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`

7th February 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: $\xrightarrow{000} \boxed{\begin{smallmatrix}\mathit{NOISE}\\ 001\hookleftarrow\end{smallmatrix}} \xrightarrow{001} \boxed{001 \sim \left\{\begin{matrix}000\\101\\011\end{matrix}\right\} \;?} \xrightarrow{???}$

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

the number of changes required to turn x into z + the number of changes required to turn z into y.

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*

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

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				
010	1	$p(1-p)^2$	0.009801	yes
001				
110				
101	2	$p^2(1-p)$	0.000099	no
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:

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

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

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 $\{[0]_n,[1]_n,\dots,[n-1]_n\}$ (or $\{\overline{0},\overline{1},\dots,\overline{n-1}\}$), 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 $0,1,2,\dots p-1$ rather than $[0]_p,[1]_p,\dots,[p-1]_p$ (or $\overline{0},\overline{1},\dots,\overline{p-1}$).

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): $\mathbf{x}=(x_1,x_2,\dots,x_n)\in\mathbb{F}_q^n$, 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 $4\mathbf{x}=(0,4,1,0)$. $\triangle$

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 $B = \{\mathbf{e_1}, \mathbf{e_2}, \dots \mathbf{e_k}\}$, where e_i has 1 in the i^{th} position and 0 elsewhere. △

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

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

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. □

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. $\triangle$

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. $\square$

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).

Chapter 3

Codes as Images

So far we have only two ways to specify a (linear) code: we can list the codewords, or we can give a spanning set, which might also be a basis. But since codes are vector subspaces, we can also describe them as the image or the kernel of a suitable mapping between spaces. It turns out that these mappings are also helpful for the encoding and decoding processes.

3.1 Mappings and Matrices

Let $\mathbb{F}_q^k$ and $\mathbb{F}_q^n$ be two vector spaces, over the same field $\mathbb{F}_q$ but of possibly different dimensions, and let $f : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ be a linear mapping between them. This means that f preserves the linear structure: if $\mathbf{x}$ and $\mathbf{y}$ are vectors in $\mathbb{F}_q^k$, and λ is a scalar in $\mathbb{F}_q$, we have $f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x})+f(\mathbf{y})$ and $f(\lambda\mathbf{x}) = \lambda f(\mathbf{x})$. Then we know that we can perform the mapping f by multiplying by a suitable matrix A, with entries from $\mathbb{F}_q$. We are writing our vectors as rows: $\mathbf{x} \in \mathbb{F}_q^k$ is a $1 \times k$ 'matrix', and $f(\mathbf{x}) \in \mathbb{F}_q^n$ is $1 \times n$. So we need A to be $k \times n$; we can write that $A \in M_{k,n}(\mathbb{F}_q)$. And we must multiply $\mathbf{x}$ by A on the *right*.

Example 24. Let $f : \mathbb{F}_5^3 \longrightarrow \mathbb{F}_5^4$ such that $f(\mathbf{x}) = \mathbf{x}A$, where $A = \begin{pmatrix} 1 & 0 & 1 & 3 \\ 3 & 2 & 0 & 1 \\ 4 & 2 & 1 & 4 \end{pmatrix} \in$ $M_{3,4}(\mathbb{F}_5)$. Then, for example, if $\mathbf{x} = (0, 1, 2)$,

$$f(\mathbf{x}) = \mathbf{x}A = (0, 1, 2) \begin{pmatrix} 1 & 0 & 1 & 3 \\ 3 & 2 & 0 & 1 \\ 4 & 2 & 1 & 4 \end{pmatrix} = (1, 1, 2, 4).$$

△

This may look unfamiliar. In Linear Algebra 1 you mostly wrote vectors as columns, and multiplied by A on the left. Of course, we could still do it that way, by taking the transpose of everything: if $\mathbf{x}A = \mathbf{y}$, then also $A^t\mathbf{x}^t = (\mathbf{x}A)^t = y^t$. But coding theory always uses row vectors, and you will get used to it!

The **image** of the mapping f is all the vectors in $\mathbb{F}_q^n$ which can be written as $f(\mathbf{x})$ for some $\mathbf{x}$ in $\mathbb{F}_q^k$:

$$\mathrm{im}(f) = f(\mathbb{F}_q^k) = \{f(\mathbf{x}) \mid \mathbf{x} \in \mathbb{F}_q^k\} = \{\mathbf{x}A \mid \mathbf{x} \in \mathbb{F}_q^k\} \subseteq \mathbb{F}_q^n$$

(Draw your own"fried egg" diagram.) Let us regard the rows of A as vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{F}_q^n$. Then since $\mathbf{x} = (x_1, \ldots, x_n)$ can be any vector in $\mathbb{F}_q^k$, $\mathrm{im}(f)$ is all the linear combinations $x_1\mathbf{a}_1 + \cdots + x_k\mathbf{a}_k$ of the $\mathbf{a}_i$. This is the span of the $\mathbf{a}_i$, so we can say that $\mathrm{Im}(f) = \langle\{\mathbf{a}_1, \ldots, \mathbf{a}_k\}\rangle$, or that the rows of A are a spanning set for the image.

3.2 Generator-matrices

If the rows of A are also linearly independent, then they are a basis for $\mathrm{Im}(f)$, and k, the number of rows, is the dimension of this image.

Definition 3.1. For some matrix $G \in M_{k,n}(\mathbb{F}_q)$, let $f : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ be defined by $f(\mathbf{x}) = \mathbf{x}G$, and let the linear code $C = \mathrm{im}(f) = \{\mathbf{x}G \mid \mathbf{x} \in \mathbb{F}_q^k\} \subseteq \mathbb{F}_q^n$. Then if G has linearly independent rows, G is a **generator-matrix** for C.

So if we are given a code as a span, $C = \langle\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}\rangle$, how can we find a generator-matrix for C? We can easily make the matrix A with rows $\mathbf{a}_i$, but we still need to determine whether these rows are linearly independent. If they are not, we must find another, smaller set of vectors which are independent, but span the same subspace of $\mathbb{F}_q^n$.

As you know, this can be done by row-reducing the matrix A. This process, which takes linear combinations of the rows, does not change their span. Of course, the row-reduction must be done in $\mathbb{F}_q$, but this is easier than in $\mathbb{R}$. The numbers stay small, and you never need to subtract or divide: just use inverses, and add or multiply.

We obtain a matrix in reduced row echelon form (RREF). If there are any rows of zeros at the bottom, we remove them. Let the resulting matrix be B, with rows $\mathbf{b}_1, \ldots \mathbf{b}_k$, $k \leq m$, and consider the two possible cases:

- **If the rows of A were independent**, there were no rows of zeros, and $k = m$. In this case both A and B correspond to maps from $\mathbb{F}_q^k$ to $\mathbb{F}_q^n$; $f_A(\mathbf{x}) = \mathbf{x}A$ and $f_B(\mathbf{x}) = \mathbf{x}B$. Their images are the same, so we can let

  $$C = \mathrm{im}(f_A) = \mathrm{im}(f_B) = \{\mathbf{x}A \mid \mathbf{x} \in \mathbb{F}_q^k\} = \{\mathbf{x}B \mid \mathbf{x} \in \mathbb{F}_q^k\}.$$

  (But of course in general, for a given $\mathbf{x}$, $\mathbf{x}A \neq \mathbf{x}B$.)

