MATH3401

Error-correcting Codes

Epiphany term 2021/22

Sam Fearn

Department of Mathematical Sciences, Durham University

The notes for this term were originally written by Dr. Sophy Darwin
and are based on a previous course developed by

Rob de Jeu, Sophy Darwin, Fredrik Strömberg, and Emilie Dufresne

Please send questions, comments or corrections to
`s.m.fearn@durham.ac.uk`

15th January 2022

Contents

Chapter 1

Basic Coding Theory

Error-correcting codes use very abstract mathematics in very concrete applications. If we try to communicate information over some "channel" (e.g., by radio transmission, by storing data on tape and retrieving it later, or by writing music on a CD and playing it later) then there is usually a chance that some errors occur: what comes out of the channel is not identical to what went in. Various strategies have been developed to help with this, one of which is the theory of error-correcting codes. Using such a code, one hopes to decode any information in such a way that the errors that occurred in transmission are corrected and the original information is retrieved.

This is the basic situation:

$$\boxed{SENDER} \xrightarrow{\text{message}} \boxed{ENCODER} \xrightarrow{\text{codeword}} \boxed{CHANNEL} \xrightarrow{\substack{\text{received}\\ \text{word}}} \boxed{DECODER} \xrightarrow{\substack{\text{decoded}\\ \text{message}}} \boxed{USER}$$

And here is an example of what might happen:

$$\boxed{\substack{\text{Yes}\\ \text{or No}}} \xrightarrow{\text{Yes}} \boxed{\substack{\text{Yes} \to 000\\ \text{No} \to 111}} \xrightarrow{000} \boxed{\substack{NOISE\\ \hookrightarrow 100}} \xrightarrow{100} \boxed{100 \sim 000 \to Yes} \xrightarrow{\text{Yes}} \boxed{USER}$$

The decoder does two things: first ($\sim$) it makes a good guess as to what codeword was sent, and then ($\to$) converts this back to a message. We shall be mostly concerned with the $\sim$ process, and how to encode so that $\sim$ works well.

There is no very good name for $\sim$. It is often called "decoding", but this should really include $\to$ as well. We can also call it "error-correction", but this is not quite right either, as we can never be *sure* we have found the original codeword - only that we *probably* have.

1.1 First Definitions

Definition 1.1. An **alphabet** is a finite set of symbols. If A is an alphabet, then A^n is the set of all lists of n symbols from A. (So $|A^n| = |A|^n$.) We call these lists **words** of **length** n. A **code** C of **block length** n on alphabet A is a subset of A^n. A **codeword** is an element of the code.

Example 1. If the alphabet $A = \{0, 1\}$, then $A^3 =$

$\{000, 001, 010, 011, 100, 101, 110, 111\}$. Above, we used $C_1 = \{000, 111\} \subseteq A^3$.

Now suppose $C_2 = \{000, 110, 101, 011\} \subseteq A^3$.

We might have: $\xrightarrow{000} \boxed{\substack{\mathit{NOISE}\\ 001\hookleftarrow}} \xrightarrow{001} \boxed{001 \sim \left\{\begin{matrix}000\\101\\011\end{matrix}\right\}\ ?} \xrightarrow{???}$

We detect an error, but it is not clear how to correct it. △

Definition 1.2. If $|A| = 2$ then C is a **binary** code.
If $|A| = 3$ then C is a **ternary** code.
If $|A| = q$ then C is a q-**ary** code. (We usually use $A = \{0, 1, 2, \ldots, q-1\}$.)

Definition 1.3. For some alphabet A, let x and y be words in A^n. The **Hamming distance** between x and y, written $d(x, y)$, is the number of places in which x and y differ. So $d(x, y)$ is also the (minimum) number of changes of a symbol needed to turn x into y. If x was transmitted, but y is received, then $d(x, y)$ **symbol-errors** have occurred.

Example 2. If $x = 0102$ and $y = 2111$ in $\{0, 1, 2\}^4$, then $d(x, y) = 3$. △

Formally d is a function, $d : A^n \times A^n \longrightarrow \{0, 1, 2, \ldots\}$. We call it a distance because in certain important ways it behaves like ordinary Euclidean distance, measured between two points in $\mathbb{R}^n$. In fact, because of properties ii), iii) and iv) of the following proposition, d qualifies as a 'metric'.

Proposition 1.4. *For words x and y of length n, the Hamming distance $d(x, y)$ satisfies:*

i) $0 \leq d(x, y) \leq n$

ii) $d(x, y) = 0 \Leftrightarrow x = y$

iii) $d(x, y) = d(y, x)$

iv) $d(x, y) \leq d(x, z) + d(z, y)$

Proof. The first three are obvious. For iv), the triangle inequality, we use the second meaning of $d(x, y)$: the RHS is

$$\begin{matrix}\text{the number of changes}\\ \text{required to turn } x \text{ into } z\end{matrix} + \begin{matrix}\text{the number of changes}\\ \text{required to turn } z \text{ into } y.\end{matrix}$$

All these changes would certainly change x into y, so the RHS must be at least the minimum number of changes to do so, which is $d(x, y)$. □

Definition 1.5. For a code C, its **minimum distance** $d(C)$ is
$\min\{d(x, y) \mid x \in C, y \in C, x \neq y\}$. So $d(C) \in \{1, 2, 3, \ldots\}$
A code of block length n with M codewords and minimum distance d
is called an (n, M, d) code (or sometimes an (n, M) code).

We sometimes also refer to a q-ary (n, M, d) code as an $(n, M, d)_q$ code.

Example 3. C=$\{0001, 2200, 0031\} \subseteq \{0, 1, 2, \ldots, 6\}^4$ is a 7-ary (4,3,1) code. △

Definition 1.6. If $C \subseteq A^n$ is a code, and x is a word in A^n, then a **nearest neighbour** of x is a codeword $c \in C$ such that $d(x, c) = \min\{d(x, y) \mid y \in C\}$. A word may have several nearest neighbours. A codeword's nearest neighbour is itself.

Example 4. If $C = \{000, 111, 110, 011\} \subseteq \{0, 1\}^3$, and $x = 100$, then $d(x, 000) = 1$, $d(x, 111) = 2$, $d(x, 110) = 1$, $d(x, 011) = 3$. So x has two nearest neighbours, 000 and 110. △

1.2 Nearest-Neighbour Decoding

In this course, we shall be using **nearest-neighbour decoding**: if a word x is received, we shall decode it to a nearest neighbour of x in our code C. This can always be done by finding $d(x, c)$ for every $c \in C$, though soon we'll have better methods.

