Chapter III: Linear Codes

III.1 Linear Codes

III.1.1 Introduction. We now try to construct some practical codes. Our first step in introducing structure is to define addition among code and message words, and consider codes whose encoding is a *linear transformation* from message words to code words. We will see that the structure we introduce will allow our codes to have error correction properties, as they did in the last section of the previous chapter, but it also will make it easy to encode and decode messages. In this section, we will limit ourselves to the relatively simple problem of trying to find a code capable of finding and correcting one error, given m message bits.

As in the previous section, our plaintext messages will be binary strings of length m, and the code messages we send will be binary strings of length N. If X is the message the sender wants to send, we let $Y = c(X)$ denote the corresponding code word. If the sender sends Y over a noisy channel, so that some of the bits get flipped, we call the message that the receiver receives Z.

Our decoding scheme will involve interpreting the received message Z as coming from the code word $c(X')$ that is closest to Z, in the sense that $c(X')$ differs from Z in fewer bits than the code word for any other message does, i.e. we decode Z to be the message word X' for which the *Hamming Distance* between Z and $c(X')$ is minimum. (Recall that in Section II.4 we defined the Hamming distance between two code words to be the number of bits in which they differ.) If $X' = X$, we win.

First notice that if we have a code where there are two code words which are only Hamming distance 1 apart, then if we send one of these code words, and there is an error in that bit, we can't tell which was the original message sent. Therefore, we need all our code words to be more than Hamming distance 1 apart.

If the minimum Hamming distance between code words is 2, then one error will never interchange code words. But suppose code words x and y differ in only two bits; a single error in either of those bits will produce a word halfway between x and y, and we cannot distinguish between them without guessing. (A code with minimum Hamming distance 2 is called a *single-error-detection code*, because the presence of a single error will always be observable; but the code will not always be able to correct the error.)

If the minimum Hamming distance between code words is 3 and if there is one error, our decoding procedure can correct it: there can be at most one code word that is a Hamming distance 1 from any received word. For example, suppose there are only two messages we need to send, 0 and 1. Then the code which encodes 0 as

000 and 1 as 111 is a single-error-correcting code: if there is at most one error made in sending our message, we might receive any of 000, 001, 010, 100 if the message was 0, and any of 111, 110, 101, 011 if the message was 1.

In general, a code with minimum distance $2d+1$ between code words is capable of correcting d errors, or detecting $2d$ of them.

III.1.2 Linear Codes. Up to this point, our codes have been binary strings. We now treat these strings as *vectors* in that we will define *addition* on them bit-by-bit, in the same manner as vector addition. We assume that our (plaintext) message words are m-bit binary sequences. We define addition on these to be addition mod 2 separately on each bit (with no bit-to-bit carrying.) Thus on each bit, we have $0+0=0$, $0+1=1$, $1+0=1$, and $1+1=0$. In this way, the encoded words form an N-dimensional vector space mod 2, and the plaintext messages are sequences of length m which when encoded form an m-dimensional subspace of an N-dimensional vector space mod 2.

In general, encoding represents a 1-to-1 mapping from the m-dimensional message vector space into the N-dimensional code word space. The encoding function c maps an m-bit message vector X to its N-bit code vector $c(X)$.

We say our code is a **Linear Code** if the encoding function c is a *linear transformation*, that is if for any messages X and X' we have

$$c(X+X') = c(X) + c(X').$$

Notice that a linear code will always encode the zero message to the N-bit vector of all 0's; for if X is zero, then we have $X+X=X$, and we obtain by linearity that $c(X) = c(X) + c(X)$, which means $c(X)$ must have 0's for all its components.

Note that in this system of arithmetic, *subtraction is the same as addition.*

In general we define the *weight* of a binary word (plaintext or encoded) to be the number of non-zero entries it has. The words that are of weight 1 are of particular interest and we introduce a notation for them; let $\langle k \rangle$ denote the word having all bits 0 except for the kth bit 1. A linear code is completely defined by describing how it acts on the weight-one message words $\langle k \rangle$. Linearity determines what it does on any sum of these to be the sum of their corresponding code words. For example, to encode the word 0111, we add up the code words for $\langle 2 \rangle$, $\langle 3 \rangle$ and $\langle 4 \rangle$; in linear algebra terminology, we say that the vectors $\langle k \rangle$ form a *basis* of the message space. To completely describe a linear code, we need only tell what it does to some basis of the message space. It immediately follows that we do not need 2^m code words to describe such a code (as we did in the previous section to describe a random code).