  Moreover, since the rows of A are linearly independent, we know that

  $$(x_1, \ldots, x_k)\begin{pmatrix} - & \mathbf{a}_1 & - \\ & \vdots & \\ - & \mathbf{a}_k & - \end{pmatrix} = x_1\mathbf{a}_1 + \cdots + x_k\mathbf{a}_k = \mathbf{0} \Longrightarrow x_1 = \ldots = x_k = 0.$$

  Sof_A is injective, and $\dim(\mathrm{im}(f_A)) = \dim(\mathbb{F}_q^k) = k$. (We know this already as the $\mathbf{a}_i$ are a basis for $\mathrm{im}(f_A)$.) All this is equally true for B. Both A and B are generator matrices for C.

- **If the rows of A were dependent**, we had to remove at least one row of zeros, and $k < m$. Then for B we can say, as above, that $f_B : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ is injective, and $\dim(\mathrm{im}(f_B)) = k$. But $f_A : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^n$ is not injective: We know there are some λ_i, not all zero, such that $\lambda_1\mathbf{a}_1 + \cdots + \lambda_m\mathbf{a}_m = \mathbf{0}$. Then if $\mathbf{x} = (\lambda_1, \ldots, \lambda_m)$, we have $\mathbf{x} \neq \mathbf{0}$ but $\mathbf{x}A = \mathbf{0}$. The map f_A maps a larger space onto a smaller, dimension m to dimension k. In this case, B is a generator matrix for $C = \mathrm{im}(f_B)$, but A is not.

Thus a generator-matrix not only specifies a code, it also immediately tells us its dimension and so its size.

Example 25. Let $C = \langle\{(0,0,3,1,4), (2,4,1,4,0), (5,3,0,1,6)\}\rangle \subseteq F_7^5$. To find $|C|$, and a generator matrix for C, we make A, and row-reduce:

$$A = \begin{pmatrix} 2 & 4 & 1 & 4 & 0 \\ 5 & 3 & 0 & 1 & 6 \\ 0 & 0 & 3 & 1 & 4 \end{pmatrix} \xrightarrow{A_{12}} \begin{pmatrix} 2 & 4 & 1 & 4 & 0 \\ 0 & 0 & 1 & 5 & 6 \\ 0 & 0 & 3 & 1 & 4 \end{pmatrix} \xrightarrow{M_1(4)} \begin{pmatrix} 1 & 2 & 4 & 2 & 0 \\ 0 & 0 & 1 & 5 & 6 \\ 0 & 0 & 3 & 1 & 4 \end{pmatrix} \xrightarrow{A_{21}(3), A_{23}(4)} \begin{pmatrix} 1 & 2 & 0 & 3 & 4 \\ 0 & 0 & 1 & 5 & 6 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Thus $B = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 \\ 0 & 0 & 1 & 5 & 6 \end{pmatrix}$ is a generator matrix for C , and $|C| = 7^2 = 49$.

$\triangle$

3.3 Encoding and Channel Decoding

Recall the basic situation described in Chapter 1. The generator-matrix provides a very easy way to encode messages to codewords: we let our messages be vectors in $\mathbb{F}_q^k$, and then multiply by the generator-matrix G to to obtain codewords in $C \subseteq \mathbb{F}_q^n$.

$$\begin{tikzcd} \boxed{\text{SENDER}} \arrow[r, "\text{message}", "\mathbf{x}\in\mathbb{F}_q^k"'] & \boxed{\begin{matrix}\text{ENCODER}\\ \mathbf{c}=\mathbf{x}G\end{matrix}} \arrow[r, "\text{codeword}", "\mathbf{c}\in C\subseteq\mathbb{F}_q^n"'] & \boxed{\text{CHANNEL}} \end{tikzcd}$$

It is crucial that the rows of G are independent, so that f_G is injective; otherwise, two different messages would have the same codeword. Also, if the code is to enable us to correct (or even detect) any errors, then there must be some words in $\mathbb{F}_q^n$ but not in C, so k must be strictly $< n$. Thus a generator-matrix is always 'landscape' in shape.

The codeword $\mathbf{c}$ then travels through the channel, where it may or may not be changed, and is received as $\mathbf{y} \in \mathbb{F}_q^n$. The receivers do their best to correct any error, finding a codeword $\mathbf{c}'$ which is the (or one of the) most likely to have been sent. (In Chapter 4 we will look again at this process.) And finally, they must find the message corresponding to this codeword, $\mathbf{x}'$ such that $\mathbf{x}'G = \mathbf{c}'$. This last process is called "channel decoding", to distinguish it from the "decoding" which attempts to correct errors. (In the first pictures in Chapter 1, '$\sim$' is decoding, and '$\to$' is channel decoding.) So at the receiver's end we have:

$$\begin{tikzcd} \boxed{\text{CHANNEL}} \arrow[r, "\text{received word}", "\mathbf{y}\in\mathbb{F}_q^n"'] & \boxed{\begin{matrix}\text{DECODER}\\ \text{error correction?}\end{matrix}} \arrow[r, "\text{codeword}", "\mathbf{c}'\in C"'] & \boxed{\text{CHANNEL DECODER}} \arrow[r, "\text{message?}", "\mathbf{x}'\in\mathbb{F}_q^k"'] & \boxed{\mathit{USER}} \end{tikzcd}$$

But how can we find $\mathbf{x}'$ from $\mathbf{c}'$? Our G is not invertible; it is not even square. Let $\mathbf{x}' = (x'_1, \dots x'_k)$, $\mathbf{c}' = (c'_1, \dots c'_n)$, and G have *columns* $\mathbf{g}_1, \dots \mathbf{g}_n$. Then

$$\mathbf{x}'G = (x_1, \dots, x_k) \begin{pmatrix} | & & | \\ \mathbf{g}_1 & \cdots & \mathbf{g}_n \\ | & & | \end{pmatrix} = (c'_1, \dots, c'_n) = \mathbf{c}'$$

So, using the usual dot product (or scalar product), we have n equations $\mathbf{x}' \cdot \mathbf{g}_i = c'_i$, each in k unknowns, the x_i. It is because $\mathbf{c}'$ is a codeword that these equations are consistent and we can find such an $\mathbf{x}'$, and because the map $f_G : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n$ is injective that there is only one solution.

You can solve these equations by any method you like; you might prefer to think about $G^t(\mathbf{x}')^t = (\mathbf{c}')^t$, and row-reduce the augmented matrix $(G^t \mid (\mathbf{c}')^t)$. But by picking the right generator-matrix for C in the first place, we can make this much easier. Suppose that G is in RREF. Then G has k columns, $\mathbf{g}_{i_1}, \dots, \mathbf{g}_{i_k}$, which have a leading 1 and zeros elsewhere. Each of these picks out one coordinate of $\mathbf{x}'$, so $c'_{i_j} = \mathbf{x}' \cdot \mathbf{g}_{i_j} = x'_j$. So in this case we can read off $\mathbf{x}'$ from $\mathbf{c}'$ without any calculation.