Example 5. Let C_1 be our original (3,2,3) code, $C = \{000, 111\} \subseteq \{0, 1\}^3$. Then we would decode 000, 100, 010, and 001 to 000. We would decode 111, 110, 101, and 011 to 111. △

Example 6. Let C_2 be the (2,2,2) code $C_2 = \{00, 11\} \subseteq \{0, 1\}^2$. Then clearly we would decode 00 to 00, and 11 to 11. But 01 and 10 each have two nearest neighbours, both 00 and 11. △

So we can deal with C_2 in two different ways:

- We decide which nearest neighbour to use, for example 01 to 00, 10 to 11.
  Or perhaps both 01 and 10 go to 00. Both of these are nearest-neighbour decoding. (Later, our algorithm for finding a nearest neighbour may decide this for us.)
- "Incomplete decoding": we do not decode 10 and 01 at all. Possibly we ask for retransmission.

Notation: The "floor function" $\lfloor x \rfloor$ means the largest integer $\leq x$.
So $\lfloor 3.7 \rfloor = 3$, $\lfloor 6 \rfloor = 6$, $\lfloor -1/2 \rfloor = -1$.

Proposition 1.7. *For a code with minimum distance d, if a word has:*

i) $\leq d - 1$ *symbol-errors, we will detect that it has some errors.*

ii) $\leq \lfloor \frac{d-1}{2} \rfloor$ *symbol-errors, nearest-neighbour decoding will correct them.*

Notice that, even with more symbol-errors than this, we *may* be able to detect or correct. But this is our guaranteed minimum performance.

Proof. Suppose codeword c is sent, but $t > 0$ symbol-errors occur, and and word x is received. So $d(c, x) = t$.

i) If c' is another codeword, we know $d(c, c') \geq d$. So if $0 < t = d(c, x) \leq d - 1$, then x is not a codeword. We notice this, so we detect that symbol-errors have occurred (though we cannot be sure which symbols have been affected).

ii) We must show that, if $t \leq \lfloor \frac{d-1}{2} \rfloor$, then c is the *unique* nearest neighbour of x; that is, if c' is any other codeword, then $d(x, c) < d(x, c')$. Suppose not. Then $d(x, c') \leq d(x, c) \leq \lfloor \frac{d-1}{2} \rfloor$. But then by the triangle inequality we have

$$d(c, c') \leq d(c, x) + d(x, c') \leq 2 \left\lfloor \frac{d-1}{2} \right\rfloor \leq d - 1.$$

This contradicts that d was minimum distance. □

Draw your own pictures for these proofs: For i), you need a circle of radius $d - 1$ centred on c. For ii), draw the "suppose" part: a triangle with vertices c, x, and c'.

Informally, we often say the code C "detects" or "corrects" so many symbol-errors, when we really mean that our decoding procedure, used with this code, detects or corrects them.

Example 7. Let C have $d = 5$. Then $\lfloor \frac{d-1}{2} \rfloor = 2$, and so this code can detect up to 4 symbol errors and correct up to 2 symbol errors. △

Example 8. The code $C_2 = \{00, 11\} \subseteq \{0, 1\}^2$ has $d = 2$. So it detects up to 2 - 1 = 1 symbol-errors, but corrects $\lfloor \frac{2-1}{2} \rfloor = 0$, as we found.

The code $C_1 = \{000, 111\} \subseteq \{0, 1\}^3$ has $d = 3$. So it detects up to 3 - 1 = 2 symbol-errors, but corrects $\lfloor \frac{3-1}{2} \rfloor = 1$. Suppose that $c = 000$ is sent, but we receive $x = 101$, so we have 2 symbol-errors. Then we detect that symbol-errors have occurred, because x is not a codeword. But nearest-neighbour decoding gives 111, so we fail to correct them. △

1.3 Probabilities

So far we have tacitly assumed that a codeword is more likely to suffer a small number of symbol-errors, than a larger number. This is why Proposition 1.7, which talks about being able to detect or correct symbol-errors *up to* a certain number, is useful. Now we must make our assumptions explicit, and calculate the probabilities of different outcomes. We start by defining a certain kind of channel, in which all symbol-errors are equally likely.

Definition 1.8. A **q-ary symmetric channel with symbol-error probability** p is a channel for a q-ary alphabet A such that:

i) For any $a \in A$, the chance that it is changed in the channel is p.

ii) For any $a, b \in A, a \neq b$, the chance that a is changed to b in the channel, written $P(b \text{ received} \mid a \text{ sent})$, is $\frac{p}{q-1}$.

Symmetric channels are easy to work with, but many real-life channels are not strictly symmetric. Part ii) of the definition says that p, the chance of change, is split equally among all $q-1$ other symbols: each wrong symbol is equally likely. So the symbol 6 would be equally likely to become 5, 0, or 9. But in fact 5 and 7 are more likely if someone is typing, 0 if copying by hand, and 9 if arranging plastic numbers on the fridge!

Part i) says that the chance of change is not affected by which symbol is sent, or its position in the codeword, or which other symbols are nearby, or whether other symbols are changed. So for example, in a symmetric channel, 12234 is equally likely to become 12334 or 12134. In fact, if someone tries to remember or copy 12234, 12334 (repeating the wrong symbol) is much more likely than 12134. Similarly, with two symbol-errors occurring, in a symmetric channel 12234 is equally likely to become 12243 or 14233. But for human error, 12243 (swapping two adjacent symbols) is the more likely.

(For many types of mechanical channel, also, 12234 is more likely to become 12243 than 14233, but this is because the physical cause of a symbol-error (a scratch on a CD, or a surge of electricity, for example) is likely also to affect adjacent symbols. So symbol-errors in the last two positions is more likely than symbol-errors in the second position and the last. There are ways to adapt nearest-neighbour decoding to help with the fact that, in real life, symbol-errors may tend to come in "bursts".)

In the language of probability, i) implies that in a symmetric channel, symbol-errors in different positions are *independent* events. This makes for easy calculations.

