MAS309 Coding theory

Matthew Fayers

January–March 2008

This is a set of notes which is supposed to augment your own notes for the Coding Theory course. They were written by Matthew Fayers, and very lightly edited my me, Mark Jerrum, for 2008. I am very grateful to Matthew Fayers for permission to use this excellent material. If you find any mistakes, please e-mail me: `mj@maths.qmul.ac.uk`. Thanks to the following people who have already sent corrections: Nilmini Herath, Julian Wiseman, Dilara Azizova.

Contents

1 Introduction and definitions

1.1 Alphabets and codes

In this course, we shall work with an *alphabet*, which is simply a finite set $\mathbb{A}$ of symbols. If $\mathbb{A}$ has size q, then we call $\mathbb{A}$ a *q-ary* alphabet (although we say *binary* and *ternary* rather than 2-ary and 3-ary). For most purposes, it is sufficient to take $\mathbb{A}$ to be the set $\{\mathtt{0}, \mathtt{1}, \dots, \mathtt{q-1}\}$. Later we shall specialise to the case where q is a prime power, and take $\mathbb{A}$ to be $\mathbb{F}_q$, the field of order q.

A *word* of length n is simply a string consisting of n (not necessarily distinct) elements of $\mathbb{A}$, i.e. an element of $\mathbb{A}^n$, and a *block code of length n* is simply a set of words of length n, i.e. a subset of $\mathbb{A}^n$. If $\mathbb{A}$ is a q-ary alphabet, we say that any code over $\mathbb{A}$ is a q-ary code. There are codes in which the words have different lengths, but in this course we shall be concerned entirely with block codes, and so we refer to these simply as codes. We refer to the words in a code as the *codewords*.

1.2 Error detection and correction

Informally, a code is *t-error-detecting* if, whenever we take a codeword and change at most t of the symbols in it, we don't reach a different codeword. So if we send the new word to someone without telling him which symbols we changed, he will be able to tell us whether we changed any symbols.

A code is *t-error-correcting* if whenever we take a codeword and change at most t of the symbols in it, we don't reach a different codeword, and we don't even reach a word which can be obtained from a different starting codeword by changing at most t of the symbols. So if we send the new word to someone without telling him which symbols we changed, he will be able to tell us which codeword we started with.

Formally, we define a metric (or distance function) on $\mathbb{A}^n$ as follows: given two words x and y, we define $d(x, y)$ to be the number of positions in which x and y differ, i.e. if $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$, then $d(x, y)$ is the number of values i for which $x_i \neq y_i$. This distance function is called the *Hamming distance*.

Lemma 1.1. *d is a metric on $\mathbb{A}^n$, i.e.:*

1. *$d(x, x) = 0$ for all $x \in \mathbb{A}^n$;*
2. *$d(x, y) > 0$ for all $x \neq y \in \mathbb{A}^n$;*
3. *$d(x, y) = d(y, x)$ for all $x, y \in \mathbb{A}^n$;*
4. *(the triangle inequality) $d(x, z) \leqslant d(x, y) + d(y, z)$ for all $x, y, z \in \mathbb{A}^n$.*

Proof. (1), (2) and (3) are very easy, so let's do (4). Now $d(x, z)$ is the number of values i for which $x_i \neq z_i$. Note that if $x_i \neq z_i$, then either $x_i \neq y_i$ or $y_i \neq z_i$. Hence

$$\{i \mid x_i \neq z_i\} \subseteq \{i \mid x_i \neq y_i\} \cup \{i \mid y_i \neq z_i\}.$$

So

$$\begin{aligned}\{|i \mid x_i \neq z_i\}| &\leqslant |\{i \mid x_i \neq y_i\} \cup \{i \mid y_i \neq z_i\}| \\ &\leqslant |\{i \mid x_i \neq y_i\}| + |\{i \mid y_i \neq z_i\}|,\end{aligned}$$

i.e.

$$d(x,z) \leqslant d(x,y) + d(y,z).$$

□

Now we can talk about error detection and correction. We say that a code C is *t-error detecting* if $d(x, y) > t$ for any two distinct words in C. We say that C is *t-error-correcting* if there do not exist words $x, y \in C$ and $z \in \mathbb{A}^n$ such that $d(x,z) \leqslant t$ and $d(y,z) \leqslant t$.

Example. The simplest kinds of error-detecting codes are *repetition codes*. The repetition code of length n over $\mathbb{A}$ simply consists of all words $\mathtt{aa}\dots\mathtt{a}$, for $\mathtt{a} \in \mathbb{A}$. For this code, any two distinct codewords differ in *every* position, and so $d(x, y) = n$ for all $x \neq y$ in C. So the code is t-error-detecting for every $t \leqslant n-1$, and is t-error-correcting for every $t \leqslant \frac{n-1}{2}$.

Given a code C, we define its *minimum distance* $d(C)$ to be the smallest distance between distinct codewords:

$$d(C) = \min\{d(x,y) \mid x \neq y \in C\}.$$

Lemma 1.2. *A code C is t-error-detecting if and only if $d(C) \geqslant t+1$, and is t-error-correcting if and only if $d(C) \geqslant 2t+1$,*

Proof. The first part is immediate from the definition of "t-error-detecting". For the second part, assume that C is *not* t-error-correcting. Then there exist distinct codewords $x, y \in C$ and and a word $z \in \mathbb{A}^n$ such that $d(x,z) \leq t$ and $d((y,z) \leq t$. By the triangle inequality, $d(x,y) \leq d(x,z) + d((y,z) \leq 2t$, and hence $d(C) \leq 2t$. Conversely, if $d(C) \leq 2t$ then choose $x, y \in C$ such that $d(x,y) \leq 2t$. There exists $z \in \mathbb{A}^n$ such that $d(x,z) \leq t$ and $d((y,z) \leq t$. (Check this! It is a property of the Hamming metric, but not of metrics in general.) Thus, C is not t-error-correcting. □

Corollary 1.3. *A code C is t-error-correcting if and only if it is $(2t)$-error-detecting.*

Proof. By the previous lemma, the properties "t-error-correcting" and "$2t$-error-detecting" for the code C are both equivalent to $d(C) \geq 2t+1$. □

From now on, we shall think about the minimum distance of a code rather than how many errors it can detect or correct.

We say that a code of length n with M codewords and minimum distance at least d is an (n, M, d)-code. For example, the repetition code described above is an (n, q, n)-code. Another example is the following 'parity-check' code, which is a binary $(4, 8, 2)$-code:

$$\{\mathtt{0000}, \mathtt{0011}, \mathtt{0101}, \mathtt{0110}, \mathtt{1001}, \mathtt{1010}, \mathtt{1100}, \mathtt{1111}\}.$$

The point of using error-detecting and error-correcting codes is that we might like to transmit a message over a 'noisy' channel, where each symbol we transmit gets mis-transmitted with a certain probability; an example (for which several of the codes we shall see have been used) is a satellite transmitting images from the outer reaches of the solar system. Using an error-detecting code, we reduce the probability that the receiver misinterprets distorted information – provided not too many errors have been made in transmission, the receiver will know that errors have been made, and can request re-transmission; in a situation where re-transmission is impractical, an error-correcting code can be used. Of course, the disadvantage of this extra 'certainty' of faithful transmission is that we are adding redundant information to the code, and so our message takes longer to transmit. In addition, for intricate codes, decoding may be difficult and time-consuming.

The main tasks of coding theory, therefore, are to find codes which enable error-detection and -correction while adding as little redundant information as possible, and to find efficient decoding procedures for these codes. Clearly, as d gets large, codes with minimum distance d have fewer and fewer codewords. So we try to find codes of a given length and a given minimum distance which have as many codewords as possible. We shall see various bounds on the possible sizes of codes with given length and minimum distance, and also construct several examples of 'good' codes.

1.3 Equivalent codes

To simplify our study of codes, we introduce a notion of equivalence of codes. If C and $\mathcal{D}$ are codes of the same length over the same alphabet, we say that C is equivalent to $\mathcal{D}$ if we can get from C to $\mathcal{D}$ by a combination of the following operations.

Operation 1 – permutation of the positions in the codewords Choose a permutation σ of $\{1,\dots,n\}$, and for a codeword $v = v_1 \dots v_n$ in C define

$$v_\sigma = v_{\sigma(1)} \dots v_{\sigma(n)}.$$

Now define

$$C_\sigma = \{v_\sigma \mid v \in C\}.$$

Operation 2 – applying a permutation of $\mathbb{A}$ in a fixed position in the codewords Choose a permutation f of $\mathbb{A}$ and an integer $i \in \{1,\dots,n\}$, and for $v = v_1 \dots v_n$ define

$$v_{f,i} = v_1,\dots,v_{i-1}f(v_i)v_{i+1}\dots v_n.$$

Now define

$$C_{f,i} = \{v_{f,i} \mid v \in C\}.$$

For example, consider the following ternary codes of length 2:

$$C = \{\mathtt{10}, \mathtt{21}, \mathtt{02}\}, \quad \mathcal{D} = \{\mathtt{01}, \mathtt{12}, \mathtt{20}\}, \quad \mathcal{E} = \{\mathtt{00}, \mathtt{11}, \mathtt{22}\}.$$

We can from C to $\mathcal{D}$ by Operation 1 – we replace each codeword $\mathtt{ab}$ with $\mathtt{ba}$. We can get from $\mathcal{D}$ to $\mathcal{E}$ by Operation 2 – we permute the symbols appearing in the second position via $\mathtt{0} \to \mathtt{2} \to \mathtt{1} \to \mathtt{0}$. So C, $\mathcal{D}$ and $\mathcal{E}$ are equivalent codes.

The point of equivalence is that equivalent codes have the same size and the same minimum distance; we can often simplify both decoding procedures and some of our proofs by replacing codes with equivalent codes.