Example 26. Let $C \subseteq \mathbb{F}_7^5$ have generator matrix $G = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 \\ 0 & 0 & 1 & 5 & 6 \end{pmatrix}$, and suppose we have corrected some received word to the codeword (6,5,3,5,0). Then to channel decode, we must find $\mathbf{x}' = (x'_i, x'_2)$ such that

$$(x'_i, x'_2) \begin{pmatrix} 1 & 2 & 0 & 3 & 4 \\ 0 & 0 & 1 & 5 & 6 \end{pmatrix} = (6, 5, 3, 5, 0).$$

Clearly $\mathbf{x}' = (6, 3)$. △

3.4 Equivalence and Standard Form

Sometimes two codes have different codewords, but are the same in all the ways that matter.

Definition 3.2. Two codes C_1, C_2 are *equivalent* if we can transform one to the other by applying a sequence of changes of two kinds to all the codewords:

i) permuting the n positions.

ii) in a particular position, permuting the $|A| = q$ symbols.

Proposition 3.3. *Two codes which are equivalent have the same parameters* (n, M, d).

Definition 3.4. Two linear codes $\subseteq \mathbb{F}_q^n$ are **permutation equivalent** if we can transform one to the other by applying a sequence of permutations to all of the codewords.

Definition 3.5. Two linear codes $\subseteq \mathbb{F}_q^n$ are *monomially equivalent* if we can transform one to the other by applying a sequence of permutations to all of the codewords of the following form:

i) permuting the n positions.

ii) in a particular position, multiplying by $\lambda \in \mathbb{F}_q, \lambda \neq 0$.

Clearly if two codes are permutation equivalent, then they are monomially equivalent, and we will simply never need to use a transformation of type ii) from the definition of monomial equivalence. Also, since $\mathbb{F}_q$ is a field, multiplying by $\lambda \neq 0$ will permute the elements of $\mathbb{F}_q$, but not all permutations of $\mathbb{F}_q$ can be obtained in this way. So if two codes are monomially equivalent, then they are also equivalent, but not vice-versa.

A change of either type is a bijective map $\pi : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^n$; it just permutes the vectors of $\mathbb{F}_q^n$. Let us write Π for a sequence of such changes, so $\Pi = \pi_s \circ \cdots \circ \pi_1$. If there exists such a Π with $\Pi(C_1) = C_2$, then C_1 and C_2 are equivalent, and we write $C_1 \equiv C_2$ if the type of equivalence being considered is clear.

It is also quite easy to see that any such π will preserve the linear structure: $\pi(\mathbf{x}+\mathbf{y}) = \pi(\mathbf{x})+\pi(\mathbf{y})$ and $\pi(\lambda\mathbf{x}) = \lambda\pi(\mathbf{x})$, and so the same is true for any Π. Since C_1 and C_2 must have the same dimension (as they are both linear codes over the same field with the same number of codewords), then as a consequence of the rank-nullity theorem it follows that $\Pi(C_1) = C_2$ if and only if Π changes a basis for C_1 into a basis for C_2. We can therefore define equivalence in terms of generator-matrices.

Definition 3.6. An $m \times m$ matrix $P \in M_{m,m}(\mathbb{Z}_2)$ is a *permutation matrix* if it has a single 1 in each row and column, and zeros elsewhere. Note that any permutation can be written as a permutation matrix of an appropriate size.

Those of you studying Representation Theory III will probably recognise such matrices as those found when considering representations of the symmetric group.

Proposition 3.7. *Let C_1 and C_2 be codes in $\mathbb{F}_q^n$ with generator-matrices G_1 and G_2 respectively. Then if $G_2 = G_1P$ for some permutation matrix P, then C_1 and C_2 are permutation equivalent.*

Example 27. Let $C_1, C_2 \subseteq \mathbb{F}_q^3$, and let C_1 have generator matrix G_1 where

$$G_1 = \begin{pmatrix} | & | & | \\ \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 \\ | & | & | \end{pmatrix}.$$

If

$$G_2 = \begin{pmatrix} | & | & | \\ \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 \\ | & | & | \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} | & | & | \\ \mathbf{g}_3 & \mathbf{g}_1 & \mathbf{g}_2 \\ | & | & | \end{pmatrix},$$

is a generator matrix for C_2, then by linearity every $c_2 \in C_2$ is the image of a $c_1 \in C_1$ under the permutation given in cycle form as (123). $\triangle$

Definition 3.8. An $m \times m$ matrix $M \in M_{m,m}(\mathbb{F}_q)$ is a *monomial matrix* if it has exactly one non-zero element in each row and column.

A monomial matrix M can always be written as $M = DP$ or $M = PD'$, for P a permutation matrix, and D, D' diagonal matrices. We call P the *permutation part* and D, D' the *diagonal part* of M respectively.

Example 28.

$$\underbrace{\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}}_{P} = \underbrace{\begin{pmatrix} 0 & 2 & 0 \\ 0 & 0 & 3 \\ 1 & 0 & 0 \end{pmatrix}}_{M} = \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}}_{P} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}}_{D'}$$

If we apply M to the generator matrix G_1 of a code $C_1 \subseteq \mathbb{F}_5^3$, as $G_2 = GM$, then this consists of first multiplying all codewords by 2 in the first position, and by three in the second position, and then by applying the permutation (123). Equivalently, we could apply the permutation first, and then multiply all words by 2 in the second position and by 3 in the third position.

The matrix G_2 is then the generator matrix for a monomially equivalent code to C_1. $\triangle$

Proposition 3.9. *Let C_1 and C_2 be codes in $\mathbb{F}_q^n$ with generator-matrices G_1 and G_2 respectively. Then if $G_2 = G_1M$ for some monomial matrix M, then C_1 and C_2 are monomially equivalent.*

Note that Propositions 3.7 and 3.9 do not say "if and only if". This is because most codes have many possible generator-matrices: if C_1 and C_2 are linearly equivalent, there must be a Π taking G_1 to *some* generator-matrix for C_2 - but it might not be G_2. However, if C_1 and C_2 are equivalent, and C_1 has generator matrix G_1, then $G_2 = G_1H$ (for H either a monomial matrix or permutation matrix as appropriate) is a generator matrix for G_2.

If we let S be the set of all codes of length n over $\mathbb{F}_q$, then both permutation equivalence and monomial equivalence are *equivalence relations*, meaning that they satisfy the following conditions:

- $C \sim C, \forall\, C \in S$ (Reflexive)
- If $C_1 \sim C_2$, then $C_2 \sim C_1$ (Symmetric)
- If $C_1 \sim C_2$ and $C_2 \sim C_3$, then $C_1 \sim C_3$ (Transitive)

We can therefore partition the set of all codes of length n over $\mathbb{F}_q$ into equivalence classes (using either permutation equivalence or monomial equivalence). For those of you who have done Algebra II, this is the *orbit* of the code under the symmetric group.