Proposition 1.9. *Let c be a codeword in a q-ary code of block-length n, sent over a q-ary symmetric channel with symbol-error probability p. Then*

$$P(x \text{ received} \mid c \text{ sent}) = \left(\frac{p}{q-1}\right)^t (1-p)^{n-t}, \quad \text{where} \quad t = d(c,x).$$

Proof. First we note that, from the definition of of a symmetric channel, the chance of a symbol remaining correct is $1-p$.

To change c to x, the channel must make the "right" change in each of the "right" t positions. From Definition 1.8, the chance of each of these events is $\frac{p}{q-1}$. Since they are independent, the chance of all of them occurring is $\left(\frac{p}{q-1}\right)^t$. But the symbols in the other $n-t$ positions must remain correct, and the chance of this is $(1-p)^{n-t}$. Again using independence, we multiply to get the result. □

In the case $t=0$, we have $x=c$. So the chance of c being correctly received is $(1-p)^n$.

Example 9. Using the code $C_2 = \{000, 111\} \subseteq \{0,1\}^3$, so $q=2$, suppose 000 is sent. Then the chance of receiving 001 is $p(1-p)^2$. There is the same chance of receiving 010. In fact, we can complete the following table:

x	$t = d(000, x)$	chance 000 received as x	chance if $p = 0.01$	n-n decodes correctly?
000	0	$(1-p)^3$	0.970299	yes
100	1	$p(1-p)^2$	0.009801	yes
010				
001				
110	2	$p^2(1-p)$	0.000099	no
101				
011				
111	3	p^3	0.000001	no

△

Note that nearest-neighbour decoding corrects 000, 100, 010, and 001 all to 000. So, if $p = .01$, then the chance of success is:

$$\begin{aligned} P(\text{we decode back to 000}) &= P(\text{we receive 000, 001, 010, or 100}) \\ &= 0.99^3 + 3 \times 0.01 \times 0.99^2 = 0.999702 \end{aligned}$$

We can also see in the table that because $p < 1 - p$, smaller $d(000, x)$ gives a larger chance. A closer x is more likely to be received. More generally, we have:

Corollary 1.10. *If $p < (q-1)/q$ then $P(x \text{ received} \mid c \text{ sent})$ increases as $d(x, c)$ decreases.*

Before we prove this, consider a channel so noisy that all information is lost: whatever symbol is sent, there is an equal chance, $1/q$, of each symbol being received. In a 'random' channel like this, the chance that a symbol is changed is $(q-1)/q$. So the corollary is considering channels which are better than random.

Proof. If $p < (q-1)/q$ then it is easy to show that $1-p > 1/q$ (the chance that a symbol remains unchanged is more than random), and also that $p/(q-1) < 1/q$ (the chance that a specified wrong symbol is received is less than random). Putting these together, we have $(1-p) > p/(q-1)$ (so the most likely symbol to be received is the correct one) and it follow that $(1-p)^{n-t}\left(\frac{p}{q-1}\right)^t$ is larger for smaller t. □

Example 10. For the (3,3,3) code $C = \{111, 202, 020\} \subseteq \{0, 1, 2\}^3$, there are more words to consider. Suppose we send codeword $c = 111$ through a ternary symmetric channel with symbol-error probability p. The following table shows the probability of different $d(x, c)$, when $p = 1/4$ and when $p = 1/2$, calculated using Proposition 1.9:

			$p = 1/4$		$p = 1/2$	
$t =$ $d(x, c)$	ex.s of such x	# of such x	for each such x, prob. received	prob. of this t	for each such x, prob. received	prob. of this t
0	111	1	$\left(1 - \frac{1}{4}\right)^3$ $= \frac{27}{64}$	$\frac{27}{64}$	$\left(1 - \frac{1}{2}\right)^3$ $= \frac{1}{8}$	$\frac{1}{8}$
1	110 112 101 $\vdots$	6 $= \binom{3}{1} \times 2$	$\left(1 - \frac{1}{4}\right)^2 \cdot \frac{1/4}{2}$ $= \frac{9}{128}$	$\frac{27}{64}$	$\left(1 - \frac{1}{2}\right)^2 \cdot \frac{1/2}{2}$ $= \frac{1}{16}$	$\frac{3}{8}$
2	100 102 120 122 $\vdots$	12 $= \binom{3}{2} \times 2^2$	$\left(1 - \frac{1}{4}\right)\left(\frac{1/4}{2}\right)^2$ $= \frac{3}{256}$	$\frac{9}{64}$	$\left(1 - \frac{1}{2}\right)\left(\frac{1/2}{2}\right)^2$ $= \frac{1}{32}$	$\frac{3}{8}$
3	000 002 $\vdots$ 222	8 $= \binom{3}{3} \times 2^3$	$\left(\frac{1/4}{2}\right)^3$ $= \frac{1}{512}$	$\frac{1}{64}$	$\left(\frac{1/2}{2}\right)^3$ $= \frac{1}{64}$	$\frac{1}{8}$

Both $p = 1/4$ and $p = 1/2$ are quite large, but we have $p < (3-1)/3$, so Corollary 1.10 applies, and we can see the probabilities increase as we go up the fourth or sixth column.

As $d(C)$ is 3 , C can correct one symbol-error and detect two. A word with ≥ 2 symbol-errors may have the the wrong nearest neighbour (102 has nearest neighbour 202), or it may have several nearest neighbours (100 has all three codeword as nearest neighbours).

So suppose now we only decode when the nearest neighbour is unique. Then for $d(x, c) = 0$ or 1 we will always decode correctly. For $d(x, c) = 2$ or 3 we will sometimes decode incorrectly (e.g.102 to 202), and sometime not at all (e.g. 100). So, $P(x \text{ correctly decoded}) = P(d(x, c) = 0 \text{ or } 1)$. Thus, for $p = 1/4$, $P(x \text{ correctly decoded}) = \frac{27}{64} + \frac{27}{64} = \frac{27}{32}$, and for $p = 1/2$ this is $\frac{1}{8} + \frac{3}{8} = \frac{1}{2}$.

How does this compare to using the trivial (1, 3, 1) code $C_0 = \{0, 1, 2\}$? Here we simply send one symbol and the chance of it being received correctly is $1 - p$. For $p = 1/4$, $\frac{27}{32} > \frac{3}{4}$, so C with nearest-neighbour decoding is a little more reliable than C_0. But for $p = 1/2$ we gain nothing. △

Both Proposition 1.9 and Corollary 1.10 are from the sender's point of view: they assume we know which codeword c was sent, and tell us about $P(x \text{ received} \mid c \text{ sent})$ for different possible xs. But the receivers know only that word x has arrived. What *they* want to know is, which word is most likely to have been sent? They must compare, for all codewords c, $P(c \text{ sent} \mid x \text{ received})$. The following proposition, closely related to Cor. 1.4, may seem obvious. To prove it, we use Bayes' theorem, which allows us to 'reverse' the conditional probabilities.