Lemma 1.4. *Suppose C is a code and σ a permutation of $\{1,\dots,n\}$, and define C_σ as above. Then* $|C| = |C_\sigma|$.

Proof. The map

$$v \mapsto v_\sigma$$

defines a function from C to C_σ, and we claim that this is a bijection. Surjectivity is immediate: C_σ is defined to be the image of the function. For injectivity, suppose that $v = v_1 \dots v_n$ and $w = w_1 \dots w_n$ are codewords in C with $v \neq w$. This means that $v_j \neq w_j$ for some j. Since σ is a permutation, we have $j = \sigma(i)$ for some i, and so v_σ and w_σ differ in position i, so are distinct. □

Lemma 1.5. *Suppose C is a code containing words v and w, and suppose σ is a permutation of $\{1, \dots, n\}$. Define the words v_σ and w_σ as above. Then*

$$d(v_\sigma, w_\sigma) = d(v, w).$$

Proof. Write $v = v_1 \dots v_n$, $w = w_1 \dots_n$, $v_\sigma = x_1 \dots x_n$, $w_\sigma = y_1 \dots y_n$. Then by definition we have $x_i = v_{\sigma(i)}$ and $y_i = w_{\sigma(i)}$ for each i. Now $d(v, w)$ is the number of positions in which v and w differ, i.e.

$$d(v, w) = |\{i \mid v_i \neq w_i\}|;$$

similarly,

$$d(v_\sigma, w_\sigma) = |\{i \mid x_i \neq y_i\}|.$$

Since $x_i = v_{\sigma(i)}$ and $y_i = w_{\sigma(i)}$, σ defines a bijection from

$$\{i \mid x_i \neq y_i\}$$

to

$$\{i \mid v_i \neq w_i\}.$$

□

Corollary 1.6. *Define C_σ as above. Then $d(C_\sigma) = d(C)$.*

Now we prove the same properties for Operation 2.

Lemma 1.7. *Suppose C is a code, f a permutation of $\mathbb{A}$ and $i \in \{1, \dots, n\}$, and define $C_{f,i}$ as above. Then $|C_{f,i}| = |C|$.*

Proof. The map

$$v \mapsto v_{f,i}$$

defines a function from C to $C_{f,i}$, and we claim that this is a bijection. Surjectivity follows by definition of $C_{f,i}$. For injectivity, suppose that v, w are codewords in C with $v \neq w$. There is some $j \in \{1, \dots, n\}$ with $v_j \neq w_j$. If $j = i$, then $f(v_i) \neq f(w_j)$ (since f is a permutation), and so $v_{f,i}$ and $w_{f,i}$ differ in position i. If $j \neq i$, then $v_{f,i}$ and $w_{f,i}$ differ in position j. □

Lemma 1.8. *Suppose C is a code containing codewords v and w, and define $v_{f,i}$ and $w_{f,i}$ as above. Then*

$$d(v_{f,i}, w_{f,i}) = d(v, w).$$

Proof. Write $v = v_1 \dots v_n$ and $w = w_1 \dots w_n$. If $v_i = w_i$, then $f(v_i) = f(w_i)$, so

$$d(v, w) = |\{j \neq i \mid v_i \neq w_i\}| = d(v_{f,i}, w_{f,i}).$$

If $v_i \neq w_i$, then (since f is a permutation) $f(v_i) \neq f(w_i)$, and so

$$d(v, w) = |\{j \neq i \mid v_i \neq w_i\}| + 1 = d(v_{f,i}, w_{f,i}).$$

□

Corollary 1.9. *Define $C_{f,i}$ as above. Then $d(C_{f,i}) = d(C)$.*

One of the reasons for using equivalent codes is that they enable us to assume that certain words lie in our codes.

Lemma 1.10. *Suppose C is a non-empty code over $\mathbb{A}$, and* $\mathtt{a} \in \mathbb{A}$. *Then C is equivalent to a code containing the codeword* $\mathtt{aa}\dots\mathtt{a}$.

Proof. We prove a stronger statement by induction. For $i \in \{0, \dots, n\}$ let $P(i)$ denote the statement 'A non-empty code C is equivalent to a code containing a word $v = v_1 \dots v_n$ with $v_1 = \dots = v_i = \mathtt{a}$'.

$P(0)$ is true: any code is equivalent to itself, and we may take any v. Now suppose $i > 0$ and that $P(i-1)$ is true. So C is equivalent to a code $\mathcal{D}$ containing a word $v = v_1 \dots v_n$ with $v_1 = \dots = v_{i-1} = \mathtt{a}$. Choose any permutation f of $\mathbb{A}$ for which $f(v_i) = \mathtt{a}$. Then $\mathcal{D}$ is equivalent to the code $\mathcal{D}_{f,i}$, which contains the word $v_{f,i}$, whose first i entries equal $\mathtt{a}$, and C is equivalent to $\mathcal{D}_{f,i}$ as well.

By induction, $P(n)$ is true, which is what we want. □

2 Good codes

2.1 The main coding theory problem

The most basic question we might ask about codes is: given n, M and d, does an (n, M, d)-code exist? Clearly, better codes are those which make M and d large relative to n, so we define $A_q(n, d)$ to be the maximum M such that a q-ary (n, M, d)-code exists. The numbers $A_q(n, d)$ are unknown in general, and calculating them is often referred to as the 'main coding theory problem'. Here are two very special cases.

Theorem 2.1.

1. $A_q(n, 1) = q^n$.
2. $A_q(n, n) = q$.

Proof.

1. We can take $C = \mathbb{A}^n$, the set of all words of length n. Any two distinct words must differ in at least one position, so the code has minimum distance at least 1. Obviously a q-ary code of length n can't be bigger than this.

2. Suppose we have a code of length n with at least $q + 1$ codewords. Then by the pigeonhole principle there must be two words with the same first symbol. These two words can therefore differ in at most $n-1$ positions, and so the code has minimum distance less than n. So $A_q(n, n) \leqslant q$. On the other hand, the repetition code described above is an (n, q, n)-code.

□

Here is a less trivial example.

Theorem 2.2.

$$A_2(5,3) = 4.$$

Proof. It is easily checked that the following binary code is $(5,4,3)$:

$$\{\mathtt{00000}, \mathtt{01101}, \mathtt{10110}, \mathtt{11011}\}.$$

So $A_2(5,3) \geqslant 4$. Now suppose C is a binary code of length 5 with minimum distance at least 3 and at least five codewords. By replacing C with an equivalent code and appealing to Lemma 1.10, we may assume that C contains the codeword $\mathtt{00000}$. Since C has minimum distance at least 3, every remaining codeword must contain at least three 1s. If there are two codewords x, y each with at least four 1s, then $d(x,y) \leqslant 2$, which gives a contradiction, so there must be at least three codewords with exactly three 1s. Trial and error shows that two of these must be distance at most 2 apart; contradiction. □

Now we come to our first non-trivial result. It is a 'reduction theorem', which in effect says that for binary codes we need only consider codes with odd values of d.

Theorem 2.3. *Suppose d is even. Then a binary (n, M, d)-code exists if and only if a binary $(n-1, M, d-1)$-code exists.*

Hence if d is even, then $A_2(n,d) = A_2(n-1, d-1)$.

Proof. The 'only if' part follows from the Singleton bound, which we state and prove in Section 2.3. So we concentrate on the 'if' part.

Suppose we have a binary $(n-1, M, d-1)$-code. Given a codeword x, we form a word $\hat{x}$ of length n by adding an extra symbol, which we choose to be $\mathtt{0}$ or $\mathtt{1}$ in such a way that $\hat{x}$ contains an even number of 1s.

Claim. If x, y are codewords in C, then $d(\hat{x}, \hat{y})$ is even.

Proof. The number of positions in which $\hat{x}$ and $\hat{y}$ differ is

$$\begin{aligned}&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1} \text{ and } \hat{y} \text{ has a } \mathtt{0})\\ +&(\text{number of places where } \hat{x} \text{ has a } \mathtt{0} \text{ and } \hat{y} \text{ has a } \mathtt{1})\end{aligned}$$

which equals

$$\begin{aligned}&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1} \text{ and } \hat{y} \text{ has a } \mathtt{0})\\ +&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1} \text{ and } \hat{y} \text{ has a } \mathtt{1})\\ +&(\text{number of places where } \hat{x} \text{ has a } \mathtt{0} \text{ and } \hat{y} \text{ has a } \mathtt{1})\\ +&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1} \text{ and } \hat{y} \text{ has a } \mathtt{1})\\ -2&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1} \text{ and } \hat{y} \text{ has a } \mathtt{1})\end{aligned}$$

which equals

$$\begin{aligned}&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1})\\ +&(\text{number of places where } \hat{y} \text{ has a } \mathtt{1})\\ -2&(\text{number of places where } \hat{x} \text{ has a } \mathtt{1} \text{ and } \hat{y} \text{ has a } \mathtt{1})\end{aligned}$$

which is the sum of three even numbers, so is even.

Now for any $x, y \in C$, we have $d(x, y) \geqslant d - 1$, and clearly this gives $d(\hat{x}, \hat{y}) \geqslant d - 1$. But $d - 1$ is odd, and $d(\hat{x}, \hat{y})$ is even, so in fact we have $d(\hat{x}, \hat{y}) \geqslant d$. So the code

$$\hat{C} = \{\hat{x} \mid x \in C\}$$

is an (n, M, d)-code.

For the final part of the theorem, we have

$$\begin{aligned} A_2(n, d) &= \max\{M \mid \text{a binary } (n, M, d)\text{-code exists}\} \\ &= \max\{M \mid \text{a binary } (n-1, M, d-1)\text{-code exists}\} \\ &= A_2(n-1, d-1). \end{aligned}$$

□

Now we'll look at some upper bounds for sizes of (n, M, d)-codes.