However, there may be some permutations which don't just send the code to another element of its equivalence class, but rather send the code to itself. In the language of algebra, these are stabilisers of the code, and the set of all stabilisers forms a group under composition.

Definition 3.10. The set of permutations sending a code to itself forms a group under composition, with the trivial permutation acting as the identity. This group is known as the *permutation automorphism group* of C, and is written as $\mathrm{PAut}(C)$.

In practice, we may think of these permutations either as matrices, or in their cycle form. The two are equivalent as demonstrated in Example 27.

For a code C of block length n, $\mathrm{PAut}(C)$ is a subgroup of the symmetric group on n symbols S_n, $\mathrm{PAut}(C) \subseteq S_n$.

Aside: If you didn't study Algebra II, you may not have seen the definition of a subgroup, so I'll repeat it here.

Definition. Given a group G, a subset $H \subseteq G$ is said to be a *subgroup* if:

1. The identity $e \in H$,
2. For any $h_1,\ h_2 \in H$, we have $h_1 h_2 \in H$,
3. For any $h \in H$, we have that the inverse $h^{-1} \in H$.

Similarly, the set of all monomial matrices which sends a code to itself also forms a group.

Definition 3.11. The set of monomial matrices sending a code to itself forms a group under composition. This group is known as the *monomial automorphism group* of C, and is written as $\mathrm{MAut}(C)$.

For a binary code we have that $\mathrm{PAut}(C) = \mathrm{MAut}(C)$, but more generally we have $\mathrm{PAut}(C) \subseteq \mathrm{MAut}(C)$.

The idea of equivalence allows us to work with particularly convenient generator-matrices:

Definition 3.12. For $k \leq n$, if a $k \times n$ generator-matrix is of the form $(I_k \mid A)$, we say that it is in **standard form**.

Here I_k is the $k \times k$ identity matrix, and A some $k \times (n-k)$ matrix.

Proposition 3.13. *Any linear code is permutation equivalent to one with a generator-matrix in standard form.*

Proof. For any code $C_1 \subseteq \mathbb{F}_q^n$ we can find a generator-matrix G_1 which is in RREF. Then by permuting columns we can obtain G_2 of form $(I \mid A)$, and this is the generator-matrix of an equivalent code C_2. □

One advantage of standard form is that it makes channel decoding even easier. The first k digits of the codeword are the corresponding message.

Example 29. Let $C_1 \subseteq \mathbb{F}_7^5$ have generator matrix $G_1 = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 \\ 0 & 0 & 1 & 5 & 6 \end{pmatrix}$ as above. By swapping columns 2 and 3, we obtain $G_2 = \begin{pmatrix} 1 & 0 & 2 & 3 & 4 \\ 0 & 1 & 0 & 5 & 6 \end{pmatrix}$, which is in standard form. Then using the equivalent code C_2 which has this generator matrix, a sender would encode message (1,5) as

$$(1,5)\begin{pmatrix} 1 & 0 & 2 & 3 & 4 \\ 0 & 1 & 0 & 5 & 6 \end{pmatrix} = (1,5,2,0,6),$$

and a receiver would channel decode (6,3,5,5,0) to (6,3). △

To end this chapter I will briefly introduce some terminology which we will not stress, but you may meet in other sources.

Suppose we encode using a generator-matrix in standard form. Then the first k digits of the codeword are the message, and the remaining $n - k$ digits, calculated from the message using the A part of the generator-matrix, are sometimes called **check-digits**. Their role is to show up symbol-errors in the message digits: if only a message-digit is changed, the result will not be a codeword. Syndrome decoding (Section 4.3) exploits this kind of idea.

The **rank** of a code is its dimension, k. (This is also the rank, in the linear algebra sense, of its generator-matrix.) Its **redundancy** is $n - k$, the number of check-digits. These are “redundant” in the sense that they are not the message, though they are certainly not useless. And the **rate** of the code is k/n, the fraction of the stream of symbols which is the actual messages. Redundancy and rate are most easily explained for a code with generator-matrix in standard form, but these names can be used for the parameters $n-k$ and k/n of any linear code.

Chapter 4

Codes as Kernels

In $\mathbb{F}_q^n$, just as in $\mathbb{R}^n$, we can calculate the **dot** (or scalar) **product** of two vectors: $\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + \cdots x_n y_n$, and if $\mathbf{x} \cdot \mathbf{y} = 0$ we say that $\mathbf{x}$ and $\mathbf{y}$ are **orthogonal**. (But since we multiply and add mod q, a non-zero vector $\mathbf{x}$ can easily have $\mathbf{x} \cdot \mathbf{x} = 0$, and so be orthogonal to itself. [1])

The **kernel** of a linear map $f : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^m$ is the vectors which it sends to $\mathbf{0}$:
$\ker(f) = \{\mathbf{x} \in \mathbb{F}_q^n \mid f(\mathbf{x}) = \mathbf{0}\}$.

By combining these two ideas we get a new way to specify a code, and to find its minimum distance. We also find a much better algorithm for detecting (and sometimes correcting) errors.

4.1 Dual codes

If C is a code in $\mathbb{F}_q^n$, then 'C dual', written $C^\perp$, is the space of all vectors in $\mathbb{F}_q^n$ which are orthogonal to every codeword in C.

Definition 4.1. Let C be a code in $\mathbb{F}_q^n$. Then its **dual** $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v} \cdot \mathbf{u} = 0 \text{ for all } \mathbf{u} \in C\}$.

But we do not have to check $\mathbf{v}$ against *every* $\mathbf{u}$ in C, one by one.

Proposition 4.2. *If C has generator matrix G, then $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}G^t = \mathbf{0}\}$.*

Proof. The rows of G are a basis for C, say $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$. Then certainly we require $\mathbf{v} \cdot \mathbf{b}_i = 0$ for every $1 \leq i \leq k$. But also, since the dot product is linear in the second input (in fact, in both), then if $\mathbf{u} = u_1\mathbf{b}_1 + \cdots + u_k\mathbf{b}_k$, we have $\mathbf{v} \cdot \mathbf{u} = u_1 \mathbf{v} \cdot \mathbf{b}_1 + \cdots + u_k \mathbf{v} \cdot \mathbf{b}_k$. Thus it is enough to check that $\mathbf{v} \cdot \mathbf{b}_i = 0$ for all the $\mathbf{b}_i$. We can do this by checking that

$$\mathbf{v} \cdot G^t = (v_1, \ldots, v_n) \begin{pmatrix} | & & | \\ \mathbf{b}_1 & \cdots & \mathbf{b}_k \\ | & & | \end{pmatrix} = (0, \ldots, 0) = \mathbf{0}.$$

[1] Thus the dot product is not (generally) an inner product on $\mathbb{F}_q$, so we cannot use $\mathbf{x} \cdot \mathbf{x}$ as a norm, and we do not have any idea of the length of a vector in $\mathbb{F}_q^n$.