Proposition 1.11. *Suppose that a q-ary (n, M, d) code C is sent over a q-ary symmetric channel with symbol-error probability p, where $p < (q-1)/q$, and each codeword $c \in C$ is equally likely to be sent. Then for any word x received,* $P(c \text{ sent} \mid x \text{ received})$ *increases as $d(x, c)$ decreases.*

Proof.

$$\begin{aligned} P(c \text{ sent} \mid x \text{ received}) &= \frac{P(c \text{ sent and } x \text{ received})}{P(x \text{ received})} \\ &= \frac{P(c \text{ sent})P(x \text{ received} \mid c \text{ sent})}{P(x \text{ received})} \\ &= \frac{P(x \text{ received} \mid c \text{ sent})}{M \cdot P(x \text{ received})}, \end{aligned}$$

since we assumed that for any $c \in C$, $P(c \text{ sent}) = 1/M$.

Also, by the law of total probability (also known as the partition theorem), if $C = \{c_1, c_2, \ldots, c_M\}$, then $P(x \text{ received}) = \sum_{i=1}^{M} P(c_i \text{ sent})P(x \text{ received} \mid c_i \text{ sent})$. This is independent of the c which was sent, since we sum over all possible sent codewords.

Now the receiver knows x, and is considering different possible c's. But in

$$P(c \text{ sent} \mid x \text{ received}) = \frac{P(x \text{ received} \mid c \text{ sent})}{M \cdot P(x \text{ received})},$$

the denominator is independent of c. Hence $P(c \text{ sent} \mid x \text{ received})$ increases as $P(x \text{ received} \mid c \text{ sent})$ increases; that is, by Cor. 1.4, as $d(x, c)$ decreases. □

So the nearest neighbours of x are indeed the most likely codewords to have been sent. If we have the conditions described in Proposition 1.11, we are justified in using nearest-neighbour decoding.

(The assumption that each codeword is equally likely to be sent is important. If this were not the case, it would obviously affect our conclusions. This is like the well-known problem in medical testing for rare diseases, when a false positive may be more likely than an actual case.)

1.4 Bounds on Codes

What makes a good (n, M, d) code? Small n will make transmission faster. Large M will provide many words, to convey many different messages. Large d will allow us to detect and correct more symbol-errors, and so make communication more reliable. But these parameters are related, so we have to make trade-offs. The following is known as the **Singleton Bound**.

Proposition 1.12. *For a q-ary (n,M,d) code, we have $M \leq q^{n-d+1}$.*

Thus, for fixed q, small n and large d will make M small. This makes sense intuitively: small n makes the space of possible words small, and large d makes the codewords far apart. So we can't fit in very many of them! The proof involves a 'projection' map which simply 'forgets' the last $d-1$ symbols of the codeword. (You draw the picture.)

Proof. Let $C \subseteq A^n$, where $|A| = q$. For $d = 1$ the proposition is trivial. For $d > 1$, define a map $f : A^n \to A^{n-d+1}$ by $f(a_1a_2\ldots a_n) = a_1a_2\ldots a_{n-d+1}$. Clearly f is not injective on A^n: if two words x and y differ only in the last position, then $f(x) = f(y)$. But if x and y are distinct codewords in C, they must differ in $\geq d$ positions. So f cannot 'forget' all these differences, so $f(x) \neq f(y)$. Thus f is injective on C, and it follows that $|f(C)| = |C|$. But $f(C) \subseteq A^{n-d+1}$, so

$$M = |C| = |f(C)| \leq |A^{n-d+1}| = q^{n-d+1}$$

as required. □

A code which saturates the Singleton bound is known as *Maximum Distance Separable* or MDS.

Example 11. Let C_n be the 'binary repetition code' of block length n,

$$C_n := \{\overbrace{00\ldots0}^{n}, \overbrace{11\ldots1}^{n}\} \subset \{0,1\}^n.$$

C_n is a $(n,2,n)_2$ code, and since $2 = 2^{n-n+1}$, C_n is an MDS code. △

Definition 1.13. Let A be an alphabet, $|A| = q$. Let $n \geq 1$ and $0 \leq t \leq n$ be integers, and x a word in A^n. Then

i) The **sphere of radius t around x** is $S(x,t) = \{y \in A^n \mid d(y,x) \leq t\}$.

ii) A code $C \subseteq A^n$ is **perfect** if there is some t such that A^n is the disjoint union of all the $S(c,t)$ as c runs through C.

Because of the ' $\leq$', $S(c,t)$ is like a solid ball around c, not just the surface of a sphere. (Of course, we use the Hamming distance, not ordinary Euclidean distance.) In a perfect code, the $S(c,t)$ partition A^n. Thus any word $x \in A^n$ is in exactly one $S(c,t)$, and that c is x's unique nearest neighbour.

Example 12. For $C_1 = \{000, 111\} \subseteq \{0,1\}^3$, we have $S(000,1) = \{000, 100, 010, 001\}$ and $S(111,1) = \{111, 011, 101, 110\}$. These are disjoint, and $S(000,1) \cup S(111,1) = \{0,1\}^3$. So C_1 is perfect. (You should draw a picture, with the words of A^n labelling the vertices of a cube in the obvious way. Or even better, make a model.) △

Example 13. Let $C_2 = \{111, 020, 202\} \subseteq \{0,1,2\}^3$. Then, for example, $S(111,1) = \{111, 110, 112, 101, 121, 011, 211\}$. Is C_2 perfect? No, because for all $c \in C$, we have $d(c, 012) = 2$. So 012 is not in any $S(c,1)$, but is in every $S(c,2)$. Thus for $t = 0$ or 1 the $S(c,t)$ do not cover all of $\{0,1,2\}^3$, and for $t = 2$ or 3 they are not disjoint. △

To decide whether a code is perfect or not, we may not need to consider the actual codewords. We can look first at the sizes of spheres in the space. As we would expect, the size of a sphere in A^n depends only on its 'radius', t, not on its centre.