2.2 Spheres and the Hamming bound

Since the Hamming distance d makes $\mathbb{A}^n$ into a metric space, we can define the sphere around any word. If $x \in \mathbb{A}^n$ is a word, then the sphere of radius r and centre x is

$$S(x, r) = \{y \in \mathbb{A}^n \mid d(x, y) \leqslant r\}.$$

(n.b. in metric-space language this is a *ball*, but the word 'sphere' is always used by coding-theorists.) The importance of spheres lies in the following lemma.

Lemma 2.4. *A code C is t-error-correcting if and only if for any distinct words $x, y \in C$, the spheres $S(x, t)$ and $S(y, t)$ are disjoint.*

This lemma gives us a useful bound on the size of a t-error-correcting code. We begin by counting the words in a sphere; recall the *binomial coefficient* $\binom{n}{r} = \frac{n!}{(n-r)!r!}$.

Lemma 2.5. *If $\mathbb{A}$ is a q-ary alphabet, x is a word over $\mathbb{A}$ of length n and $r \leqslant n$, then the sphere $S(x, r)$ contains exactly*

$$\binom{n}{0} + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^r\binom{n}{r}$$

words.

Proof. We claim that for any i, the number of words y such that $d(x, y)$ equals i is $(q-1)^i\binom{n}{i}$; the lemma then follows by summing for $i = 0, 1, \dots, r$.

$d(x, y) = i$ means that x and y differ in exactly i positions. Given x, in how many ways can we choose such a y? We begin by choosing the i positions in which x and y differ; this can be done in $\binom{n}{i}$ ways. Then we choose what symbols will appear in these i positions in y. For each position, we can choose any symbol other than the symbol which appears in that position in x – this gives us $q - 1$ choices. So we have $(q-1)^i$ choices for these i symbols altogether. □

Theorem 2.6 (Hamming bound)**.** *If C is a t-error-correcting code of length n over a q-ary alphabet $\mathbb{A}$, then*

$$|C| \leqslant \frac{q^n}{\binom{n}{0} + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^t\binom{n}{t}}.$$

Proof. Each codeword has a sphere of radius t around it, and by Lemma 2.4 these spheres are disjoint. So the total number of words in all these spheres together is

$$M \times \left(\binom{n}{0} + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^t\binom{n}{t} \right),$$

and this can't be bigger than the total number of possible words, which is q^n. □

The Hamming bound is also known as the *sphere-packing bound.*

Corollary 2.7. *For $n, q, t > 0$,*

$$A_q(n, 2t+1) \leqslant \frac{q^n}{\binom{n}{0} + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^t\binom{n}{t}}.$$

Proof. Suppose C is a q-ary $(n, M, 2t+1)$-code. Then C is t-error-correcting (from Section 1), so

$$|M| \leqslant \frac{q^n}{\binom{n}{0} + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^t\binom{n}{t}},$$

by the Hamming bound. □

Definition. A q-ary (n, M, d)-code is called *perfect* if $d = 2r + 1$ and

$$M = \frac{q^n}{\binom{n}{0} + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^r\binom{n}{r}}$$

for some r, that is, if equality holds in the Hamming bound. For example, if n is odd and $q = 2$, then the repetition code described in §1.2 is perfect (check this!). Later, we shall see some more interesting examples of perfect codes.

2.3 The Singleton bound

Theorem 2.8 (Singleton bound)**.**

1. *Suppose $n, d > 1$. If a q-ary (n, M, d)-code exists, then a q-ary $(n-1, M, d-1)$-code exists. Hence $A_q(n, d) \leqslant A_q(n-1, d-1)$.*
2. *Suppose $n, d \geqslant 1$. Then $A_q(n, d) \leqslant q^{n-d+1}$.*

Proof.

1. Let C be a q-ary (n, M, d)-code, and for $x \in C$, let $\overline{x}$ be the word obtained by deleting the last symbol. Let $\overline{C} = \{\overline{x} \mid x \in C\}$.

   Claim. If $x, y \in C$ with $x \neq y$, then $d(\overline{x}, \overline{y}) = d - 1$ or d.

   Proof. We have $d(x, y) \geqslant d$, so x and y differ in at least d positions; if the last position is one of these, then $\overline{x}$ and $\overline{y}$ differ in only $d - 1$ positions. If the last position is not one of the positions where x and y differ, then $\overline{x}$ and $\overline{y}$ differ in d positions.

The first consequence of the claim is that, since $d > 1$, $\overline{x}$ and $\overline{y}$ are distinct when x and y are. So $|\overline{C}| = M$. The second consequence is that $d(\overline{C}) \geqslant d - 1$. So $\overline{C}$ is an $(n-1, M, d-1)$-code.

To show that $A_q(n,d) \leqslant A_q(n-1, d-1)$, take an (n, M, d)-code C with $M = A_q(n,d)$. Then we get an $(n-1, M, d-1)$-code $\overline{C}$, which means that $A_q(n-1, d-1) \geqslant M = A_q(n,d)$.

2. We prove this part by induction on d, with the case $d = 1$ following from Theorem 2.1. Now suppose $d > 1$ and that the inequality holds for $d - 1$ (and $n - 1$). This means

$$A_q(n-1, d-1) \leqslant q^{(n-1)-(d-1)+1} = q^{n-d+1}.$$

Now apply the first part of the present theorem.

□

Note that the Singleton bound finishes off the proof of Theorem 2.3.

2.4 Another bound

This bound seems not to have a name.

Theorem 2.9. *Suppose* $n > 1$. *Then*

$$A_q(n,d) \leqslant qA_q(n-1,d).$$

Proof. It suffices to prove that if a q-ary (n, M, d)-code exists, then so does a q-ary $(n-1, P, d)$-code, for some $P \geqslant M/q$. Indeed, we can take $M = A_q(n,d)$, which will give $qP \geqslant A_q(n,d)$, so that $qA_q(n-1,d) \geqslant A_q(n,d)$. So let C be a q-ary (n, M, d)-code. Look at the last symbol of each codeword, and for each $\mathsf{a} \in \mathbb{A}$, let $n(\mathsf{a})$ be the number of codewords ending in a.

Claim. For some $\mathsf{a} \in \mathbb{A}$ we have $n(\mathsf{a}) \geqslant M/q$.

Proof. Suppose not, i.e. $n(\mathsf{a}) < M/q$ for all $\mathsf{a} \in \mathbb{A}$. Then we get $\sum_{\mathsf{a}\in\mathbb{A}} n(\mathsf{a}) < M$. But $\sum_{\mathsf{a}\in\mathbb{A}} n(\mathsf{a})$ is the number of codewords, which is M. Contradiction.

So take some a such that $n(\mathsf{a}) \geqslant M/q$, and let C' denote the set of codewords ending in a. For each $x \in C'$, define $\overline{x}$ to be the word obtained by deleting the last symbol from x, and then define $\overline{C} = \{\overline{x} \mid x \in C'\}$.

Claim. For $x, y \in C'$ with $x \neq y$, we have $d(\overline{x}, \overline{y}) \geqslant d$.

Proof. We have $d(x,y) \geqslant d$, so x and y differ in at least d positions. Furthermore, none of these positions is the last position, since x and y both have an a here. So x and y differ in at least d positions among the first $n - 1$ positions, which means that $\overline{x}$ and $\overline{y}$ differ in at least d places. □

The first consequence of this claim is that if $x, y \in C'$ with $x \neq y$, then $\overline{x} \neq \overline{y}$. So $|\overline{C}| = |C'| = n(\mathsf{a})$. The second consequence is that $d(\overline{C}) \geqslant d$. So $\overline{C}$ is an $(n-1, n(\mathsf{a}), d)$-code. □

2.5 The Plotkin bound

The Plotkin bound is more complicated, but more useful. There is a version for arbitrary q, but we'll address only binary codes. First, we prove some elementary inequalities that we shall need later.

Lemma 2.10. *Suppose N, M are integers with $0 \leqslant N \leqslant M$. Then*

$$N(M-N) \leqslant \begin{cases} \dfrac{M^2}{4} & (\textit{if } M \textit{ is even}) \\ \dfrac{M^2-1}{4} & (\textit{if } M \textit{ is odd}) \end{cases}.$$

Proof. The graph of $N(M-N)$ is an unhappy quadratic with its turning point at $N = M/2$, so to maximise it we want to make N as near as possible to this (remembering that N must be an integer). If M is even, then we can take $N = M/2$, while if M is odd we take $N = (M-1)/2$. □

Now we need to recall some notation: remember that if $x \in \mathbb{R}$, then $\lfloor x \rfloor$ is the largest integer which is less than or equal to x.

Lemma 2.11. *If $x \in \mathbb{R}$, then $\lfloor 2x \rfloor \leqslant 2\lfloor x \rfloor + 1$.*

Proof. Let $y = \lfloor x \rfloor$; then $x < y + 1$. So $2x < 2y + 2$, so $\lfloor 2x \rfloor \leqslant 2y + 1$. □

Now we can state the Plotkin bound – there are two cases, depending on whether d is even or odd. But in fact either one of these can be recovered from the other, using Theorem 2.3.

Theorem 2.12 (Plotkin bound)**.**

1. *If d is even and $n < 2d$, then*
$$A_2(n,d) \leqslant 2\left\lfloor \frac{d}{2d-n} \right\rfloor.$$
2. *If d is odd and $n < 2d+1$, then*
$$A_2(n,d) \leqslant 2\left\lfloor \frac{d+1}{2d+1-n} \right\rfloor.$$

The proof is a double-counting argument. Suppose C is a binary (n, M, d)-code. We suppose that our alphabet is $\{\mathtt{0}, \mathtt{1}\}$, and if $v = (v_1 \dots v_n)$ and $w = (w_1 \dots w_n)$ are codewords, then we define $v + w$ to be the word $(v_1 + w_1)(v_2 + w_2) \dots (v_n + w_n)$, where we do addition modulo 2 (so $\mathtt{1} + \mathtt{1} = \mathtt{0}$).