□

Multiplying by G^t is of course a linear map $f_{G^t} : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^k$, and the $\mathbf{v}$s we want are exactly $\ker(f_{G^t})$, or the nullspace of G^t. Draw a picture of these spaces and maps: C and $C^\perp$ are both in $\mathbb{F}_q^n$. They will intersect, at least in $\mathbf{0}$. C is the image of the map f_G coming from $\mathbb{F}_q^k$; $C^\perp$ is the kernel of the map f_{G^t} going to $\mathbb{F}_q^k$.

Proposition 4.3. *Let C be a code in $\mathbb{F}_q^n$. Then $C^\perp$ is a code, and if* $\dim(C) = k$*, then* $\dim(C^\perp) = n - k$.

Proof. Since f_{G^t} is a linear map, its kernel is a (linear) subspace, and so a (linear) code. The dimension of the kernel is the 'nullity' of the map, and we know[2] that for the linear map $f_{G^t} : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^k$, we have rank + nullity = $\dim(\mathbb{F}_q^n) = n$. The rank of the map is the row-rank of G^t; in fact row- or column-rank of G or G^t are all four equal to k. So the nullity is $n - k$. □

The 'dual' idea appears in many different areas of mathematics, but it is usually, as in this case, a 'self-inverse' operation:

Proposition 4.4. *For $C \subseteq \mathbb{F}_q^n$, $(C^\perp)^\perp = C$.*

Proof. If C has basis $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ and $C^\perp$ has basis $\{\mathbf{v}_1, \dots, \mathbf{v}_{n-k}\}$, then we know that for any $\mathbf{u}_i$ and $\mathbf{v}_j$ we have $\mathbf{v}_j \cdot \mathbf{u}_i = 0$. But this also shows that every $\mathbf{u}_i \in (C^\perp)^\perp$, so $C \subseteq (C^\perp)^\perp$. By Proposition 4.3 we know that $\dim(C^\perp)^\perp = n - (n-k) = k = \dim(C)$, so they must be equal. □

Suppose C has generator matrix G with rows $\mathbf{u}_1 \dots \mathbf{u}_k$, how can we find out more about $C^\perp$? We would like to find a basis, and thus a generator matrix for it. The vectors in $C^\perp$ are those $\mathbf{v}$ such that $\mathbf{v}G^t = (v_1, \dots, v_n)\begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_k \\ | & & | \end{pmatrix} = \mathbf{0}$. As in Section 3.3, G^t is not invertible, but we can solve the k equations $\mathbf{v} \cdot \mathbf{u}_i = 0$. Again, one way to do this is to take transposes, $(G^t)^t\mathbf{v}^t = G\mathbf{v}^t = \mathbf{0}$, and then row-reduce the augmented matrix $(G \mid \mathbf{0})$. Once we have G in RREF, we can find a basis for $C^\perp$ from the new, simpler equations.

The following algorithm "automates" this process, working straight from G in RREF to the basis for $C^\perp$.

Algorithm: Finding a basis for a dual code
Suppose that C has a generator matrix $G = (g_{ij}) \in M_{k,n}(\mathbb{F}_q)$, and G is in RREF.

- Let $L = \{1 \leq j \leq n \mid G$ has a leading 1 in column $j\}$.
- For each $1 \leq j \leq n$, $j \notin L$, make a vector $\mathbf{v}_j$ as follows:
  * for $m \notin L$: the m^{th} entry of $\mathbf{v}_j$ is 1 if $m = j$, 0 otherwise.
  ** Fill in the other entries of $\mathbf{v}_j$ (left to right) as $-g_{1j}, \dots, -g_{kj}$.
- These $n - k$ vectors $\mathbf{v}_j$ are a basis for $C^\perp$.

[2]Strictly, in Linear Algebra you only proved this for vector spaces over $\mathbb{R}$, but it is true in general.

Example 30. Let C be the code in $\mathbb{F}_5^7$ with generator matrix

$$G = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 1 & 1 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix}$$

To find a basis for $C^\perp$ we first note that G is already in RREF, and the leading 1s are in columns 1, 3, and 6. Thus $L = \{1, 3, 6\}$, and we make vectors for a basis $\{\mathbf{v}_2, \mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_7\}$. Step * fills in the "non-L" entries, so that the incomplete vectors look a bit like a standard basis:

$$\mathbf{v}_2 = (\ \ , 1, \ \ , 0, 0, \ \ , 0) \quad \mathbf{v}_4 = (\ \ , 0, \ \ , 1, 0, \ \ , 0)$$
$$\mathbf{v}_5 = (\ \ , 0, \ \ , 0, 1, \ \ , 0) \quad \mathbf{v}_7 = (\ \ , 0, \ \ , 0, 0, \ \ , 1)$$

Then step ** uses the corresponding columns to complete the vectors. For example, since column 7 is (0,3,4), we complete $\mathbf{v}_7$ with the additive inverses of these: 0, 2, and 1. So we have

$$\mathbf{v}_2 = (3, 1, 0, 0, 0, 0, 0) \quad \mathbf{v}_4 = (2, 0, 4, 1, 0, 0, 0)$$
$$\mathbf{v}_5 = (1, 0, 3, 0, 1, 0, 0) \quad \mathbf{v}_7 = (0, 0, 2, 0, 0, 1, 1)$$

△

Notice that, since G is in RREF, in column j all the entries after the j^{th} will be 0. This is why, in step **, we find that $\mathbf{v}_j$ is all zeros after the j^{th} entry (which is the the 1 from step *).

We will not write out a formal proof that this algorithm works: it is a straightforward calculation but involves a lot of notation. But, having found your $\mathbf{v}_j$, it is easy to check they are indeed a basis: Firstly, step * ensures that each $\mathbf{v}_j$ has a 1 in column j, where all the others have 0, so the vectors are linearly independent. Secondly, to see they are in $\ker(f_{G^t})$, check that each $\mathbf{v}_j G^t = \mathbf{0}$. This shows why we do step **: everything cancels out just right. Since we know that $\dim(\ker(f_{G^t})) = n - k$, this proves we have a basis.

We can now make a generator-matrix H for $C^\perp$, by taking the $\mathbf{v}_j$, in order, as rows. In general, H is not in RREF, but we can row-reduce it if necessary. As in Section 3.4, if G is in standard form, the process is even easier:

Proposition 4.5. *If $C \subseteq \mathbb{F}_q^n$ has generator-matrix $G = (I_k \mid A)$, then a generator-matrix for $C^\perp$ is $H = (-A^t \mid I_{n-k})$.*

Again this is fiddly to prove in general, but becomes obvious with examples; this H is exactly the generator-matrix for $C^\perp$ produced by the algorithm above. Again, H can be row-reduced to RREF, but not necessarily to standard form.

4.2 Check-matrices

In the last section we showed that $C^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}G^t = \mathbf{0}\}$, where G is a generator-matrix for C. But if we then find H, a generator-matrix for $C^\perp$, it is also true that

$C = (C^{\perp})^{\perp} = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}H^t = \mathbf{0}\}$. This is a very useful new way to specify any linear code.

