought of as a set of \(n\)-tuples from \(F\), where each vector contains both the message word and a redundancy, which are the remaining \(n-k\) components of the code word. For any finite field order \(q\), there are then \(q^k\) possible code words. In the common base of binary codes, for \(n\) digits, there are \(2^n\) possible code words.

**Example 1.** The set \(\{0000, 0101, 1010, 1111\}\) is a \((4, 2)\) binary code. The first two digits of each string are the numbers 0 through 3, whereas the trailing two bits illustrate the redundancy.

When discussing error-correcting and detecting capabilities, it is necessary to refer to the **Hamming Distance** and **Hamming Weight**.

**Definition 2.** Hamming Distance. The Hamming distance between any two vectors \(\alpha, \beta \in V\) is the number of components in which they differ. Let \(d(\alpha, \beta)\) denote the Hamming distance between the two vectors \(\alpha, \beta\).

**Definition 3.** Hamming Weight. The Hamming weight of a vector \(\alpha \in V\) is the number of nonzero components. The Hamming weight of a linear code is the minimum weight of any nonzero vector in the code. Let \(wt(\alpha)\) denote the Hamming weight of the vector \(\alpha\).

Another way to look at the Hamming distance is to notice that this is the number of substitutions required to change one vector into another, or the number of errors that transformed one codeword into another. The Hamming weight can also be thought of as the difference between the Hamming distance of a code word and the zero code, \(\{00 \cdots 0\}\). For binary codes, the Hamming distance is merely the number of ones in the codeword.

### 2.2 Error Detection

Linear codes allow for well defined algorithms for error detection. These routines are so mature and developed that they are now typically implemented with hardware circuits rather than software programs. How well an error is detected is based on the code’s Hamming weight. In the end, it may come down to a choice between detection and correction.

**Theorem 1.** If the Hamming weight of a linear code is at least \(2t + 1\), then the code can correct any \(t\) or fewer errors. Consequently, the same code and detect \(2t\) or fewer errors.
Proof. Suppose that a transmitted code word $u$ is received as the vector $v$, and that at most $t$ errors have been made in transmission. Then, by definition we have $d(u,v) \leq t$. If $w$ is any code word other than $u$, $w - u$ is a nonzero code word. We may assume

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

and so $t + 1 \leq d(w,v)$. So the code word closest to the received vector $v$ is $u$, and therefore $v$ is correctly decoded as $u$. \hfill \Box$

This theorem is interesting because it implies that the user must choose either to correct $t$ errors or detect $2t$ errors. This means that sometimes it is possible to not decode a received message at all. In this case, the received will request a retransmission of the applicable packet or message.

**Example 2.** Let the Hamming weight of a vector be 4. Then, we know it will correct any single error and detect any two errors ($t = 1, s = 1$) or detect any three errors ($t = 0, s = 3$)

To generate a linear code, a useful method is via a generator matrix. The matrix $G$ is a linear transform which maps a subspace $V$ of $F^k$ to a subspace $W$ of $F^n$ such that for any $\alpha \in V$ the vector $vG$ will agree with $v$ in the first $k$ components and build an amount of redundancy into the vector. This $k \times n$ linear transform has a matrix representation of

$$\begin{bmatrix} I_{k \times k} & A_{k \times n} \end{bmatrix}$$

where the individual elements of $A$, $a_{ij} \in F$. This linear transform $G$ is called the standard generator matrix, and basically is able to encode any code. It is worth noting that any matrix of rank $k$ will transform $F^k$ to a $k$-dimensional subspace of $F^n$, but the standard generator matrix has the advantage that the original message vectors form the first $k$ components of the resultant vectors.

2.3 Error Decoding

Once an error has been detected, it must be corrected, or decoding via a given method based on the original source encoding. The receiver needs to know what method to employ, and relies on having prior knowledge on the method that was originally used in encoding.

A very common method of decoding is the Parity-Check Matrix. This method is still the predominant decoding technique for standard telephone modem systems. For this method, we presume all of the code words were encoded via a common generator matrix $G$. We simply need to undo the encoding that the matrix $G$ put onto the words.

Let $V$ be a linear code over the finite field $F$ given by the standard generator matrix $G = \begin{bmatrix} I_{k \times k} & A_{k \times n} \end{bmatrix}$. Then, the $n \times (n - k)$ matrix

$$H = \begin{bmatrix} -A \\ I_{n-k} \end{bmatrix}$$
is referred to as the parity check matrix. To decode any received message vector \( m \), we follow this procedure:

1. If the product \( mH \) is zero, then we presume that no error was made.

2. If there is a unique nonzero row \( i \) of \( H \) such that \( mH \) is \( s \) times row \( i \) for some \( s \in \mathbb{F} \), assume the sent word was \( m - (0 \cdots s \cdots 0) \), where \( s \) occurs in the \( i^{th} \) component. If there is more than one instance, do not decode.