Lemma 1.14. *If $|A| = q$, $n \geq 1$, and $x \in A^n$, then*

$$|S(x,t)| = \sum_{k=0}^{t} \binom{n}{k} (q-1)^k.$$

Proof. How many $y \in A^n$ have $d(x,y) = k$? To make such a y from x, we must first choose k positions to change: $\binom{n}{k}$ ways to do this. Then for each chosen position, we choose one of the $q-1$ other symbols: $(q-1)^k$ ways. So there are $\binom{n}{k}(q-1)^k$ such y. Now we build up the sphere in layers, by letting k go from 0 to t. □

Example 14. For the code C_2 above, we have, $q = 3$, $n = 3$. So $|S(x,1)| = \binom{3}{0} + \binom{3}{1}(3-1) = 1+6 = 7$, as we saw, and $|S(x,2)| = \binom{3}{0} + \binom{3}{1}(3-1) + \binom{3}{2}(3-1)^2 = 1+6+12 = 19$. Since $|\{0,1,2\}^3| = 27$, and neither 7 nor 19 divides 27, clearly this space cannot be partitioned by spheres of either size. Three spheres containing 7 words each cannot fill the space, and three containing 19 must overlap, just as we saw by considering the word 012.

Of course, $|S(x,3)| = 27$, and $|S(x,0)|$ is always 1, and these do divide 27. But to use these spheres to make a perfect code, we would have to have, respectively, just one codeword, or $C = A^n$. These 'trivial' codes are no use to us. △

We can use spheres to give us another bound on the size of a code. This is known as the **Hamming Bound** or the **Sphere-packing Bound**:

Proposition 1.15. *A q-ary (n, M, d) code satisfies:*

$$M \cdot \sum_{k=0}^{t} \binom{n}{k} (q-1)^k \leq q^n, \quad \text{where } t = \left\lfloor \frac{d-1}{2} \right\rfloor.$$

Notice that we are relating the radius of the sphere to the minimum distance of the code; in fact, we must have $d = 2t+1$ or $2t+2$. The first part of the proof is really the same as that of the second part of Proposition 1.7. The notation is different, but the picture is the same.

Proof. Let the code be $C \subseteq A^n$. First we must show that the $S(c,t)$ for codewords $c \in C$ are disjoint. Suppose not, so there is some $x \in A^n$ such that $x \in S(c_1,t)$ and $x \in S(c_2,t)$, where $c_1 \neq c_2$. This means that $d(x,c_1)$ and $d(x,c_2)$ are both $\leq t$. So by the triangle inequality we have $d(c_1,c_2) \leq d(x,c_1) + d(x,c_2) \leq 2t$. But this contradicts $d(c_1,c_2) \geq d(C) \geq 2t+1$.

Now

$$\bigcup_{c \in C} S(c,t) \subseteq A^n, \quad \text{so} \quad \left| \bigcup_{c \in C} S(c,t) \right| \leq q^n.$$

But since the $S(c,t)$ are disjoint, we have

$$\left|\bigcup_{c\in C} S(c,t)\right| = \sum_{c\in C} |S(c,t)| = M|S(c,t)|.$$

Thus $M|S(c,t)| \leq q^n$, which by Lemma 1.7 proves the proposition. □

We have equality in the Hamming Bound if and only if the code is perfect; in this case the spheres $S(c,t)$, which are as large as they can be without overlapping, will fill A^n. Thus for a perfect code, M must divide q^n, and so must $|S(c,t)| = \sum_{k=0}^{t} \binom{n}{k}(q-1)^k$ for some t. We used this idea in the example above. It is also not hard to show that a perfect code must have d odd (see Q16).

Chapter 2

Linear Codes

In Chapter 1, we used $0, 1, 2, \ldots$ purely as symbols in an alphabet A, and our code could be any subset of A^n. To make progress we now want to do arithmetic with our symbols, so our alphabet must be a field F. Then we can regard our words in A^n as vectors in F^n, which is a vector space over F. Moreover, we shall require that our code C is a subspace of F^n.

2.1 Finite Fields

A field is a set with two binary operations, which obey the standard rules of arithmetic.

Definition 2.1. A non-empty set F with addition $F \times F \to F$, mapping (a, b) to $a + b$, and multiplication $F \times F \to F$, mapping (a, b) to ab, is called a **field** if the following axioms hold.

i) Associativity of addition: For every a, b, and c in F, $(a + b) + c = a + (b + c)$

ii) Additive identity: There exists 0 in F such that for every $a \in F$, $a + 0 = a = 0 + a$

iii) Additive inverse: For every $a \in F$, there exists $b \in F$ such that $a + b = 0 = b + a$

iv) Commutative addition: For every a and b in F, $a + b = b + a$

v) Associativity of multiplication: For every a, b, and c in F, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

vi) Multiplicative identity: There exists 1 in F such that, for every $a \in F$, $1 \cdot a = a = a \cdot 1$

vii) Multiplicative inverse: For every $a \in F$, $a \neq 0$, there exists $b \in F$ such that $a \cdot b = 1 = b \cdot a$

viii) Commutative multiplication: For every a and b in F, $a \cdot b = b \cdot a$

ix) Distributivity: For every a, b, and c in F, $(a+b) \cdot c = a \cdot c + b \cdot c$ and $a \cdot (b+c) = a \cdot b + a \cdot c$

x) Multiplicative and additive inverses are different: $0 \neq 1$

Axioms i - iv make $(F, +)$ into an abelian (or commutative) group; axioms v - viii make $(F - \{0\}, \cdot)$ into another abelian group; axioms i - vi & ix make $(F, +, \cdot)$ a ring. Note that the definitions of addition and multiplication as maps $F \times F \to F$ imply that these operations are closed. It is easy to show that the identities 0 and 1 are unique, and that for each $a \in F$, the additive inverse $-a$ and the multiplicative inverse a^{-1} are unique. We also have some convenient notations: for m a non-negative integer, we write $m \cdot a$ for adding, and a^m for multiplying, m copies of a. It's all very familiar.

However, the fields you know best are the reals $\mathbb{R}$, the complex numbers $\mathbb{C}$, and the rationals $\mathbb{Q}$, and these are no good as alphabets, because they are infinite.