The standard way to describe a linear code is to write down the matrix M of the linear transformation c. We call this the encoding matrix of the code. The rows of the encoding matrix M are the code words for the weight one messages. Thus the kth row of M is $c(\langle k\rangle)$. Let's take an example:

Example 1. Suppose $m = 4$ and $N = 7$; and suppose we have $c(\langle 1\rangle) = 1000110$, $c(\langle 2\rangle) = 0100010$, $c(\langle 3\rangle) = 0010101$, $c(\langle 4\rangle) = 0001111$. The matrix M is then the 4×7 array of numbers shown below.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The encoding rule for this or any such code involves *matrix multiplication*; here it means that to encode the message X we add together the rows for which the component of X is 1. In general this encoding rule can be written as:

$$c(X) = XM.$$

This means that the jth component of $c(X)$ is $\sum_k X_k M(k,j)$, where X_k is the kth component of X and $M(k,j)$ is the entry of M in the kth row and the jth column. (Aside: In general the *matrix product* of matrices A and B has entry (i,j) given by $\sum_k A(i,k)B(k,j)$. It is defined when the number of columns of A is the same as the number of rows of B.) Thus, to encode the message 0110, we perform the (binary) matrix multiplication

$$(0\ 1\ 1\ 0)\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

obtaining the answer 0110111.

We now turn to the main question of this section: how can we find a single-error-correcting linear code? As we have noted, a code is able to correct a single error if and only if the minimum distance between code words is at least three.

Lemma 1 *A linear code will be able to correct a single error if and only if there are no code words of weight 1 or 2.*

Proof. $\Longrightarrow$: By contradiction. If there were a code word of weight 1 or 2, this word would be of Hamming distance 1 or 2 from the zero code word, and so the code would

not be able to correct an error.
$\Longleftarrow$: Again by contradiction. Suppose the linear code were not able to correct a single error. Then there must be two code words, X and X' such that their Hamming distance was less than three. Therefore, the vector $X - X'$ would have weight one or two. But by linearity of our code, $X - X'$ is also a code word. □

So our goal is to find a matrix M so that if we multiply any message vector by M, we get code words which have weight at least 3. We will actually be able to construct such an M with an additional nice property. We will construct a single-error-correcting linear code *whose first m digits are the message encoded.* Call the remaining $N - m$ digits of the code word the *check bits* for the message.

With all these conditions, we can say a lot about what the matrix M must look like. Since we want the first m digits of a code word $c(X)$ to be the same as the first m digits of the message X, the first m columns of the encoding matrix form an identity matrix, as in the example above. We denote the identity matrix by the letter I and occasionally add a subscript to indicate its size. The matrix for one of these codes then has the form

$$\begin{array}{ccc} & m & N-m \\ m & (I & K) \end{array}$$

where K is a matrix with m rows and $N - m$ columns. The matrix K will be called the *check bits* of the code.

Suppose our matrix code is single-error-correcting, so that no code word has weight 1 or 2. This imposes the following conditions on the K part of the encoding matrix:

1. The weight of each row in the K part of the matrix must be at least 2, since the whole row (being a code word) must have weight at least 3 while its I-part has weight exactly 1.

2. No two rows of K can be identical; else the weight of that code word obtained by summing (or taking the difference of) the two rows will have weight 2.

Example 2. Example 1 fails the first condition in its second row. However if we change the last entry in that row to a 1, the new matrix, given here, obeys both conditions:

$$\begin{array}{rcccccccc}
c(\langle 1\rangle) & = & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
c(\langle 2\rangle) & = & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
c(\langle 3\rangle) & = & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
c(\langle 4\rangle) & = & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}$$

The K-matrix for this code is

$$\begin{pmatrix} 110 \\ 011 \\ 101 \\ 111 \end{pmatrix}$$

In fact, we will show that these two conditions are sufficient to define a single-error-correcting code, and not only that, we can easily find and correct errors with such codes. That the conditions are sufficient follows from the observation that they are sufficient to ensure that the minimum weight of any code word is 3 or more. Any message word of weight 3 or more will yield a code word of weight at least three in just the I message part of its code word. It is any easy exercise to show that we have ensured with our conditions on K that any message word of weight 1 or 2 will also have a code word with weight at least 3.

We now prove that codes with these properties can correct errors in a more direct way: by giving a procedure for actually correcting the error with such a code. Suppose we know the encoding matrix M of a linear code, and we receive the vector Z, where we know Z differs from the code word actually sent ($Y = XM$) in at most one bit-position.