3. If the code is binary, then: if \( mH \) is the \( i^{th} \) row of \( H \) for exactly one \( i \), then we know than an error was made in that component of \( m \). If \( mH \) is more than one row of \( H \), do not decode.

As this point of the discussion, it would be appropriate to make the point that any reference to Coding Theory or Information Theory must absolutely include a discussion of probability. This is important for many reasons. In all of these cases, there is an underlying idea of the probability that an error occurred. Many times, in processor and power limited cases (such as spacecraft), computational resources are finite, and can only be expended if it is absolutely necessary. Though it is out of the scope of this paper, generally all complex digital devices first calculate a Bit Error Probability before engaging in error correction or error decoding. If the BER is above a certain threshold, then error correction is attempted. The error probability is intensely dependent on the type of communication system, the kind of noise, and the transmission method. Additionally, several decoding procedures depend exclusively on the probability of an error occurring.

For the following examples, let \( C \subset \mathbb{F}_2^n \) be a linear code of length \( n \) and \( x, y \in \mathbb{F}_2^n \).

**Example 3.** In Ideal Observer Decoding, the code word \( x \) is received. This method then picks a code word \( y \in C \) to maximize the conditional probability

\[
P(y \text{ sent}|x \text{ received})
\]

This is also known as maximizing the entropy of the system, a concept from Information Theory. In the event that more than one \( y \) is selected (that is, the system is over determined), it is usually requested that the message be retransmitted.

**Example 4.** In Maximum Likelihood Decoding, the codeword \( x \) is received and then a \( y \in C \) is chosen to maximize the conditional probability

\[
P(x \text{ received}|y \text{ sent})
\]

In other words, in Ideal Observer Decoding, we choose a \( y \) that would have most likely been received as \( x \). In Maximum Likelihood Decoding, we choose \( y \) that would have most likely resulted in \( x \) being received.
Theorem 2. If each code word is equally likely to be sent, then Maximum Likelihood is equal to Ideal Observer Decoding.

Proof. By the definition of conditional probability, we know that

\[ P(x \text{ received}|y \text{ sent}) = \frac{P(x \text{ received} \cap y \text{ sent})}{P(y \text{ sent})} = \frac{P(x \text{ received} \cap y \text{ sent})}{P(y \text{ sent})} \cdot \frac{P(y \text{ sent})}{P(x \text{ sent})} = \frac{P(x \text{ received} \cap y \text{ sent})}{P(x \text{ sent})} = P(y \text{ received} \cap x \text{ sent}) \]

Example 5. In Nearest Neighbor Decoding, \( x \) is received and \( y \) is chosen to minimize the hamming distance \( d(x,y) = \# \{ i : x_i \neq y_i \} \). In other words, this method chooses the \( y \in C \) that is closest to \( x \in F^n_2 \). As with the other methods, in the event that more than one distinct \( y \) is chosen, the message is retransmitted in the hope the errors will not occur in the same fashion.

Theorem 3. If the probability of error \( p \) is less than 0.5, and each statistical event are mutually independent, then Nearest Neighbor Decoding is equivalent to Maximum Likelihood Decoding.

Proof. Let \( d \) be the Hamming distance between two vectors \( x \) and \( y \). Then

\[ P(y \text{ received}|x \text{ sent}) = (1-p)^{n-d}p^d = (1-p)^n \left( \frac{1}{1-p} \right)^d = (1-p)^n \left( \frac{p}{1-p} \right)^d \]

Since \( p < 0.5 \), this quantity is maximized by minimizing \( d \). The equality of the first line is a result of the probabilistic nature of the problem, and verification is left to the reader.

There are numerous schemes of coding/decoding, each suited to its own best and worst case. Some of these include coset decoding and syndrome decoding, both of which are very valid systems. The mechanics of these decoding operations are omitted in this paper, however they are used later in the problem set in a manner such that their properties can easily be discerned.
3 Code Division Multiple Access

Algebraic coding is what makes wireless communication possible at bandwidth above speeds in the single bits per second. We will investigate how this is possible, but we need some background first.

In a cell phone environment, all of the users in a given location are all talking at the same time on the same frequency band. Because of this, it would be logical to think that as one user uses increasing bandwidth for different applications (such as wireless email), all other user’s links must suffer proportionally. To get around this, CDMA allows all users access to all frequencies at all times. Therefore, each user spans enough “good” frequencies to access all requested services. However, by allowing several users access to the same frequency at the same time, on the same channel, how are we able to recover each user’s information? To do this, we must create an orthogonality structure of the signals that the users are assigned. This structure is an algebraic code, called a spreading sequence.