Definition 4.6. Let $H \in M_{n-k,n}(\mathbb{F}_q)$ have linearly independent rows, and let $C = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}H^t = \mathbf{0}\}$. Then H is a **check-matrix** for C.

The name makes sense: we use H (or, in practice, its transpose) to 'check' whether $\mathbf{v}$ is in C or not. Notice that the rank of the map f_{H^t} is the rank of the matrix H^t, which is $n-k$. So the dimension of the code C defined in this way, which is the nullity of f_{H^t}, is $n-(n-k)=k$.

Proposition 4.7. *If the code C has generator-matrix G and check-matrix H, then $C^{\perp}$ has check-matrix G and generator-matrix H.*

Proof. Suppose $\dim(C) = k$. Then G has k rows, and H has $n-k$ rows. Also, by Proposition 4.3, $\dim(C^{\perp}) = n-k$.

The rows of G are linearly independent, and by Prop. 4.1 we know that $C^{\perp} = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}G^t = \mathbf{0}\}$, so G is a check-matrix for $C^{\perp}$.

The rows of H are orthogonal to every codeword in C, so they are in $C^{\perp}$. They are also linearly independent, and there are $n-k$ of them, so they form a basis for $C^{\perp}$. $\square$

The relationships among a code, its dual, and their respective generator- and check-matrices can be clarified by drawing pictures of the spaces and maps involved. They can also be very usefully summarised in the following table:

	C	$C^{\perp}$
Generator-matrix	G	H
Check-matrix	H	G

In the last section we discussed an algorithm which finds the basis of a dual space. So it finds H from G. But this means it also finds a check-matrix for C from its generator-matrix. Or, if we are given the check-matrix H for C, we can regard H as a generator-matrix for $C^{\perp}$, and then use the same algorithm to find a generator-matrix for $C = (C^{\perp})^{\perp}$. So we can use the algorithm to move either horizontally or vertically on the table; for this reason we can call it "the $G \leftrightarrow H$ algorithm".

If the matrix you have (either G or H) is in standard form $(I_k \mid A)$, the simpler algorithm of Proposition 4.5 can also be used to find the other one. Moreover, if we have H or G in form $(A \mid I_k)$, we can regard it as a check-matrix corresponding, by Proposition 4.5, to a generator matrix of form $(I_{n-k} \mid -A^t)$. (See Q47) For this reason, $(A \mid I_k)$ can be regarded as standard form for check-matrices. But since every check-matrix for a code C is also a generator-matrix for $C^{\perp}$ this could be confusing; it seems best to specify each time whether we mean standard form $(I_k \mid A)$ or standard form $(A \mid I_k)$.

Example 31. Let $C = \{\mathbf{v} \in \mathbb{F}_2^5 \mid \mathbf{v}H^t = \mathbf{0}\}$, with the single-row check-matrix $H = (1\,1\,1\,1\,1)$. Then the codewords of C are $\mathbf{c} = (c_1, \dots, c_5)$ such that $c_1 + \cdots + c_5 = 0$, so those with even weight. Thus H performs a simple "parity check"; to make a codeword we can choose 0 or 1 freely for any four of the entries, but the final entry must make the weight even. To find a basis for this code, since H is in standard form $(I_1 \mid A)$, we can use Proposition 4.5 and write down a generator-matrix $G_1 = (A^t \mid I_4)$. (For a binary code, $A = -A$.) But H is also in form $(A \mid I_1)$, so $G_2 = (I_4 \mid A^t)$ is another generator matrix. In fact, $G_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$ is the RREF form of $G_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$. △

What if $C = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{v}A^t = \mathbf{0}\}$, but $A \in M_{m,n}(F_q)$ does not have linearly independent rows? Or perhaps we do not know whether its row are independent or not? It is still true that $C = \ker(f_{A^t})$, and we might call A an "acting check-matrix" for C - it is doing the checking job, but it may not be fully qualified. Then, also, the rows of A are a spanning set for $C^\perp = \{\mathbf{v}A \mid \mathbf{v} \in \mathbb{F}_q^m\} = \mathrm{im}(f_A)$, but may not be a basis. We could similarly call A an "acting generator-matrix" for $C^\perp$.

Of course, using a check-matrix (or an acting check-matrix) to define a code is only a convenient new notation for a very familiar idea. You are familiar with defining a subspace using equations in the coordinates.

Example 32. If $H = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 1 \end{pmatrix} \in M_{2,3}(\mathbb{F}_5)$, and $C = \{\mathbf{v} \in \mathbb{F}_5^3 \mid \mathbf{v}H^t = \mathbf{0}\}$, then $C = \{(v_1, v_2, v_3) \in \mathbb{F}_5^3 \mid v_1 + 2v_2 + 3v_3 = 0 \text{ and } 4v_2 + v_3 = 0\}$. △

To solve such sets of equations, you would manipulate them in ways which correspond to elementary row operations on the check-matrix. This confirms that (as with generator-matrices) row-reducing a check-matrix for a code C gives another check-matrix for C.

4.3 Syndrome Decoding

In medicine, a "syndrome" is a collection of symptoms or characteristics which occur together. They are often apparently unrelated, but are assumed to have a single cause; over the last few decades, a genetic cause has been identified for many syndromes.

Similarly, the "syndrome" of a received word is useful evidence as to what error it may have suffered. We find the syndrome using the check-matrix. Thus, just as a generator-matrix makes it easy for a sender to encode a message, a check-matrix can help a receiver to decode a received word.

Definition 4.8. Suppose a code C has check-matrix H, so $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}H^t = 0\}$. For any received word $\mathbf{y}$, its **syndrome** is $S(\mathbf{y}) = \mathbf{y}H^t$.

Thus $S(\mathbf{y}) = \mathbf{0}$ if and only if $\mathbf{y}$ is a codeword. In this case we assume that it is in fact the one which was sent and no error-vector was added. In this way, the syndrome detects errors.

But a non-zero syndrome can also help to correct errors, by helping us to guess an error which is likely to have occurred. We know that $f_{H^t} : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^{n-k}$ is a linear map. So if $\mathbf{y} = \mathbf{c} + \mathbf{e}$, where $\mathbf{c} \in C$, then $S(\mathbf{y}) = S(\mathbf{c}) + S(\mathbf{e}) = \mathbf{0} + S(\mathbf{e}) = S(\mathbf{e})$. So the syndrome of the received word is the same as that of the error-vector $\mathbf{e}$. The syndrome is able to ignore the codeword and just "pick out" the error.

Unfortunately knowing $S(\mathbf{e})$ does not tell us $\mathbf{e}$, because the syndrome map f_{H^t} is not injective: two different errors can have the same syndrome. The following algorithm associates each possible syndrome with a single, likely, error-vector.

Algorithm: Syndrome decoding

Let C be a q-ary $[n, k]$ code, with check matrix $H \in M_{n-k,n}(\mathbb{F}_q)$,
so $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}H^t = \mathbf{0}\}$.