To find finite fields, we start from arithmetic modulo n. Here the set $\mathbb{Z}/n$ (or $\mathbb{Z}_n$, or $\mathbb{Z}/n\mathbb{Z}$) is the congruence classes $\{[0]_n, [1]_n, \ldots, [n-1]_n\}$ (or $\{\overline{0}, \overline{1}, \ldots, \overline{n-1}\}$), and we add or multiply by using any representative of that class. It is easy to check that every axiom except for vii will hold for any n. But vii holds (that is, $\mathbb{Z}/n$ has multiplicative inverses for all non-zero elements) if and only if n is prime.

For n prime we have a field, and to make this clear (and also to be able to use n for block-length, as in Chapter 1) we shall write $\mathbb{F}_p$ instead of $\mathbb{Z}/n$. Once we have specified p, we can write simply $0, 1, 2, \ldots p-1$ rather than $[0]_p, [1]_p, \ldots, [p-1]_p$ (or $\overline{0}, \overline{1}, \ldots, \overline{p-1}$).

Are there finite fields with a non-prime number q of elements? The answer is yes if and only if q is a prime power, $q = p^r, r \in \mathbb{Z}_+$. But of course $\mathbb{F}_{3^2} = \mathbb{F}_9 \neq \mathbb{Z}/9$, because $\mathbb{Z}/9$ is not a field. We shall construct and use such non-prime fields $\mathbb{F}_q$ in Chapter **??**.

The notation $\mathbb{F}_q$ is justified, because it can be shown that any two fields with the same number of elements are isomorphic. For general statements we shall call our field $\mathbb{F}_q$, but *until we reach Chapter* **??** *q will always be prime* (that is, $r = 1$). This allows us to keep 'p' for the symbol-error probability.

2.2 Finite Vector Spaces

Just as $\mathbb{R}^n$ is a vector space over $\mathbb{R}$, $\mathbb{F}_q^n = (\mathbb{F}_q)^n$ is a vector space over $\mathbb{F}_q$. Everything you have learned about vector spaces (and most of your ideas about $\mathbb{R}^n$) will still work. These notes gives only a quick, informal reminder of the main definitions and ideas from linear algebra which we shall use. Formal definitions, results and examples will be written in terms of finite fields, and spaces and codes over them. The main difference is that we can now count the vectors in a space: to start with, $|\mathbb{F}_q^n| = q^n$. We can even write a complete list of them.

We shall still sometimes call our vectors 'words', and some of them will be our codewords. Because words (in English) are horizontal, we will usually write vectors as rows (rather than columns): $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{F}_q^n$, where the x_i are in $\mathbb{F}_q$. You need to know which field $\mathbb{F}_q$ you are working over, because all arithmetic must be done mod q.

Example 15. The vectors $\mathbf{x} = (0, 1, 2, 0)$ and $\mathbf{y} = (1, 1, 1, 1)$ could be vectors in $\mathbb{F}_3^4$, but they could also be vectors in $\mathbb{F}_7^4$. In $\mathbb{F}_3^4$, we would have $\mathbf{x} + \mathbf{y} = (1, 2, 0, 1)$, and $2\mathbf{x} = (0, 2, 1, 0)$. But in $\mathbb{F}_7^4$, it would be $\mathbf{x} + \mathbf{y} = (1, 2, 3, 1)$, $2\mathbf{x} = (0, 2, 4, 0)$ and $4\mathbf{x} = (0, 4, 1, 0)$. △

In Chapter 1, a code could be any subset of A^n. But we now make a more restrictive definition.

Definition 2.2. A **linear code** is a subspace of the vector space $\mathbb{F}_q^n$, for some finite field $\mathbb{F}_q$ and non-negative integer n.

Recall that a **subspace** of a vector space is a subset which is closed under vector addition and scalar multiplication. Thus, if we are given a subset of $\mathbb{F}_q^n$ as a list, it is straightforward (if tedious) to check whether it is a linear code or not.

Example 16. Let $\mathbf{x} = (0, 1, 2, 0)$, and $\mathbf{y} = (1, 1, 1, 1,)$, $\mathbf{z} = (0, 2, 1, 0)$ and $\mathbf{0} = (0, 0, 0, 0)$ be vectors in $\mathbb{F}_3^4$. Then $C_1 = \{\mathbf{x}, \mathbf{y}\}$ is not a linear code since $\mathbf{x} + \mathbf{y} = (1, 2, 0, 1) \notin C_1$. But $C_2 = \{\mathbf{x}, \mathbf{z}, \mathbf{0}\}$ is a linear code, as adding any combination of these vectors (which also includes multiplying them by 0, 1 or 2) gives one of them. △

If the subset S is not closed, we can keep adding in vectors as necessary until it is. The resulting space is the **span** of the set, written $\langle S\rangle$, and is by construction a linear code.

Example 17. The span of C_1 is the linear code $C_3 = \langle C_1\rangle = \langle\{\mathbf{x}, \mathbf{y}\}\rangle =$

$$\{(0,0,0,0),(0,1,2,0),(0,2,1,0),(1,1,1,1),(1,2,0,1),(1,0,2,1),(2,2,2,2),(2,0,1,2),(2,1,0,2)\}.$$

We could also notice that $C_2 = \{0\mathbf{x}, 1\mathbf{x}, 2\mathbf{x}\} = \langle\{\mathbf{x}\}\rangle$, so in fact $\langle C_2\rangle = C_2$. △

If $\langle S\rangle = C$ then we say S is a **spanning set** for C. There are usually many possible spanning set for a subspace.

A set of vectors $S = \{\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_k}\}$ in $\mathbb{F}_q^n$ is **linearly independent** if and only if no non-trivial linear combination of them equals the zero-vector $\mathbf{0}$; that is, for $\lambda_i \in \mathbb{F}_q$, iff:

$$\lambda_1\mathbf{x_1} + \lambda_2\mathbf{x_2} + \ldots + \lambda_k\mathbf{x_k} = \mathbf{0} \implies \lambda_1 = \lambda_2 = \ldots = \lambda_k = 0.$$

A linearly independent spanning set for a linear code C is a **basis** for C. While there may still be many possible bases, these will all have the same number of vectors, and this number is the **dimension** of C, written $\dim(C)$.

Example 18. The spanning sets of the code listed above, $C_3 \subseteq \mathbb{F}_3^4$, include
$S_1 = \{(0, 1, 2, 0), (1, 1, 1, 1)\}$,
$S_2 = \{(0, 1, 2, 0), (2, 2, 2, 2)\}$, and
$S_3 = \{(0, 1, 2, 0), (1, 1, 1, 1), (2, 0, 1, 2)\}$.