A really useful feature of this addition operation is the following.

Lemma 2.13. *Suppose v, are binary words of length n. Then $d(v, w)$ is the number of $\mathtt{1}$s in $v + w$.*

Proof. By looking at the possibilities for v_i and w_i, we see that

$$(v+w)_i = \begin{cases} \mathtt{0} & (\text{if } v_i = w_i) \\ \mathtt{1} & (\text{if } v_i \neq w_i) \end{cases}.$$

So

$$\begin{aligned} d(v,w) &= |\{i \mid v_i \neq w_i\}| \\ &= |\{i \mid (v+w)_i = \mathtt{1}\}|. \end{aligned}$$

□

Now we write down an $\binom{M}{2}$ by n array A whose rows are all the words $v+w$ for pairs of distinct codewords v, w. We're going to count the number of 1s in this array in two different ways.

Lemma 2.14. *The number of* 1*s in A is at most*

$$\begin{cases} n\dfrac{M^2}{4} & (\textit{if } M \textit{ is even}) \\ n\dfrac{M^2-1}{4} & (\textit{if } M \textit{ is odd}) \end{cases}.$$

Proof. We count the number of 1s in each column. The word $v+w$ has a 1 in the jth position if and only if one of v and w has a 1 in the jth position, and the other has a 0. If we let N be the number of codewords which have a 1 in the jth position, then the number ways of choosing a pair v, w such that $v+w$ has a 1 in the jth position is $N(M-N)$. So the number of 1s in the jth column of our array is

$$N(M-N) \leqslant \begin{cases} \dfrac{M^2}{4} & (\text{if } M \text{ is even}) \\ \dfrac{M^2-1}{4} & (\text{if } M \text{ is odd}) \end{cases}$$

by Lemma 2.10. This is true for every j, so by adding up we obtain the desired inequality. □

Now we count the 1s in A in a different way.

Lemma 2.15. *The number of* 1*s in A is at least* $d\binom{M}{2}$.

Proof. We look at the number of 1s in each row. The key observation is that if v, w are codewords, then the number of 1s in $v+w$ is $d(v,w)$, and this is at least d. So there are at least d 1s in each row, and hence at least $d\binom{M}{2}$ altogether. □

Proof of the Plotkin bound. We assume first that d is even. Suppose we have a binary (n, M, d)-code C, and construct the array as above. Now we simply combine the inequalities of Lemma 2.14 and Lemma 2.15. There are two cases, according to whether M is even or odd.

Case 1: M even By combining the two inequalities, we get

$$\begin{aligned} & d\binom{M}{2} \leqslant n\frac{M^2}{4} \\ \Rightarrow \quad & \frac{dM^2}{2} - \frac{dM}{2} \leqslant \frac{nM^2}{4} \\ \Rightarrow \quad & (2d-n)M^2 \leqslant 2dM. \end{aligned}$$

By assumption, $2d-n$ and M are both positive, so we divide both sides by $(2d-n)M$ to get

$$M \leqslant \frac{2d}{2d-n}.$$

But M is an integer, so in fact

$$M \leqslant \left\lfloor \frac{2d}{2d-n} \right\rfloor \leqslant 2\left\lfloor \frac{d}{2d-n} \right\rfloor + 1;$$

since M is even and $2\left\lfloor \frac{d}{2d-n} \right\rfloor + 1$ is odd, we can improve this to

$$M \leqslant 2\left\lfloor \frac{d}{2d-n} \right\rfloor.$$

Case 2: *M* odd We combine the two inequalities to get

$$d\binom{M}{2} \leqslant n\frac{M^2-1}{4},$$

or, dividing through by $M - 1 > 0$,

$$d\,\frac{M}{2} \leqslant n\,\frac{M+1}{4}.$$

It follows that $(2d - n)M \leq n$ and hence

$$M \leqslant \frac{n}{2d-n} = \frac{2d}{2d-n} - 1.$$

Now M is an integer, so we get

$$\begin{aligned} M &\leqslant \left\lfloor \frac{2d}{2d-n} - 1 \right\rfloor \\ &= \left\lfloor \frac{2d}{2d-n} \right\rfloor - 1 \\ &\leqslant 2\left\lfloor \frac{d}{2d-n} \right\rfloor + 1 - 1 \\ &= 2\left\lfloor \frac{d}{2d-n} \right\rfloor, \end{aligned}$$

and we have the Plotkin bound for d even.

Now we consider the case where d is odd; but this follows by Theorem 2.3. If d is odd and $n < 2d + 1$, then $d + 1$ is even and $(n + 1) < 2(d + 1)$. So by the even case of the Plotkin bound we have

$$A_2(n+1, d+1) \leqslant \left\lfloor \frac{(d+1)}{2(d+1)-(n+1)} \right\rfloor,$$

and by Theorem 2.3 this equals $A_2(n, d)$. □

3 Error probabilities and nearest-neighbour decoding

3.1 Noisy channels and decoding processes

In this section, we consider the situations in which our codes might be used, and show why we try to get a large distance between codewords. The idea is that we have the following process.

$$\begin{tikzcd}
\boxed{\text{codeword}} \arrow[d, "\text{Noisy channel}"'] \\
\boxed{\text{distorted word}} \arrow[d, "\text{Decoding process}"'] \\
\boxed{\text{codeword}}
\end{tikzcd}$$

We'd like the codeword at the bottom to be the same as the codeword at the top as often as possible. This relies on a good choice of code, and a good choice of decoding process. Most of this course is devoted to looking at good codes, but here we look at decoding processes. Given a code C of length n over the alphabet $\mathbb{A}$, a *decoding process* is simply a function from $\mathbb{A}^n$ to C – given a received word, we try to 'guess' which word was sent.

We make certain assumptions about our noisy channel, namely that all errors are independent and equally likely. This means that there is some error probability p such that any transmitted symbol $\mathtt{a}$ will be transmitted correctly with probability $1 - p$, or incorrectly with probability p, and that if there is an error then all the incorrect symbols are equally likely. Moreover, errors on different symbols are independent – whether an error occurs in one symbol has no effect on whether errors occur in later symbols. We also assume that $p \leqslant \frac{1}{2}$.

Suppose we have a decoding process $f : \mathbb{A}^n \to C$. We say that f is a *nearest-neighbour decoding process* if for all $w \in \mathbb{A}^n$ and all $v \in C$ we have

$$d(w, f(w)) \leqslant d(w, v).$$

This means that for any received word, we decode it using the nearest codeword. Note that some codewords may be equally near, so there may be several different nearest-neighbour decoding processes for a given code.

Example. Let C be the binary repetition code of length 5:

$$\{\mathtt{00000}, \mathtt{11111}\}.$$

Let f be the decoding process

$$w \mapsto \begin{cases} \mathtt{00000} & \text{(if } w \text{ contains at least three } \mathtt{0}\text{s)} \\ \mathtt{11111} & \text{(if } w \text{ contains at least three } \mathtt{1}\text{s)} \end{cases}.$$

Then f is the unique nearest-neighbour decoding process for C.

Given a code and a decoding process, we consider the *word error probability*: given a codeword w, what is the probability that after distortion and decoding, we end up with a different codeword? Let's calculate this for the above example, with $w = \mathtt{00000}$. It's clear that this will be decoded wrongly if at least three of the symbols are changed into $\mathtt{1}$s. If the error probability of the channel is p, then the probability that this happens is

$$p^3(1-p)^2\binom{5}{3} + p^4(1-p)\binom{5}{4} + p^5\binom{5}{5} = 6p^5 - 15p^4 + 10p^3.$$

For example, if $p = \frac{1}{4}$, then the word error probability is only about 0.104.

In general, word error probability depends on the particular word, and we seek a decoding process which minimises the maximum word error probability. It can be shown that the best decoding process in this respect is always a nearest-neighbour decoding process (remembering our assumption that $p \leqslant \frac{1}{2}$).

3.2 Rates of transmission and Shannon's Theorem

Given a q-ary (n, M, d)-code C, we define the *rate* of C to be

$$\frac{\log_q M}{n};$$

this can be interpreted as the ratio of 'useful information' to 'total information' transmitted. For example, the q-ary repetition code of length 3 is a $(3, q, 3)$-code, so has rate $\frac{1}{3}$. The useful information in a codeword can be thought of as the first digit – the rest of the digits are just redundant information included to reduce error probabilities.

Clearly, its good to have a code with a high rate. On the other hand, it's good to have codes with low word error probabilities. Shannon's Theorem says that these two aims can be achieved simultaneously, as long as we use a long enough code. We'll restrict attention to binary codes. We need to define the capacity of our noisy channel – if the channel has error probability p, then the capacity is

$$C(p) = 1 + p\log_2 p + (1 - p)\log_2(1 - p).$$

Theorem 3.1 (Shannon's Theorem)**.** *Suppose we have a noisy channel with capacity C. Suppose ϵ and ρ are positive real numbers with $\rho < C$. Then for any sufficiently large n, there exists a binary code of length n and rate at least ρ and a decoding process such that the word error probability is at most ϵ.*

What does the theorem say? It says that as long as the rate of our code is less than the capacity of the channel, we can make the word error probability as small as we like. The proof of this theorem is well beyond the scope of this course.

Note that if $p = \frac{1}{2}$, then $C(p) = 0$. This reflects the fact that it is hopeless transmitting through such a channel – given a received word, the codeword sent is equally likely to be any codeword.