Definition 4. Let $C$ be an $(n, k)$ linear code over $\mathbb{F}$ with generator matrix $G$ and parity-check matrix $H$. Then for any vector $v \in \mathbb{F}^n$, we have $vH = 0$ if and only if $v \in C$

Another, more practical way of thinking about orthogonality is for any two vectors, $\alpha, \beta \in C$ we have $\cos \theta_{\alpha, \beta} = 0$. Even more generally two vectors, $\alpha, \beta$ are orthogonal if their inner product $\langle \alpha, \beta \rangle = 0$. With this understanding of orthogonality, we are able to transmit our bits.

Suppose Users 1 and 2 want to transmit one information bit to their receivers. Let the bits be given by

$$b_1[1] = 1, \quad b_2[1] = -1$$

Remember that both users will be transmitting at the same time at the same frequency. To separate $b_1[1]$ and $b_2[1]$ at the receiver, we must assign mutually orthogonal codes to each. These codes can be thought of as vectors in “high” dimensionality vector space. For this simple example, let the vector space $V = \mathbb{R}^4$. Let Users 1 and 2 be assigned the respective codes

$$p_1[m] = \{1, 1, -1, -1\}$$
$$p_2[m] = \{1, -1, 1, -1\}$$

We see that by taking the inner product of the vectors that they are in fact mutually orthogonal to each other. We will follow the following steps for the recovery of our coded bit at the receiver.

- Each users information bits are first multiplied onto their individual codes.
- The total transmitted signal is created by adding together all of the coded sequences.
At the receiver, individual filters use the different codes to recover each user’s information bits from the received signal.

To transmit, first we multiply these codes by our information bit, resulting in $b_1^1 p_1^1[m]$ and $b_2^1 p_2^1[m]$. In this current example, the system transmits the sum signal $s[n]$ over the channel:

$$s[n] = b_1^1 p_1^1[m] + b_2^1 p_2^1[m], \quad 1 \leq n \leq 4, 1 \leq m \leq 4$$

We now have

$$s[n] = \{0, 2, -2, 0\}$$

For this example, assume there is no noise and that we are operating under the convenient mathematical fiction of a loss-less channel so that our received signal is $r[n] = s[n]$. The individual bits $b_1^1$ and $b_2^1$ can be recovered by sending the received signal $r[m]$ through detection filters which are matched to the individual codes. The rationale shall be omitted for brevity, however we will define the detection filter $h_i^i[m]$ for the $i$th user as the reflected version of the code. Recalling our original orthogonal codes, this gives us for our matched detection filters

$$h_1^1[m] = \{-1, -1, 1, 1\}$$
$$h_2^1[m] = \{-1, 1, -1, 1\}$$

To obtain the output of the receiver, $y_i[n]$ we must pass the received signal through the output filter. In the case of all linear, time invariant systems such this the output is defined as the convolution of the input with the system’s impulse response. In this case, it is the convolution of $h_i^i[m]$ with $r[m]$. Mathematically, the convolution sum is

$$y_i[n] = \sum_{m=1}^{4} h_i^i[m] r[n + 1 - m]$$

or more compactly and elegantly,

$$h_i^i[n] \ast r[n]$$

Performing this convolution reveals that at time $n=4$ (the length of the code) the magnitude of $y_i[n]$ is greatest. This is where the determination of which bit was sent is made. At $n = 4$, $y_1[4] = +4$. This signifies a positive correlation between the received signal at time $n = 4$ and $p_1$, and so the received bit is $b_1^1 = 1$. Similarly, $y_2[4] = -4$, which is where we sample. Because the value is negative, we know the bit $b_2^1 = -1$.

This example shows that with a system of orthogonal codes we are able to reconstruct the received bits of several users transmitting in a cellular environment at the same time, on the same frequency range. In actual environments, the length of the codes are on the order of 1024 or 2056 long in order to ensure orthogonality for a substantial number of users. In cases such as this, the spreading sequences are generated from long pseudo random sequences of equally likely $\pm 1$ symbols.
4 Gallian Exercises

31.3 Referring to Example 1, use the nearest-neighbor method to decode the received words 000110 and 110100.

1. 0000110: This string has one error, and is in fact 1000110.
2. 1110100: According to nearest-neighbor, we decode the string with no errors.

31.10 Let \( C = \{0000000, 1110100, 0111010, 0011101, 100110, 0100111, 1010011, 1101001\} \).
Determine the error-correcting and error-detecting capability of \( C \).
The weight of \( C \) is 4, so there exists two options: we can detect up to any 3 errors, or we can correct 1 error and detect two other errors.