Construction of a syndrome look-up table

1. List the elements of $\mathbb{F}_q^n$ in non-decreasing order of weight.
2. Set up a table with two columns: **syndrome** $S(\mathbf{x})$ | **error-vector** $\mathbf{x}$.
3. Let $\mathbf{x}$ be the next element in the list and calculate $S(\mathbf{x})$.
4. If $S(\mathbf{x})$ is in the syndrome column already, do nothing.
   If it is not, write a new row: $S(\mathbf{x}) \mid \mathbf{x}$.
5. Repeat (3) and (4) until you have q^{n-k} rows.

Decoding (error 'correction') Having received a word $\mathbf{y}$,

1. Compute $S(\mathbf{y}) = \mathbf{y}H^t$.
2. Find $S(\mathbf{y})$ in the syndrome column.
3. Find the error-vector $\mathbf{x}$ that is in the same row.
4. Decode $\mathbf{y}$ to $\mathbf{y} - \mathbf{x}$.

Example 33. Let $C_1 = \{\mathbf{x} \in \mathbb{F}_2^4 \mid \mathbf{x}H^t = \mathbf{0}\}$, where $H = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ is a check-matrix for C_1. We calculate syndromes, starting with words of weight 0, then 1, then 2: $S(0,0,0,0) = (0,0)$, $S(1,0,0,0) = (1,0)$, $S(0,1,0,0) = (1,0)$, $S(0,0,1,0) = (0,1)$, $S(0,0,0,1) = (0,1)$, $S(1,1,0,0) = (0,0)$, $S(1,0,1,0) = (1,1)$, $S(1,0,0,1) = (1,1) \ldots$

Omitting the repeated syndromes, we make the following look-up table:

Syndrome $S(\mathbf{x})$	Error-vector $\mathbf{x}$
(0,0)	(0,0,0,0)
(1,0)	(1,0,0,0)
(0,1)	(0,0,1,0)
(1,1)	(1,0,1,0)

We can stop here, as we have 2^{4-2} rows; equivalently, we have every possible syndrome.

Now suppose we receive $\mathbf{y}_1 = (1,1,0,1)$. Then $S(\mathbf{y}_1) = (0,1)$, so the table says that the error-vector was (0,0,1,0), and we decode to $(1,1,0,1) - (0,0,1,0) = (1,1,1,1) = \mathbf{c}_1$. Similarly, $\mathbf{y}_2 = (0,1,0,0)$ decodes to $\mathbf{c}_2 = (1,1,0,0)$. △

By the theory, both these $\mathbf{c}_i$ should be in C_1; we can check this by finding $S(\mathbf{c}_i)$. We could also use the "$G \leftrightarrow H$ algorithm" to find a generator matrix G for C. Surprisingly, we find that $G = H$, so $C_1 = C_1^{\perp}$; C_1 is 'self-dual' [3]. So this is actually the code for which, in Section 2.3, we made this decoding array:

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

We see that the $\mathbf{c}_i$ are in the top row, which lists the code. Also, the left-hand column of the array matches the error-vector column of the look-up table; these are the (guessed) errors we will subtract. And certainly this array gives the same decoding as the look-up table for (1,1,0,1) and (0,0,1,0). We can also see a examples of the following:

Proposition 4.9. *Two words are in the same row of a decoding array if and only if they have the same syndrome.*

Proof. In general, finding the two words in the array (see below) expresses them as $\mathbf{y}_1 = \mathbf{c}_1 + \mathbf{x}_1$ and $\mathbf{y}_2 = \mathbf{c}_2 + \mathbf{x}_2$, with $\mathbf{c}_1 \in C$, and we know already that $S(\mathbf{y}_1) = S(\mathbf{x}_1)$ and $S(\mathbf{y}_2) = S(\mathbf{x}_2)$.

$\mathbf{0}$	$\mathbf{c}_2$	$\mathbf{c}_1$
$\mathbf{x}_1$		$\mathbf{y}_1$
$\mathbf{x}_2$	$\mathbf{y}_2$	

If $\mathbf{y}_1$ and $\mathbf{y}_2$ are in the same row, then $\mathbf{x}_1 = \mathbf{x}_2 = \mathbf{x}$, so $S(\mathbf{y}_1) = S(\mathbf{y}_2) = S(\mathbf{x})$.

Conversely, if $S(\mathbf{y}_1) = S(\mathbf{y}_2)$ then $S(\mathbf{y}_1 - \mathbf{y}_2) = S(\mathbf{y}_1) - S(\mathbf{y}_2) = \mathbf{0}$, so $\mathbf{y}_1 - \mathbf{y}_2 = \mathbf{c} \in C$. Then $\mathbf{y}_1 = \mathbf{y}_2 + \mathbf{c} = \mathbf{x}_2 + \mathbf{c}_2 + \mathbf{c}$. Since $\mathbf{c}_2 + \mathbf{c} \in C$, it must be in the top row, so $\mathbf{y}_1$ is in $\mathbf{x}_2$'s row. □

In effect, syndrome decoding is just a more efficient way to do array decoding; without either making or searching through the array, finding $S(\mathbf{y})$ tells us which row of the array $\mathbf{y}$ would be on. So it follows from Proposition 2.10 that syndrome decoding, also, is nearest-neighbour decoding. (We can also prove this directly: Q53)

As with the array, there is some choice in the construction of the syndrome look-up table; it comes in the initial ordering of the words of $\mathbb{F}_q^n$. If this is different, we may get a different column of error-vectors to subtract, which will certainly result in different decoding of some words.

[3]This could not happen over $\mathbb{R}$.

Example 34. Let $C_2 = \{\mathbf{x} \in \mathbb{F}_3^3 \mid \mathbf{x}H^t = 0\}$, where $H = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{pmatrix}$ is a check-matrix for C_2. Then this is one possible syndrome look-up table:

Syndrome $S(\mathbf{x})$	Error-vector $\mathbf{x}$
(0,0)	(0,0,0)
(1,0)	(1,0,0)
(2,0)	(2,0,0)
(0,1)	(0,1,0)
(0,2)	(0,2,0)
(2,2)	(0,0,1)
(1,1)	(0,0,2)
(1,2)	(1,2,0)
(2,1)	(1,0,2)

Here we have used every possible $\mathbf{x}$ of weight 1, so the order in which we considered them did not matter. But the last two lines could instead be:

Syndrome $S(\mathbf{x})$	Error-vector $\mathbf{x}$
(2,1)	(0,2,1)
(1,2)	(2,0,1)

We can conclude that any error-vector of weight ≤ 1, but only some errors of weight 2, will be correctly identified and subtracted. Which errors of weight 2 are correctly subtracted, and which are not, depends on which table we use. For this reason we might decide to practice incomplete decoding: cut the table short, and if we receive a word with syndrome (1,2) or (2,1) ask for retransmission. △

Looking back to C_1, we see that the table lists only some $\mathbf{x}$'s of weight 1, so we cannot be sure of reliably correcting even error-vectors of weight 1. But we knew this: $d(C_1) = 2$, so by Proposition 1.7 we will detect a single symbol-error, but nearest-neighbour decoding may not correct it.