4 Linear codes

For the rest of the course, we shall be restricting our attention to linear codes; these are codes in which the alphabet $\mathbb{A}$ is a finite field, and the code itself forms a vector space over $\mathbb{A}$. These codes are of great interest because:

- they are easy to describe – we need only specify a basis for our code;
- it is easy to calculate the minimum distance of a linear code – we need only calculate the distance of each word from the word $\mathtt{00}\dots\mathtt{0}$;
- it is easy to decode an error-correcting linear code, via *syndrome decoding*;
- many of the best codes known are linear; in particular, every known non-trivial perfect code has the same parameters (i.e. length, number of codewords and minimum distance) as some linear code.

4.1 Revision of linear algebra

Recall from Linear Algebra the definition of a field: a set $\mathbb{F}$ with distinguished elements 0 and 1 and binary operations + and $\times$ such that:

- $\mathbb{F}$ forms an abelian group under +, with identity element 0 (that is, we have
  - ◇ $a + b = b + a$,
  - ◇ $(a + b) + c = a + (b + c)$,
  - ◇ $a + 0 = a$,
  - ◇ there exists an element $-a$ of $\mathbb{F}$ such that $-a + a = 0$

  for all $a, b, c \in \mathbb{F}$);
- $\mathbb{F} \setminus \{0\}$ forms an abelian group under $\times$, with identity element 1 (that is, we have
  - ◇ $a \times b = b \times a$,
  - ◇ $(a \times b) \times c = a \times (b \times c)$,
  - ◇ $a \times 1 = a$,
  - ◇ there exists an element a^{-1} of $\mathbb{F}$ such that $a^{-1} \times a = 1$

  for all $a, b, c \in \mathbb{F} \setminus \{0\}$);
- $a \times (b + c) = (a \times b) + (a \times c)$ for all $a, b, c \in \mathbb{F}$.

We make all the familiar notational conventions: we may write $a \times b$ as $a.b$ or ab; we write $a \times b^{-1}$ as a/b; we write $a + (-b)$ as $a - b$.

We shall need the following familiar property of fields.

Lemma 4.1. *Let $\mathbb{F}$ be a field, and $a, b \in \mathbb{F}$. Then $ab = 0$ if and only if $a = 0$ or $b = 0$.*

We also need to recall the definition of a vector space. If $\mathbb{F}$ is a field, then a *vector space over* $\mathbb{F}$ is a set V with a distinguished element 0, a binary operation + and a function $\times : (\mathbb{F} \times V) \to V$ (that is, a function which, given an element λ of $\mathbb{F}$ and an element v of V, produces a new element $\lambda \times v$ of V) such that:

- V is an abelian group under + with identity 0;
- for all $\lambda, \mu \in \mathbb{F}$ and $u, v \in V$, we have
  - ◇ $(\lambda \times \mu) \times v = \lambda \times (\mu \times v)$,
  - ◇ $(\lambda + \mu) \times v = (\lambda \times v) + (\mu \times v)$,
  - ◇ $\lambda \times (u + v) = (\lambda \times u) + (\lambda \times v)$,
  - ◇ $1 \times v = v$.

There shouldn't be any confusion between the element 0 of $\mathbb{F}$ and the element 0 of V, or between the different versions of + and $\times$. We use similar notational conventions for + and $\times$ those that we use for fields.

If V is a vector space over $\mathbb{F}$, then a *subspace* is a subset of V which is also a vector space under the same operations. In fact, a subset W of V is a subspace if and only if

- $0 \in W$,
- $v + w \in W$, and
- $\lambda v \in W$

whenever $v, w \in W$ and $\lambda \in \mathbb{F}$.

Suppose V is a vector space over $\mathbb{F}$ and $v_1, \dots, v_n \in V$. Then we say that $v_1, \dots, v_n$ are *linearly independent* if there do not not exist $\lambda_1, \dots, \lambda_n \in \mathbb{F}$ which are not all zero such that

$$\lambda_1 v_1 + \dots + \lambda_n v_n = 0.$$

We define the *span* of $v_1, \dots, v_n$ to be the set of all linear combinations of $v_1, \dots, v_n$, i.e. the set

$$\langle v_1, \dots, v_n \rangle = \{\lambda_1 v_1 + \dots + \lambda_n v_n \mid \lambda_1, \dots, \lambda_n \in \mathbb{F}\}.$$

The span of $v_1, \dots, v_n$ is always a subspace of V. If it the whole of V, we say that $v_1, \dots, v_n$ span V.

We say that V is *finite-dimensional* if there is a finite set $v_1, \dots, v_n$ that spans V. A set $v_1, \dots, v_n$ which is linearly independent and spans V is called a *basis* for V. If V is finite-dimensional, then it has at least one basis, and all bases of V have the same size. This is called the *dimension* of V, which we write as $\dim(V)$.

Suppose V, W are vector spaces over $\mathbb{F}$. A *linear map* from V to W is a function $\alpha : V \to W$ such that

$$\alpha(\lambda u + \mu v) = \lambda\alpha(u) + \mu\alpha(v)$$

for all $\lambda, \mu \in \mathbb{F}$ and $u, v \in V$. If α is a linear map, the *kernel* of α is the subset

$$\ker(\alpha) = \{v \in V \mid \alpha(v) = 0\}$$

of V, and the *image* of α is the subset

$$\mathrm{Im}(\alpha) = \{\alpha(v) \mid v \in V\}$$

of W. $\ker(\alpha)$ is a subspace of V, and we refer to its dimension as the *nullity* of α, which we write $n(\alpha)$. $\mathrm{Im}(\alpha)$ is a subspace of W, and we refer to its dimension as the *rank* of α, which we write $r(\alpha)$. The *Rank–nullity Theorem* says that if α is a linear map from V to W, then

$$n(\alpha) + r(\alpha) = \dim(V).$$

We shall only be interested in one particular type of vector space. For a non-negative integer n, we consider the set $\mathbb{F}^n$, which we think of as the set of column vectors of length n over $\mathbb{F}$, with operations

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ \vdots \\ x_n + y_n \end{pmatrix}$$

and

$$\lambda \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \lambda x_1 \\ \vdots \\ \lambda x_n \end{pmatrix}.$$

$\mathbb{F}^n$ is a vector space over $\mathbb{F}$ of dimension n. Sometimes we will think of the elements of $\mathbb{F}^n$ as row vectors rather than column vectors, or as words of length n over $\mathbb{F}$.

Given m, n and an $n \times m$ matrix A over $\mathbb{F}$, we can define a linear map $\mathbb{F}^m \to \mathbb{F}^n$ by

$$\begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + \dots + a_{1m}x_m \\ \vdots \\ a_{n1}x_1 + \dots + a_{nm}x_m \end{pmatrix}.$$

Every linear map from $\mathbb{F}^m$ to $\mathbb{F}^n$ arises in this way. The *rank* of A is defined to be the rank of this linear map. The *column rank* of A is defined to be the dimension of $\langle c_1, \dots, c_m \rangle$, where $c_1, \dots, c_m$ are the columns of A regarded as vectors in $\mathbb{F}^n$, and the *row rank* is defined to be the dimension of $\langle r_1, \dots, r_n \rangle$, where $r_1, \dots, r_n$ are the rows of A regarded as (row) vectors in $\mathbb{F}^m$. We shall need the result that the rank, row rank and column rank of A are all equal.

Note that when we think of $\mathbb{F}^n$ as the space of row vectors rather than column vectors, we may think of a linear map as being multiplication on the right by an $m \times n$ matrix.

4.2 Finite fields and linear codes

The examples of fields you are most familiar with are $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$. But these are of no interest in this course – we are concerned with *finite fields*. The classification of finite fields goes back to Galois.

Theorem 4.2. *Let q be an integer greater than* 1. *Then a field of order q exists if and only if q is a prime power, and all fields of order q are isomorphic.*

If q is a prime power, then we refer to the unique field of order q as $\mathbb{F}_q$. For example, if q is actually a prime, then $\mathbb{F}_q$ simply consists of the integers mod q, with the operations of addition and multiplication mod q. It is a reasonably easy exercise to show that this really is a field – the hard part is to show that multiplicative inverses exist, and this is a consequence of the Chinese Remainder Theorem.

If q is a prime power but not a prime, then the field $\mathbb{F}_q$ is awkward to describe without developing lots of theory. But this need not worry us – all the explicit examples we meet will be over fields of prime order. Just remember that there is a field of each prime power order. As an example, the addition and multiplication tables for the field of order 4 are given below; we write $\mathbb{F}_4 = \{0, 1, a, b\}$.

$+$	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

$\times$	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

What this means for coding theory is that if we have a q-ary alphabet $\mathbb{A}$ with q a prime power, then we may assume that $\mathbb{A} = \mathbb{F}_q$ (since $\mathbb{A}$ is just a *set* of size q with no additional structure, we lose nothing by re-labelling the elements of $\mathbb{A}$ as the elements of $\mathbb{F}_q$) and we gets lots of extra structure on $\mathbb{A}$ (i.e. the structure of a field) and on $\mathbb{A}^n = \mathbb{F}_q^n$ (the structure of a vector space).

Definition. Assume that q is a prime power and that $\mathbb{A} = \mathbb{F}_q$. A *linear code* over $\mathbb{A}$ is a subspace of $\mathbb{A}^n$.