31.13 Find all code words of the \((7,4)\) binary linear code whose generator matrix is

\[
G = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix} = \begin{bmatrix} I_{4 \times 4} \mid A \end{bmatrix}
\]

To do this, we treat the binary numbers representing 0 through 15 (e.g. all permutations of a 4 bit binary word) as our message vector \( m_i \) and multiply them by \( G \) to form the \( i^{th} \) row of \( C \). All of the individual code words are:

\[
C = \left[ \frac{m_0 \cdot G}{m_1 \cdot G} \ldots \frac{m_{15} \cdot G}{m_{15} \cdot G} \right] = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]
Where the parity check matrix is:

\[ H = \begin{bmatrix} -A \\ I_{3\times3} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

The weight of this code is 3, so it is able to correct any single error.

31.17 Construct a (6, 3) binary linear code with generator matrix

\[ G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I_{3\times3} & A \end{bmatrix} \]

Here we are working with 3 bit binary words, so there are 8 possible words to send. To develop the code, we again treat the word as a 3 \times 1 vector and form the code by multiplying by \( G \) for the respective rows of \( C \). The code words are then:

\[ C = \begin{bmatrix} m_0 \cdot G \\ \vdots \\ m_7 \cdot G \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \]

The codes are constructed so that the respective decodings are as follows. Via the nearest-neighbor method:

1. 001001 is decoded as 001101. Eliminating the trailing 3-bit redundancy, we finally obtain the word 001 (a single correction).
2. 011000 is decoded as 111000. Similarly eliminating the acquired redundancy (this comment will be omitted for future decodes), we obtain 111 (a single correction).
3. 000110 is decoded as 100110 \( \rightarrow \) 100 (a single correction)
4. 100001 is not decoded as there are greater than two errors.
To decode with the parity check method, we must first find the parity check matrix $H$.

$$
H = [-A]_{3 \times 3} \begin{bmatrix}
1 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
$$

Decoding the given vectors $d_i$ from the nearest neighbor approach we have

1. $d_1 = 001001$, and $d_1 \cdot H = 100$, so we conclude there exists an error in the 4th component of $d_1$. Then, decode as 001101 $\rightarrow$ 001.

2. $d_2 = 011000$, and $d_2 \cdot H = 110$, so we conclude there exists an error in the first component of $d_2$. Then, decode as 110000 $\rightarrow$ 111.

3. Similarly, $d_3 \cdot H = 110$ so we decode as 100110 $\rightarrow$ 100.

4. Lastly, $d_4 \cdot H = 111$, so we know there exists two or more errors, and so we do not decode.

For a coset decoding approach, we must define a standard array. To do this, we place the code words from $C$ as the first row. For each subsequent rows, choose a vector not already in the standard array, and call this the *coset leader*. Then, generate the coset of this vector with $C$ and list this as the next row. For reasons which will become obvious, we will note the location of our received words in the array. This results in

$$
S = \begin{bmatrix}
00000 & 100110 & 010011 & 001101 & 110101 & 101011 & 011110 & 111000 \\
10000 & \text{000110} & 110011 & 101101 & 010101 & 001011 & 111110 & 011000 \\
01000 & 110110 & 000111 & 011101 & 100101 & 111011 & 001110 & 101000 \\
00100 & 101110 & 011011 & 000101 & 111101 & 100011 & 010110 & 110000 \\
00010 & 100010 & 010111 & \text{001001} & 110001 & 101111 & 011010 & 111100 \\
00001 & 100111 & 010010 & 001100 & 110100 & 101010 & 011111 & 111001 \\
100001 & 001111 & 110010 & 101100 & 010100 & 001010 & 111111 & 011001
\end{bmatrix}
$$

To decode, we locate each received word in the array and then decode the word at the top of the column in which the word is located.

1. $001101 \rightarrow 001$

2. $111000 \rightarrow 111$

3. $100110 \rightarrow 100$
Lastly, to decode using the syndrome method we calculate the syndromes of the coset leaders by taking the product of the code word and parity check matrix $H$.

\[
\begin{align*}
000000 \cdot H &= 000 \\
100000 \cdot H &= 110 \\
010000 \cdot H &= 011 \\
001000 \cdot H &= 101 \\
000100 \cdot H &= 100 \\
000010 \cdot H &= 010 \\
000001 \cdot H &= 001 \\
100001 \cdot H &= 111 \\
\end{align*}
\]

We then calculate the individual syndromes of the received words in an identical fashion and check for matches with the code word and parity check matrix products.

\[
\begin{align*}
001001 \cdot H &= 100 \\
011000 \cdot H &= 110 \\
000110 \cdot H &= 110 \\
100001 \cdot H &= 111 \\
\end{align*}
\]

So that we decode with the bit-wise difference between the matches as follows:

1. $001001$ is decoded as $001001 - 000100 = 001101$ → $001$
2. $011000$ is decoded as $011000 - 100000 = 111000$ → $111$
3. $000110$ is decoded as $000110 - 100000 = 100110$ → $100$
4. $100001$ is decoded as $100001 - 100001 = 000000$ → $000$