S_3 is not a basis, because $1(0, 1, 2, 0) + 2(1, 1, 1, 1) + 2(2, 0, 1, 2) = (6, 3, 6, 6) = (0, 0, 0, 0)$. But both S_1 and S_2 are linearly independent sets, and thus bases for C_3. So $\dim(C_3) = 2$.

To prove the linear independence of S_1, we can note that if $\lambda_1(0, 1, 2, 0) + \lambda_2(1, 1, 1, 1) = (0, 0, 0, 0)$, then by the first position $\lambda_2 = 0$, and so then by the second position $\lambda_1 = 0$ also.

△

Example 19. For the whole space $\mathbb{F}_q^n$, the 'standard basis' is $B = \{\mathbf{e_1}, \mathbf{e_2}, \ldots \mathbf{e_k}\}$, where e_i has 1 in the i^{th} position and 0 elsewhere. $\triangle$

A basis B for a linear code C is "just right" for making every vector $\mathbf{c} \in C$ as a linear combination of vectors from B: a spanning set can make each $\mathbf{c}$ *at least* one way, a linearly independent set can make each $\mathbf{c}$ *at most* one way, but a basis can make each $\mathbf{c}$ *exactly* one way. We use this property to prove the following:

Proposition 2.3. *If a linear code $C \subseteq \mathbb{F}_q^n$ has dimension k, then $|C| = q^k$.*

Proof. Let $B = \{\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_k}\}$ be any basis for C. There is a one-to-one correspondence between codewords $\mathbf{c} \in C$ and linear combinations $\lambda_1\mathbf{x_1} + \lambda_2\mathbf{x_2} + \ldots + \lambda_k\mathbf{x_k}$, with $\lambda_i \in \mathbb{F}_q$. Each λ_i can take any of q values. $\square$

Example 20. For $C_3 \subseteq \mathbb{F}_3^4$ listed above, $\dim(C_3) = 2$, and $|C_3| = 9 = 3^2$. $\triangle$

We can now update our (n, M, d) notation for the parameters of a code:

Definition 2.4. A q-ary $[n, k, d]$ code is a linear code, a subspace of $\mathbb{F}_q^n$ of dimension k with minimum distance d.

The square or round brackets prevent ambiguity: any q-ary $[n, k, d]$ code is also a q-ary (n, q^k, d) code, but not vice-versa. From now on, almost all codes will be linear, so I will write "code" for "linear code"

2.3 Array Decoding

The zero element 0 plays a very special role in $\mathbb{F}_q$, and so does the zero vector $\mathbf{0}$ in $\mathbb{F}_q^n$. We are also interested in how many entries of a general vector $\mathbf{x} \in \mathbb{F}_q^n$ are (or are not) zero.

Definition 2.5. For $\mathbf{x} \in \mathbb{F}_q^n$, the **weight** of $\mathbf{x}$, written $w(\mathbf{x})$, is the number of non-zero entries in $\mathbf{x}$.

Weights are closely related to Hamming distances. To show this, we must first define the difference between two vectors, using several properties of our vector space. Notice that by axioms vi and iii any field has a -1, the additive inverse of 1. (For q prime we could also write this as $q - 1$.) Then since we can multiply vectors by scalars, we can write $-\mathbf{y}$ for $-1 \cdot \mathbf{y}$, and $\mathbf{x} - \mathbf{y}$ for $\mathbf{x} + (-\mathbf{y})$.

Lemma 2.6. *For $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{F}_q^n$, we have $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} - \mathbf{y})$.*

Proof. The vector $\mathbf{x} - \mathbf{y}$ has non-zero entries exactly where $\mathbf{x}$ and $\mathbf{y}$ differ. $\square$

So any Hamming distance can be written as a weight. But also, since $w(\mathbf{x}) = w(\mathbf{x} - \mathbf{0}) = d(\mathbf{x}, \mathbf{0})$, any weight can be written as a Hamming distance. This allows us to prove the following useful fact:

Proposition 2.7. *For the code $C \subseteq \mathbb{F}_q^n$, $d(C)$ is the minimum weight of any non-zero codeword in C.*

Proof. First we define two sets of non-negative integers:

$$W = \{w(\mathbf{x}) \mid \mathbf{x} \in C, \mathbf{x} \neq \mathbf{0}\} \text{ and } D = \{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\}.$$

Then the proposition says that $\min(D) = \min(W)$. We can show more: that $D = W$. For any $w(\mathbf{x}) \in W$, we know that both $\mathbf{x}$ and $\mathbf{0}$ are in C, so $w(\mathbf{x}) = d(\mathbf{x}, \mathbf{0}) \in D$. Conversely, for any $d(\mathbf{x}, \mathbf{y}) \in D$, we know $\mathbf{x}$ and $\mathbf{y}$ are both in C. Because C is a subspace, $\mathbf{x} - \mathbf{y}$ must also be in C, so $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} - \mathbf{y}) \in W$. □

This proposition means that to find $d(C)$ for a linear code with q^k words, we need to consider only q^k weights , rather than $\binom{q^k}{2} = \frac{q^k(q^k-1)}{2}$ distances.

Example 21. For code C_3 listed in Section 2.2, we can see that $d(C) = 2$, without finding $\binom{9}{2} = 36$ distances. But note that C_3 can also be written as $\langle\{(1,1,1,1),(1,2,0,1)\}\rangle$; $d(C)$ is not always obvious from a basis. △

For linear codes we have a much better way to talk about errors.

Definition 2.8. Suppose that a codeword $\mathbf{c} \in C \subseteq \mathbb{F}_q^n$ is sent, and $\mathbf{y} \in \mathbb{F}_q^n$ is received. Then the **error-vector** is $\mathbf{e} = \mathbf{y} - \mathbf{c}$.

We can think of the channel as adding $\mathbf{e}$ to $\mathbf{c}$, and the decoding process aims to subtract it again. But of course the receiver does not know which error-vector was added, and can only choose a *likely* error-vector to subtract. Which error-vectors are likely? We know that $d(\mathbf{y}, \mathbf{c}) = w(\mathbf{e})$; this is the number of symbol-errors which $\mathbf{c}$ has suffered. This allows us to re-write Propositions 1.9 and 1.11 for linear codes.