Example. The binary $(5, 4, 3)$-code

$$\{\mathtt{00000}, \mathtt{01101}, \mathtt{10110}, \mathtt{11011}\}$$

that we saw earlier is linear. To check this, we have to show that it is closed under addition and scalar multiplication. Scalar multiplication is easy: the only elements of $\mathbb{F}_2$ are $\mathtt{0}$ and $\mathtt{1}$, and we have

$$\mathtt{0}x = \mathtt{00000}, \qquad \mathtt{1}x = x$$

for any codeword x. For addition, notice that we have $x + x = \mathtt{00000}$ and $x + \mathtt{00000} = x$ for any x, and

$$\begin{aligned}\mathtt{01101} + \mathtt{10110} &= \mathtt{11011},\\ \mathtt{01101} + \mathtt{11011} &= \mathtt{10110},\\ \mathtt{10110} + \mathtt{11011} &= \mathtt{10110}.\end{aligned}$$

4.3 The minimum distance of a linear code

One of the advantages of linear codes that we mentioned earlier is that it's easy to find the minimum distance of a linear code. Given a code C and a codeword x, we define the *weight* weight(x) of x to be the number of *non-zero* symbols in x.

Lemma 4.3. *Suppose x, y and z are codewords in a linear code over $\mathbb{F}_q$, and λ is a non-zero element of $\mathbb{F}_q$. Then:*

1. *$d(x + z, y + z) = d(x, y)$;*
2. *$d(\lambda x, \lambda y) = d(x, y)$;*
3. *$d(x, y) = \mathrm{weight}(x - y)$.*

Proof. We write $x = x_1 \dots x_n, y = y_1 \dots y_n, z = z_1 \dots z_n$.

1. The ith symbol of $x + z$ is $x_i + z_i$, and the ith symbol of $y + z$ is $y_i + z_i$. So we have

   $$d(x + z, y + z) = |\{i \mid x_i + z_i \neq y_i + z_i\}.$$

   Now for any x_i, y_i, z_i we have $x_i + z_i \neq y_i + z_i$ if and only if $x_i \neq y_i$ (since we can just add or subtract z_i to/from both sides). So

   $$\begin{aligned}d(x + z, y + z) &= \{i \mid x_i \neq y_i\}\\ &= d(x, y).\end{aligned}$$

2. The ith symbol of λx is λx_i, and the ith symbol of λy is λy_i, so

   $$d(\lambda x, \lambda y) = |\{i \mid \lambda x_i \neq \lambda y_i\}.$$

   Now since $\lambda \neq 0$ we have $\lambda x_i \neq \lambda y_i$ if and only if $x_i \neq y_i$ (since we can multiply both sides by λ or λ^{-1}). So we find

   $$\begin{aligned}d(\lambda x, \lambda y) &= |\{i \mid x_i \neq y_i\}|\\ &= d(x, y).\end{aligned}$$

3. Clearly weight$(x-y) = d(x-y, 0)$, which equals $d(x, y)$ by part (1).

□

Corollary 4.4. *The minimum distance of a linear code C equals the minimum weight of a non-zero codeword in C.*

Proof. It suffices to show that, for any δ,

(C contains distinct codewords x, y with $d(x, y) = \delta$) $\Leftrightarrow$ (C contains a non-zero codeword x with weight$(x) = \delta$).

($\Leftarrow$) C must contain the zero element of $\mathbb{F}_q^n$, namely the word $\mathtt{00}\dots\mathtt{0}$. This is because C must contain *some* word x, and hence must contain $0x = \mathtt{00}\dots\mathtt{0}$. So if x is a non-zero codeword with weight δ, then x and $\mathtt{00}\dots\mathtt{0}$ are distinct codewords with $d(w, \mathtt{00}\dots\mathtt{0}) = \delta$.

($\Rightarrow$) If x, y are distinct codewords with $d(x, y) = \delta$, then $x-y$ is a non-zero codeword with weight$(x-y) = \delta$.

□

4.4 Bases and generator matrices

Another advantage of linear codes that I mentioned above is that you can specify the whole code by specifying a basis. Recall that a basis of a vector space V over $\mathbb{F}$ is a subset $\{e_1, \dots, e_k\}$ such that every $v \in V$ can be *uniquely* written in the form

$$v = \lambda_1 e_1 + \lambda_2 e_2 + \cdots + \lambda_k e_k,$$

where $\lambda_1, \dots, \lambda_k$ are elements of $\mathbb{F}$.

Any two bases have the same size, and this size is called the *dimension* of the code.

Example. The set $\{\mathtt{01101}, \mathtt{11011}\}$ is a basis for the code in the last example.

In general, V will have lots of different bases to choose from. But (recall from Linear Algebra) that any two bases have the same size, and this size we call the *dimension* of V. So the code in the examples above has dimension 2. If a code C is of length n and has dimension k as a vector space, we say that C is an $[n, k]$-code. If in addition C has minimum distance at least d, we may say that C is an $[n, k, d]$-code. So the code in the example above is a binary $[5, 2, 3]$-code.

Lemma 4.5. *A vector space V of dimension k over $\mathbb{F}_q$ contains exactly q^k elements.*

Proof. Suppose $\{e_1, \dots, e_k\}$ is a basis for V. Then, by the definition of a basis, every element of V is uniquely of the form

$$\lambda_1 e_1 + \lambda_2 e_2 + \cdots + \lambda_k e_k,$$

for some choice of $\lambda_1, \dots, \lambda_k \in \mathbb{F}_q$. On the other hand, every choice of $\lambda_1, \dots, \lambda_k$ gives us an element of V, so the number of vectors in V is the number of choices of $\lambda_1, \dots, \lambda_k$. Now there are q ways to choose each λ_i (since there are q elements of $\mathbb{F}_q$ to choose from), and so the total number of choices of these scalars is q^k. □

As a consequence, we see that a q-ary $[n,k,d]$-code is a q-ary (n,q^k,d)-code. This highlights a slight disadvantage of linear codes – their sizes must be powers of q. So if we're trying to find optimal codes for given values of n,d (i.e. (n,M,d)-codes with $M = A_q(n,d)$), then we can't hope to do this with linear codes if $A_q(n,d)$ is not a power of q. In practice (especially for $q = 2$) many of the values of $A_q(n,d)$ are powers of q.

It will be useful in the rest of the course to arrange a basis of our code in the form of a matrix.

Definition. Suppose C is a q-ary $[n,k]$-code. A *generator matrix* for C is a $k \times n$ matrix with entries in $\mathbb{F}_q$, whose rows form a basis for C.

Examples.

1. The binary $[5,2,3]$-code from the last example has various different generator matrices; for example
$$\begin{pmatrix}01101\\10110\end{pmatrix}, \qquad \begin{pmatrix}10110\\11011\end{pmatrix}.$$
2. If q is a prime power, then the q-ary repetition code is linear. It has a generator matrix
$$(11\ldots1).$$
3. Recall the binary parity-check code
$$C = \{v = v_1v_2\ldots v_n \mid v_1,\ldots,v_n \in \mathbb{F}_2, v_1 + \cdots + v_n = 0\}.$$
This has a generator matrix
$$\begin{pmatrix}1 & 0 & \ldots & 0 & 0 & 1\\ 0 & 1 & \ldots & 0 & 0 & 1\\ \vdots & \vdots & \ddots & \vdots & \vdots & 1\\ 0 & 0 & \ldots & 1 & 0 & 1\\ 0 & 0 & \ldots & 0 & 1 & 1\end{pmatrix}.$$

4.5 Equivalence of linear codes

Recall the definition of equivalent codes from earlier: C is equivalent to $\mathcal{D}$ if we can get from C to $\mathcal{D}$ via a sequence of operations of the following two types.

Operation 1: permuting positions Choose a permutation σ of $\{1,\ldots,n\}$, and for $v = v_1\ldots v_n \in A^n$ define
$$v_\sigma = v_{\sigma(1)}\ldots v_{\sigma(n)}.$$
Now replace C with
$$C_\sigma = \{v_\sigma \mid v \in C\}.$$

Operation 2: permuting the symbols in a given position Choose $i \in \{1,\ldots,n\}$ and a permutation f of A. For$v = v_1\ldots v_n \in A^n$, define
$$v_{f,i} = v_1\ldots v_{i-1}(f(v_i))v_{i+1}\ldots v_n.$$
Now replace C with
$$C_{f,i} = \{v_{f,i} \mid v \in C\}.$$

There's a slight problem with applying this to linear codes, which is that if C is linear and $\mathcal{D}$ is equivalent to C, then $\mathcal{D}$ need not be linear. Operation 1 is OK, as we shall now show.

Lemma 4.6. *Suppose C is a linear $[n, k, d]$-code over $\mathbb{F}_q$, and σ is a permutation of $\{1, \dots, n\}$. Then the map*

$$\begin{aligned} \phi : C &\longrightarrow \mathbb{F}_q^n \\ v &\longmapsto v_\sigma \end{aligned}$$

is linear, and C_σ is a linear $[n, k, d]$-code.

Proof. Suppose $v, w \in C$ and $\lambda, \mu \in \mathbb{F}_q$. We need to show that

$$\phi(\lambda v + \mu w) = \lambda\phi(v) + \mu\phi(w),$$

i.e.

$$(\phi(\lambda v + \mu w))_i = (\lambda\phi(v) + \mu\phi(w))_i$$

for every $i \in \{1, \dots, n\}$. We have

$$\begin{aligned} (\phi(\lambda v + \mu w))_i &= ((\lambda v + \mu w)_\sigma)_i \\ &= (\lambda v + \mu w)_{\sigma(i)} \\ &= (\lambda v)_{\sigma(i)} + (\mu w)_{\sigma(i)} \\ &= \lambda(v_{\sigma(i)}) + \mu(w_{\sigma(i)}) \\ &= \lambda(v_\sigma)_i + \mu(w_\sigma)_i \\ &= \lambda(\phi(v))_i + \mu(\phi(w))_i \\ &= (\lambda\phi(v))_i + (\mu\phi(w))_i \\ &= (\lambda\phi(v) + \mu\phi(w))_i, \end{aligned}$$

as required.