Suppose we can find an $N \times (N-m)$ matrix D no row of which is all zeroes and no two rows of which are the same, such that the matrix-product MD vanishes (i.e. is the matrix of all zeroes). We multiply the received vector Z by D. Now suppose Z is actually off, and has one error, i.e. it differs from Y by a single bit and hence by a single weight-1 error vector $\langle k \rangle$: $Z = XM + \langle k \rangle$. Here k marks the location of the error.

Then if we form ZD, we have

$$ZD = (XM + \langle k \rangle)D = XMD + \langle k \rangle D = \langle k \rangle D = k\text{th row of } D.$$

So to find our error location k, all we need to do is compute ZD and see which row of D it is. That will tell us the error location, k. Changing the kth bit of Z will give the encoded message $c(X)$, whose first m bits will be the original message. (Notice that if the coded message was, in fact, transmitted without error, and Z is actually a correct code word, then by the above equation, ZD will vanish, indicating there are no errors.)

But how do we find such a matrix D? The matrix of the form

$$\begin{pmatrix} K \\ -I_{N-m} \end{pmatrix}$$

does the trick. On multiplying this matrix on the left by

$$(I_m \; K)$$

one gets

$$(K - K)$$

(the $+K$ coming from I_m times K, and the $-K$ coming from K times $-I_{N-m}$). This is the zero matrix. (Note that since we are working mod 2, we can ignore the minus sign in the definition of D, and just take

$$\begin{pmatrix} K \\ I_{N-m} \end{pmatrix}$$

However, it is useful to keep in mind the minus sign when one is doing non-binary coding.)

Let's do an example with our last code.

Example 2 (continued). Suppose we receive a message (1110111). We form the D matrix

$$D = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and multiply:

$$XD = (\, 1 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \,) \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (1 \quad 1 \quad 1),$$

which corresponds to 4th row of D. In consequence, there is an error in 1110111 is in the fourth bit, and the original message was (1111), which are the first four bits of 1111111.

We have described a method of decoding a single-error-correcting linear code if it is in IK form. We might ask, suppose we have a code that is not in IK form. Can we use this same trick to find the error? The answer is yes.

The problem of finding the erroneous bit depends only on the code words, and does not depend upon which message word gets which code word. We may therefore make any change we want to the coding matrix that does not change the identity of the code words, without affecting the error-correcting problem at all.

What changes of M do not affect the code words? Changing the *basis* used to describe the message space words won't change the words, and will change the matrix. Thus, instead of describing the code by what it does to message words of weight 1, we can pick any other linearly independent set of m message words, and write the matrix whose rows (in any order) consist of the code words for these.

One simple way to do this is to replace any row of M by its sum with any (constant multiple of) any other row or rows of M (this is called an *elementary row operation*, as is multiplying a row by a constant; of course with binary addition, the only constant is 1 so that multiplying by constants doesn't do much). Any change of basis can be achieved by a sequence of such elementary row operations.

This supplies us with a way to locate the error when the matrix is not in IK form: put it in IK form by a sequence of *elementary row operations*. Call this new matrix M'. Use the sub-matrix K of M' to build the decoding matrix D. Then the recipient can use D to locate the error by simply multiplying the received word Z by the matrix D; the resulting vector is either the all-zeroes vector (if no transmission-error occurred) or the kth row of the D-matrix (if a transmission-error occurred in the kth position). Using this, the recipient can locate the error.

Why does this work? Recall that Z is either Y or $Y + \langle k \rangle$, where $Y = XM$. Since the row-space of M is the same as the row-space of M' (the IK-form of M), the encoded message $Y = XM$ is in the row-space of M' as well; say $Y = X'M'$. Then $YD = (X'M')D = X'(M'D)$. But $M'D$ is the zero-matrix, as in our earlier discussion. Hence YD is the zero-vector. Thus ZD is the zero-vector if no transmission-error occurred or else it's $\langle k \rangle D$ (the kth row of D) if a transmission-error occurred in the kth position. Since all the rows of k are distinct, the recipient can figure out the value of k and, by flipping the kth bit of Z, recover Y.

In such a code one cannot recover X from Y by reading off the message bits directly. However, if we keep track of the basis vectors which get the coding matrix into IK form, the message will be that linear combination of the basis vectors defined by the message bits of the corrected coded message.