Proposition 2.9. *Let the code $C \subseteq \mathbb{F}_q^n$, be sent over a q-ary symmetric channel with symbol-error probability p. For any $\mathbf{c} \in C$, $\mathbf{y} \in \mathbb{F}_q^n$, and $\mathbf{e} = \mathbf{y} - \mathbf{c}$,*

$$P(\mathbf{y} \text{ received} \mid \mathbf{c} \text{ sent}) = P(\mathbf{e} \text{ added in channel}) = \left(\frac{p}{q-1}\right)^{w(\mathbf{e})} (1-p)^{n-w(\mathbf{e})}.$$

If also $p < (q-1)/q$, and each codeword $c \in C$ is equally likely to be sent, then for any given $\mathbf{y}$ $P(\mathbf{c} \text{ sent} \mid \mathbf{y} \text{ received})$ increases as $w(\mathbf{e})$ decreases.

The most likely error-vectors are those of least weight, or in other words those involving the fewest symbol-errors. So we should still use nearest-neighbour decoding: for a linear code, given a received word $\mathbf{y}$ we must find a codeword $\mathbf{c}$ such that $w(\mathbf{e}) = d(\mathbf{y}, \mathbf{c})$ is as small as possible; as before this will be one of the most likely codewords to have been sent. We could do this by calculating $\mathbf{e}_i = \mathbf{y} - \mathbf{c}_i$ for each $c_i \in C$, and then comparing all the $w(\mathbf{e}_i)$. But it is much more efficient to make an array, as follows. (This is sometimes called a 'Slepian' or 'Standard' array.)

Algorithm: Array Decoding
Let C be a code of dimension k in $\mathbb{F}_q^n$. We construct an array as follows:

1. Write the q^k codewords as the top row, with $\mathbf{0}$ in the first column.
2. Consider the vectors of $\mathbb{F}_q^n$ which are not yet in the array, and choose one of lowest available weight, $\mathbf{e}$.
3. Write $\mathbf{e}$ into the first column, and then complete this new row by adding $\mathbf{e}$ to each codeword in the top row.
4. If the array has q^{n-k} rows, then STOP. Otherwise, go to 2.

Now, decode any received word $\mathbf{y}$ to the codeword at the top of its column.

Example 22. Let C be the $[4,2,2]$ code $\langle(1,1,0,0),(0,0,1,1)\rangle \subseteq \mathbb{F}_2^4$. Then one possible array is:

$(0,0,0,0)$	$(1,1,0,0)$	$(0,0,1,1)$	$(1,1,1,1)$
$(1,0,0,0)$	$(0,1,0,0)$	$(1,0,1,1)$	$(0,1,1,1)$
$(0,0,1,0)$	$(1,1,1,0)$	$(0,0,0,1)$	$(1,1,0,1)$
$(1,0,1,0)$	$(0,1,1,0)$	$(1,0,0,1)$	$(0,1,0,1)$

This array decodes (0,0,0,1) to (0,0,1,1), and (0,1,1,0) to (1,1,0,0). In each case the word is decoded to a nearest neighbour, though this nearest neighbour is not unique. △

For this method to be well defined, we require that every possible vector in $\mathbb{F}_q^n$ appears exactly once in the array. (See Q22) We should also prove the following:

Proposition 2.10. *Array decoding is nearest-neighbour decoding.*

Proof. Suppose that our received word $\mathbf{y}$ is in the same row as $\mathbf{e}_1$ in the first column, and in the same column as codeword $\mathbf{c}_1$ in the top row. So we decode $\mathbf{y}$ to $\mathbf{c}_1$; that is, we assume that error-vector $\mathbf{e}_1$ was added in the channel, and now subtract it.

$\mathbf{0}$	$\mathbf{c}_1$
$\mathbf{e}_1$	$\mathbf{y}$

We can show by contradiction that $\mathbf{c}_1$ is a nearest neighbour of $\mathbf{y}$. Suppose not, that is, there is some $\mathbf{c}_2$ such that $\mathbf{y} = \mathbf{c}_2 + \mathbf{e}_2$, with $w(\mathbf{e}_2) < w(\mathbf{e}_1)$. Then we have $\mathbf{c}_2 + \mathbf{e}_2 = \mathbf{y} = \mathbf{c}_1 + \mathbf{e}_1$, so $\mathbf{e}_2 = \mathbf{e}_1 + (\mathbf{c}_1 - \mathbf{c}_2)$. Since $\mathbf{c}_1 - \mathbf{c}_2$ must be a codeword in the top row, $\mathbf{e}_2$ is in the same row as $\mathbf{e}_1$, so it is not in any row higher up. Thus when $\mathbf{e}_1$ was picked by step 2. of the algorithm, $\mathbf{e}_2$ was not yet in the array, it was available, so $\mathbf{e}_1$ did not have least possible weight. Contradiction. □

As we know, nearest-neighbour decoding does not always find the right codeword. If we use an array, what is the probability that, after transmission and array decoding, the receivers will have the correct word, the one that was sent? Effectively, decoding with an array subtracts one of the vectors $\mathbf{e}$ in the first column from the received word y. Thus decoding will be successful if and only if the channel added one of these vectors.

Proposition 2.11. *Let C be a code $\subseteq \mathbb{F}_q^n$, sent over a q-ary symmetric channel with symbol-error probability p. Suppose the decoding array has α_i vectors of weight i in its first column. Then for any codeword c sent, the chance that it is successfully decoded is*

$$\sum_{i=0}^{n} \alpha_i \left(\frac{p}{q-1}\right)^i (1-p)^{n-i}.$$

Example 23. For the array above, with $q = 2$, we have $\alpha_0 = 1,\ \alpha_1 = 2,\ \alpha_2 = 1,\ \alpha_3 = 0,\ \alpha_4 = 0$. So the chance of successful decoding is $(1-p)^4 + 2p(1-p)^3 + p^2((1-p)^2$. △

Proof. The chance of successful decoding is the chance that one of the error-vectors in the first column occurred; since these are disjoint possibilities, we add their individual probabilities. (We include the zero error-vector - that is, the possibility that the codeword is received correctly.) □

Steps 1 and 2 of the algorithm involve choice, so that many different arrays could be made for the same code. In general, some of these arrays would decode some received words differently. However, for a perfect code, all arrays will perform identical decoding. These ideas are explored in more detail in the homework (Q25-27).