Now C_σ is by definition the image of ϕ, and so is a subspace of $\mathbb{F}_q^n$, i.e. a linear code. We know that $d(C_\sigma) = d(C)$ from before, and that $|C_\sigma| = |C|$. This implies $q^{\dim C_\sigma} = q^{\dim C}$ by Lemma 4.5, i.e. $\dim C_\sigma = \dim C = k$, so C_σ is an $[n, k, d]$-code. □

Unfortunately, Operation 2 does not preserve linearity. Here is a trivial example of this. Suppose $q = 2$, $n = 1$ and $C = \{\mathbb{0}\}$. Then C is a linear $[1, 0]$-code. If we choose $i = 1$ and f the permutation which swaps $\mathbb{0}$ and 1, then we have $C_{f,i} = \{1\}$, which is not linear. So we need to restrict Operation 2. We define the following.

Operation 2′ Suppose C is a linear code of length n over $\mathbb{F}_q$. Choose $i \in \{1, \dots, n\}$ and $a \in \mathbb{F}_q \setminus \{\mathbb{0}\}$. For $v = v_1 \dots v_n \in \mathbb{F}_q^n$ define

$$v_{a,i} = v_1 \dots v_{i-1}(av_i)v_{i+1} \dots v_n.$$

Now replace C with the code

$$C_{a,i} = \{v_{a,i} \mid v \in C\}.$$

We want to show that Operation 2′ preserves linearity, dimension and minimum distance. We being by showing that it's a special case of Operation 2.

Lemma 4.7. *If $a \in \mathbb{F}_q \setminus \{\mathbb{0}\}$, the the map*

$$\begin{aligned} f : \mathbb{F}_q &\longrightarrow \mathbb{F}_q \\ x &\longmapsto ax \end{aligned}$$

is a bijection, i.e. a permutation of $\mathbb{F}_q$.

Proof. Since f is a function from a finite set to itself, we need only show that f is injective. If $x, y \in \mathbb{F}_q$ and $f(x) = f(y)$, then we have $ax = ay$. Since $a \neq \{\mathbb{0}\}$, a has an inverse a^{-1}, and we can multiply both sides by a^{-1} to get $x = y$. So f is injective. □

Now we show that the operation which sends v to $v_{a,i}$ is linear, which will mean that it sends linear codes to linear codes.

Lemma 4.8. *Suppose C is a linear $[n, k, d]$-code over $\mathbb{F}_q$, $i \in \{1, \dots, n\}$ and $0 \neq a \in \mathbb{F}_q$. Then the map*

$$\begin{aligned} \phi : \mathbb{F}_q^n &\longrightarrow \mathbb{F}_q^n \\ v &\longmapsto v_{a,i} \end{aligned}$$

is linear, and $C_{a,i}$ is a linear $[n, k, d]$-code over $\mathbb{F}_q$.

Proof. For any vector $v \in \mathbb{F}_q^n$, we have

$$\phi(v)_j = (v_{a,i})_j = \begin{cases} av_j & (j = i) \\ v_j & (j \neq i) \end{cases}.$$

Now take $v, w \in \mathbb{F}_q^n$ and $\lambda, \mu \in \mathbb{F}_q$. We must show that

$$(\phi(\lambda v + \mu w))_j = (\lambda\phi(v) + \mu\phi(w))_j$$

for each $j \in \{1, \dots, n\}$. For $j \neq i$ we have

$$\begin{aligned} (\phi(\lambda v + \mu w))_j &= (\lambda v + \mu w)_j \\ &= (\lambda v)_j + (\mu w)_j \\ &= \lambda v_j + \mu w_j \\ &= \lambda(\phi(v))_j + \mu(\phi(w))_j \\ &= (\lambda\phi(v))_j + (\mu\phi(w))_j, \\ &= (\lambda\phi(v) + \mu\phi(w))_j, \end{aligned}$$

while for $j = i$ we have

$$\begin{aligned} (\phi(\lambda v + \mu w))_j &= a(\lambda v + \mu w)_j \\ &= a((\lambda v)_j + (\mu w)_j) \\ &= a(\lambda v_j + \mu w_j) \\ &= a\lambda v_j + a\mu w_j \\ &= \lambda(av_j) + \mu(aw_j) \\ &= \lambda(\phi(v)_j) + \mu(\phi(w)_j) \\ &= (\lambda\phi(v))_j + (\mu\phi(w))_j \\ &= (\lambda\phi(v) + \mu\phi(w))_j, \end{aligned}$$

as required.

Now $C_{a,i}$ is by definition the image of ϕ, and this is a subspace of $\mathbb{F}_q^n$, i.e. a linear code. We know from before (since Operation 2′ is a special case of Operation 2) that $d(C_{a,i}) = d(C)$ and $|C_{a,i}| = |C|$, and this gives $\dim C_{a,i} = \dim C = k$, so that $C_{a,i}$ is a linear $[n, k, d]$-code. □

In view of these results we re-define equivalence for linear codes: we say that linear codes C and $\mathcal{D}$ are equivalent if we can get from one to the other by applying Operations 1 and 2′ repeatedly.

Example. Let $n = q = 3$, and define

$$C = \{000, 101, 202\},$$
$$\mathcal{D} = \{000, 011, 022\},$$
$$\mathcal{E} = \{000, 012, 021\}.$$

Then C, $\mathcal{D}$ and $\mathcal{E}$ are all $[3, 1]$-codes over $\mathbb{F}_q$ (check this!). We can get from C to $\mathcal{D}$ by swapping the first two positions, and we can get from $\mathcal{D}$ to $\mathcal{E}$ by multiplying everything in the third position by 2. So C, $\mathcal{D}$ and $\mathcal{E}$ are equivalent linear codes.

We'd like to know the relationship between equivalence and generator matrices: if C and $\mathcal{D}$ are equivalent linear codes, how are their generator matrices related? Well, a given code usually has more than one choice of generator matrix, and so first we'd like to know how two different generator matrices for the same code are related.

We define the following operations on matrices over $\mathbb{F}_q$:

MO1. permuting the rows;

MO2. multiplying a row by a non-zero element of $\mathbb{F}_q$;

MO3. adding a multiple of a row to another row.

You should recognise these as the 'elementary row operations' from Linear Algebra. Their importance is as follows.

Lemma 4.9. *Suppose C is a linear $[n, k]$-code with generator matrix G. If the matrix H can be obtained from G by applying the row operations (MO1–3) repeatedly, then H is also a generator matrix for C.*

Proof. Since G is a generator matrix for C, we know that the rows of G are linearly independent and span C. So G has rank k (the number of rows) and row space C. We know from linear algebra that elementary row operations do not affect the rank or the row space of a matrix, so H also has rank k and row space C. So the rows of H are linearly independent and span C, so form a basis for C, i.e. H is a generator matrix for C. □

Now we define two more matrix operations:

MO4. permuting the columns;

MO5. multiplying a column by a non-zero element of $\mathbb{F}_q$.

Lemma 4.10. *Suppose C is a linear $[n, k]$-code over $\mathbb{F}_q$, with generator matrix G. If the matrix H is obtained from G by applying matrix operation 4 or 5, then H is a generator matrix for a code $\mathcal{D}$ equivalent to C.*

Proof. Suppose G has entries g_{jl}, for $1 \leqslant j \leqslant k$ and $1 \leqslant l \leqslant n$. Let $r_1, \dots, r_k$ be the rows of G, i.e.

$$r_j = g_{j1} g_{j2} \dots g_{jn}.$$

By assumption $\{r_1, \dots, r_k\}$ is a basis for C; in particular, the rank of G is k.

Suppose H is obtained using matrix operation 4, applying a permutation σ to the columns. This means that

$$h_{jl} = g_{j\sigma(l)},$$

so row j of H is the word

$$g_{j\sigma(1)} g_{j\sigma(2)} \dots g_{j\sigma(n)}.$$

But this is the word $(r_j)_\sigma$ as defined in equivalence operation 1. So the rows of H lie in the code C_σ, which is equivalent to C.

Now suppose instead that we obtain H by applying matrix operation 5, multiplying column i by $a \in \mathbb{F}_q \setminus \{0\}$. This means that

$$h_{jl} = \begin{cases} g_{jl} & (l \neq i) \\ a g_{jl} & (l = i) \end{cases},$$

so that row j of H is the word

$$g_{j1} g_{j2} \dots g_{j(i-1)} (a g_{ji}) g_{j(i+1)} \dots g_{jn}.$$

But this is the word $(r_j)_{a,i}$ as defined in equivalence operation 2. So the rows of H lie in the code $C_{a,i}$, which is equivalent to C.

For either matrix operation, we have seen that the rows of H lie in a code $\mathcal{D}$ equivalent to C. We need to know that they form a basis for $\mathcal{D}$. Since there are k rows and $\dim(\mathcal{D}) = \dim(C) = k$, it suffices to show that the rows of H are linearly independent, i.e. to show that H has rank k. But matrix operations 4 and 5 are elementary column operation, and we know from linear algebra that these don't affect the rank of a matrix. So $\mathrm{rank}(H) = \mathrm{rank}(G) = k$. □

We can summarise these results as follows.

Proposition 4.11. *Suppose C is a linear $[n, k]$-code with a generator matrix G, and that the matrix H is obtained by applying a sequence of matrix operations 1–5. Then H is a generator matrix for a code $\mathcal{D}$ equivalent to C.*

Proof. By applying matrix operations 1–3, we get a new generator matrix for C, by Lemma 4.9, and C is certainly equivalent to itself. By applying matrix operations 4 and 5, we get a generator matrix for a code equivalent to C, by Lemma 4.10. □

Note that in the list of matrix operations 1–5, there is a sort of symmetry between rows and columns. In fact, you might expect that you can do another operation

MO6. adding a multiple of a column to another column

but you *can't*. Doing this can take you to a code with a different minimum distance. For example, suppose $q = 2$, and that C is the parity-check code of length 3:

$$C = \{\mathtt{000}, \mathtt{011}, \mathtt{101}, \mathtt{110}\}.$$

We have seen that $d(C) = 2$ and that C has a generator matrix

$$G = \begin{pmatrix} 101 \\ 011 \end{pmatrix}.$$

If we applied operation 6 above, adding column 1 to column 2, we'd get the matrix

$$H = \begin{pmatrix} 111 \\ 011 \end{pmatrix}.$$

This is a generator matrix for the code

$$\{000, 111, 011, 100\},$$

which has minimum distance 1, so is not equivalent to C. So the difference between 'row operations' and 'column operations' is critical.