On the other hand, using Proposition 4.5 (or by guessing and checking) we find that $C_2 = \{(0,0,0), (1,1,1), (2,2,2)\}$, so $d(C_2) = 3$ and we can indeed reliably correct one symbol-error, but not two. Equivalently we know that for this code, spheres $S(\mathbf{c}, 1)$ around the codewords are disjoint, but the $S(\mathbf{c}, 2)$ intersect. (Q24 and 25 consider alternative arrays for this code.)

The examples we've discussed so far have all been for binary or ternary codes. For codes over a larger alphabet, the number of rows in a syndrome table can get quite large. However, since the syndrome is a linear map on $\mathbb{F}_q^n$, we have $S(\lambda\mathbf{y}) = \lambda S(\mathbf{y})$ for any non-zero $\lambda \in \mathbb{F}_q^n$ – we can see this explicitly in Example 34 above.

For codes with $q > 2$, we can therefore define a *reduced* syndrome table, where we only add new syndromes to our table if they aren't of the form $\lambda S(\mathbf{x})$, for any non-zero $\lambda \in \mathbb{F}_q$, and any $S(\mathbf{x})$ already in our table. To decode a received word $\mathbf{y}$, we then calculate $S(\mathbf{y})$ as normal, but now we need to find the row such that $\lambda S(\mathbf{y})$ is in the first column, for some non-zero λ which we need to calculate. We then decode $\mathbf{y}$ to $\mathbf{y} - \lambda\mathbf{x}$, where $\mathbf{x}$ is the error vector in the corresponding row of our table. See Q52 for an example of this idea.

4.4 Minimum distance from a check-matrix

In the last section, $d(C)$ turned out to be relevant to the reliability of our syndrome look-up table. But to find it, we had first to find the words of the code. We will now establish a way to get $d(C)$ directly from a check-matrix, which links up many of the ideas so far.

In fact, it only needs to be an "acting check-matrix". We start with the following:

Lemma 4.10. *For some $A \in M_{m,n}(\mathbb{F}_q)$, let $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}A^t = \mathbf{0}\}$. Then:*
There are d columns of A which are linearly dependent
$\iff$ there is some codeword $\mathbf{c} \in C$ with $0 < w(\mathbf{c}) \leq d$.

Proof. Let the columns of A be $\mathbf{a}_1, \dots, \mathbf{a}_n$.

$\Longrightarrow$ Suppose we have d linearly dependent columns, $\mathbf{a}_{i_1}, \dots, \mathbf{a}_{i_d}$. This means there exist $\lambda_1, \lambda_2, \dots, \lambda_d$ in $\mathbb{F}_q$, not all 0, such that $\lambda_1\mathbf{a}_{i_1} + \cdots + \lambda_d\mathbf{a}_{i_d} = \mathbf{0}$. Now let $\mathbf{c}$ be a word with λ_j in position i_j, 0 elsewhere. Then $0 < w(\mathbf{c}) \leq d$. But also, when multiplying $\mathbf{c}A^t$, each λ_j picks out row i_j of A^t, so

$$\mathbf{c}A^t = (0, \dots 0, \lambda_1, 0, \dots, 0, \lambda_d, 0, \dots, 0) \begin{pmatrix} & \vdots & \\ - & \mathbf{a}_{i_1} & - \\ & \vdots & \\ - & \mathbf{a}_{i_d} & - \\ & \vdots & \end{pmatrix} = \lambda_1\mathbf{a}_{i_1} + \cdots + \lambda_d\mathbf{a}_{i_d} = \mathbf{0}.$$

So $\mathbf{c} \in C$.

$\Longleftarrow$ If $\mathbf{c} = (c_1, c_2, \dots, c_n) \in C$, and $0 < w(\mathbf{c}) \leq d$, we know that $c_1\mathbf{a}_1 + \cdots + c_n\mathbf{a}_n = \mathbf{c}A^t = 0$, and that between 1 and d of the $\mathbf{c}_i$ are non-zero. If we choose $c_{i_1}, \dots, c_{i_d}$ to include all the non-zero c_i, then we still have $c_{i_1}\mathbf{a}_{i_1} + \cdots + c_{i_d}\mathbf{a}_{i_d} = 0$, with not all $c_i = 0$. Thus $\mathbf{a}_{i_1}, \dots, \mathbf{a}_{i_d}$ are linearly dependent. □

Example 35. Let $C = \{\mathbf{x} \in \mathbb{F}_7^5 \mid \mathbf{x}A^t = 0\}$, where $A = \begin{pmatrix} 3 & 1 & 1 & 4 & 1 \\ 2 & 2 & 5 & 1 & 4 \\ 6 & 3 & 5 & 0 & 2 \end{pmatrix} \in M_{3,5}(\mathbb{F}_7)$.

Because $(0,1,2,0,4)\begin{pmatrix} 3 & 2 & 6 \\ 1 & 2 & 3 \\ 1 & 5 & 5 \\ 4 & 1 & 0 \\ 1 & 4 & 2 \end{pmatrix} = (0,0,0)$, we know two things:

- $(0,1,2,0,4) \in C$, so C contains a codeword of weight 3.
- $1(1,2,3) + 2(1,5,5) + 4(1,2,4) = (0,0,0)$, so A has 3 columns which are linearly dependent.

△

Theorem 4.11. *For some $A \in M_{m,n}(\mathbb{F}_q)$, let $C = \{\mathbf{x} \in \mathbb{F}_q^n \mid \mathbf{x}A^t = \mathbf{0}\}$. Then there is some set of $d(C)$ columns of A which are linearly dependent, but any $d(C) - 1$ columns of A are linearly independent.*

Proof. For a linear code, by Proposition 2.7 $d(C) = min\{w(\mathbf{c}) \mid \mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}\}$. So we know:

- There is some $\mathbf{c} \in C$ with $w(\mathbf{c}) = d(C)$. So by Lemma 4.10 there are $d(C)$ columns which are linearly dependent.
- There is no $\mathbf{c} \in C$ with $w(\mathbf{c}) \leq d(C) - 1$. So by Lemma 4.10 there is no set of $d(C) - 1$ columns which are linearly dependent.

□

This theorem is mostly used in reverse: We find the number d such that A has a set of d dependent columns, but no smaller such sets. Then we conclude that d is the minimum distance of the code. One can remember the theorem as something like "$d(C)$ is the size of a smallest set of linearly dependent columns in the check-matrix".

Example 36. For the code C in the example above, we have found that columns 2, 3 and 5 are linearly dependent. But this only tells us that $d(C) \leq 3$. To be sure that $d(C) = 3$, we need also to check that there are no linearly dependent pairs of columns, that is, no column is a multiple of another. For many of the $\binom{5}{2}$ pairs this is easy: its zero means that column 4 is not a multiple of any other, and (since they are not identical) the top entry 1 in columns 2, 3, and 5 means they cannot be multiples of each other. It remains to check that column 1 is not a multiple of column 2, 3 or 5. It is not, so $d(C) = 3$. △