Example. Suppose our code matrix is

$$\begin{array}{lcccccccc}
c(\langle 1\rangle) & = & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
c(\langle 2\rangle) & = & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
c(\langle 3\rangle) & = & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
c(\langle 4\rangle) & = & 0 & 0 & 0 & 1 & 1 & 0 & 1
\end{array}$$

If we replace the third row by the sum of the third and fourth, and then the second by the sum of the second and new third, and finally, replace the first by the sum of the first, new second and fourth, we obtain the new matrix in IK form:

$$\begin{array}{lcccccccc}
c(\langle 1\rangle + \langle 2\rangle + \langle 3\rangle) & = & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
c(\langle 2\rangle + \langle 3\rangle + \langle 4\rangle) & = & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
c(\langle 3\rangle + \langle 4\rangle) & = & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
c(\langle 4\rangle) & = & 0 & 0 & 0 & 1 & 1 & 0 & 1
\end{array}$$

We get the D-matrix

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

1100101

Now suppose the sender encodes the message $X = 1001$ using the coding matrix M, and sends the resulting codeword $Y = 1100101$, which the recipient receives as $Z = 1100111$ (an error has occurred in the sixth position). The recipient multiplies this vector by D, obtaining the vector 010, which is the sixth row of D. The recipient deduces that there was an error in the sixth position, and so infers that Y was 1100101. To recover X from Y, the recipient first writes this string as a combination of rows of M' (which is easy to do, since M' is in IK-form): Y is the sum of the first two rows of M'. But the first row of M' is $c(\langle 1\rangle) + c(\langle 2\rangle) + c(\langle 3\rangle)$, while the second row of M' is $c(\langle 2\rangle) + c(\langle 3\rangle) + c(\langle 4\rangle)$. Adding mod 2, we find that Y' is $c(\langle 1\rangle) + c(\langle 4\rangle)$, which (by linearity) is $c(\langle 1\rangle + \langle 4\rangle)$, and this tells the recipient that the sender's original message was $\langle 1\rangle + \langle 4\rangle$, or 1001.

In general the problem of getting back X from $c(X)$ when $c(X) = XM$ is that of inverting the matrix M. Since M has more columns than rows, one can truncate it to a square matrix M' (so long as the rows of M' remain linearly independent) and invert M'. In fact, the standard way to invert a matrix is to use elementary row operations to make it into an identity matrix, and to perform the same row operations starting with an identity matrix. This is exactly the procedure of putting M into IK form. Decoding by taking the linear combination of the rows of this matrix given by the message bits of $c(X)$ is the act of multiplying $c(X)$ by the inverse of a square part of M.

Another question we can address here is: Suppose we allow ourselves exactly k check bits. How large can m be so that we can find a single-error-correcting linear code that can handle m message bits using code words of length only $N = m + k$?

Since k is the length of a row of D, and since no row of D can vanish, and all N rows of D must be distinct, it follows that N can be no more than $2^k - 1$. Thus in a single-error-correcting code with k check bits, m can be no more than $2^k - 1 - k$. Notice that in a code with exactly this last number of check bits (as in our examples), every one of the 2^N possible received words is within distance 1 of some code word: every possible result for ZD is interpretable as a single-bit error (or no error at all). A code with these properties is said to be a *perfect* single-error-correcting code. Perfect d-error-correcting codes have every possible received word within distance d of some code word (and minimum distance between words of $2d + 1$).

Actually, the preceding result applies to non-linear codes as well. Any single-error-correcting code with m message bits and N code word bits must satisfy $2^N \geq 2^m + N \cdot 2^m$. That is because each of the 2^m code words, in the presence of 1 bit of noise, can become any of N different binary strings, all of which must be different from one another and from the 2^m code words. Dividing by 2^m, and using $k = N - m$, we get $2^k \geq 1 + N$, or $N \leq 2^k - 1$. Perfect codes achieve this bound exactly: $N = 2^k - 1$.

Putting it differently: a code is d-error-correcting if every string of length N is within Hamming distance d of *at most* one code word. It is *perfect* if moreover every string of length N is within Hamming distance d of *exactly* one code word.

We can construct single-error-correcting perfect codes without too much trouble. Perfect codes that can correct more than one error are much more difficult to find, and are few and far between.

Exercises.

1. Form a perfect single-error-correcting code with $N = 15$. Find the appropriate decoding matrix D for it, and decode (010101010101010) using it.

2. Consider the 5-message-bit code whose code matrix has five rows that are the vector (110010000) and its first four cyclic shifts. (That is, its first row is (110010000),