Armed with these operations, we can define a standard way in which we can write generator matrices.

Definition. Let G be a $k \times n$ matrix over $\mathbb{F}_q$, with $k \leqslant n$. We say that G is in *standard form* if

$$G = (I_k|A),$$

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

For example, the generator matrix for the binary parity-check code given above is in standard form.

Lemma 4.12. *Suppose G is a $k \times n$ matrix over $\mathbb{F}_q$ whose rows are linearly independent. By applying matrix operations 1–5, we may transform G into a matrix in standard form.*

Proof. For $i = 1, \dots, k$ we want to transform column i into

$$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where the 1 is in position i. Suppose we have already done this for columns $1, \dots, i-1$.

Step 1. Since the rows of our matrix are linearly independent, there must be some non-zero entry in the ith row. Furthermore, by what we know about columns $1, \dots, i-1$, this non-zero entry must occur in one of columns $i, \dots, n$. So we apply matrix operation 4, permuting columns $i, \dots, n$ to get a non-zero entry in the (i,i)-position of our matrix.

Step 2. Suppose the (i,i)-entry of our matrix is $\mathtt{a} \neq 0$. Then we apply matrix operation 2, multiplying row i of our matrix by $\mathtt{a}^{-1}$, to get a 1 in the (i,i)-position. Note that this operation does not affect columns $1, \dots. i-1$.

Step 3. We now apply matrix operation 3, adding multiples of row i to the other rows in order to 'kill' the remaining non-zero entries in column i. Note that this operation does not affect columns $1, \dots . i-1$.

By applying Steps 1–3 for $i = 1, \dots, k$ in turn, we get a matrix in standard form. Note that it is automatic from the proof that $k \leqslant n$. □

Corollary 4.13. *Suppose C is a linear $[n,k]$-code over $\mathbb{F}_q$. Then C is equivalent to a code with a generator matrix in standard form.*

Proof. G has linearly independent rows, so by Lemma 4.12 we can transform G into a matrix H in standard form using matrix operations 1–5. By Proposition 4.11, H is a generator matrix for a code equivalent to C. □

4.6 Decoding with a linear code

Recall the notion of a nearest-neighbour decoding process from §3. For linear codes, we can find nearest-neighbour decoding processes in an efficient way, using cosets.

Definition. Suppose C is an $[n,k]$-code over $\mathbb{F}_q$. For $v \in \mathbb{F}_q^n$, define

$$v + C = \{v + w \mid w \in C\}.$$

The set $v + C$ is a called a *coset* of C.

You should remember the word coset from group theory, and this is exactly what cosets are here – cosets of the group C as a subgroup of $\mathbb{F}_q^n$.

Example. Let $q = 3$, and consider the $[2,1]$-code

$$C = \{00, 12, 21\}.$$

The cosets of C are

$$00 + C = \{00, 12, 21\},$$

$$11 + C = \{11, 20, 02\},$$

$$22 + C = \{22, 01, 10\}.$$

The crucial property of cosets is as follows.

Proposition 4.14. *Suppose C is an $[n,k]$-code over $\mathbb{F}_q$. Then:*

1. *every coset of C contains exactly q^k words;*
2. *every word in $\mathbb{F}_q^n$ is contained in some coset of C;*
3. *if the word v is contained in the coset $u + C$, then $v + C = u + C$;*
4. *any word in $\mathbb{F}_q^n$ is contained in at most one coset of C;*
5. *there are exactly q^{n-k} cosets of C.*

Proof.

1. C contains q^k words, and the map from C to $v + C$ given by $w \mapsto v + w$ is a bijection.

2. The word v is contained in $v + C$, since $v = 0 + v$ and $0 \in C$.

3. Since $v \in u + C$, we have $v = u + x$ for some $x \in C$. If $w \in v + C$, then $w = v + y$ for some $y \in C$, so $w = u + (x + y) \in u + C$. So $v + C$ is a subset of $u + C$; since they both have the same size, they must be equal.

4. Suppose $u \in v + C$ and $u \in w + C$. Then we have $u = v + x$ and $u = w + y$ for some $x, y \in C$. Since C is closed under addition, we have $y - x \in C$, so we find that $v = w + (y - x) \in w + C$. Hence $v + C = w + C$ by part (3).

5. There are q^n words altogether in $\mathbb{F}_q^n$, and each of them is contained in exactly one coset of C. Each coset has size q^k, and so the number of cosets must be $q^n/q^k = q^{n-k}$.

□

Given a linear $[n, k]$-code C and a coset $w + C$, we define a *coset leader* to be a word of minimal weight in $w + C$. Now we define a *Slepian array* to be a $q^{n-k} \times q^k$ array, constructed as follows:

- choose one leader from each coset (note that the word $\mathtt{00}\dots\mathtt{0}$ must be the leader chosen from the coset $\mathtt{00}\dots\mathtt{0} + C = C$, since it has smaller weight than any other word);

- in the first row of the array, put all the codewords, with $\mathtt{00}\dots\mathtt{0}$ at the left and the other codewords in any order;

- in the first column put all the coset leaders – the word $\mathtt{00}\dots\mathtt{0}$ is at the top, and the remaining leaders go in any order;

- now fill in the remaining entries by letting the entry in row i and column j be

(leader at the start of row i) + (codeword at the top of column j).

Example. For the code in the last example, we may choose `00`, `02` and `10` as coset leaders, and draw the Slepian array

$$\begin{array}{ccc} \mathtt{00} & \mathtt{12} & \mathtt{21} \\ \mathtt{02} & \mathtt{11} & \mathtt{20} \\ \mathtt{10} & \mathtt{22} & \mathtt{01} \end{array}.$$

Lemma 4.15. *In a Slepian array, every word appears once.*

Proof. Let v be a word in $\mathbb{F}_q^n$. The v lies in some coset $x + C$, by Proposition 4.14(2). Let y be the chosen leader for this coset; then y appears in column 1, in row i, say. Since $y \in x + C$, we have $y + C = x + C$, by Proposition 4.14(3). So $v \in y + C$, and so we can write $v = y + u$, where $u \in C$. u lies in row 1 of the array, in column j, say, and so v lies in row i and column j of the array. □

Now we show how to use a Slepian array to construct a decoding process. Let C be an $[n, k]$-code over $\mathbb{F}_q$, and let S be a Slepian array for C. We define a decoding process $f : \mathbb{F}_q^n \to C$ as follows. For $v \in \mathbb{F}_q^n$, we find v in the array S (which we can do, by Lemma 4.15). Now we let $f(v)$ be the codeword at the top of the same column as v.

Theorem 4.16. *f is a nearest-neighbour decoding process.*

Proof. We need to show that for any $v \in \mathbb{F}_q^n$ and any $w \in C$,

$$d(v, f(v)) \leqslant d(v, w).$$

Find v in the Slepian array, and let u be the word at the start of the same row as v. Then, by the construction of the Slepian array,

$$v = u + f(v) \in u + C.$$

This gives

$$v - w = u + (f(v) - w) \in u + C.$$

Of course $u \in u + C$, and u was chosen to be a leader for this coset, which means that

$$\text{weight}(u) \leqslant \text{weight}(v - w).$$

So (by Lemma 4.3)

$$d(v, f(v)) = \text{weight}(u) \leqslant \text{weight}(v - w) = d(v, w).$$

□

Example. Let $q = 2$, and consider the repetition code

$$C = \{\texttt{000}, \texttt{111}\}.$$

A Slepian array for this is

$$\begin{array}{cc} \texttt{000} & \texttt{111} \\ \texttt{001} & \texttt{110} \\ \texttt{010} & \texttt{101} \\ \texttt{100} & \texttt{011} \end{array}.$$

So if we receive the word `101`, we decode it as `111`, and as long as no more than one error has been made in transmission, this is right. Notice, in fact, that all the words in the first column are distance 1 away from `000`, and all the words in the second column are distance 1 away from `111`.

It looks as though there's a problem with constructing Slepian arrays, which is that we need to write out all the cosets of C beforehand so that we can find coset leaders. In fact, we don't.

Algorithm for constructing a Slepian array

Row 1: Write the codewords in a row, with the word `00`...`0` at the start, and the remaining codewords in any order.

The other rows: For the remaining rows: if you've written every possible word, then stop. Otherwise, choose a word u that you haven't written yet *of minimal weight*. Put u at the start of the row, and then for the rest of the row, put

$$u + (\text{codeword at the top of column } j)$$

in column j.

This is a much better way to construct Slepian arrays, since we only need to know the code, not the cosets. However, we'll see that we can do better than to use a Slepian array in the next section.

5 Dual codes and parity-check matrices

5.1 The dual code

We begin by defining a scalar product on $\mathbb{F}_q^n$. Given words

$$x = (x_1 \dots x_n), \qquad y = (y_1 \dots y_n),$$

we define

$$x.y = x_1y_1 + \cdots + x_ny_n \in \mathbb{F}_q.$$

You should be familiar with this from linear algebra.

Lemma 5.1. *The scalar product . is symmetric and bilinear, i.e.*

$$v.w = w.v$$

and

$$(\lambda v + \mu v').w = \lambda(v.w) + \mu(v'.w)$$

for all words v, v', w and all $\lambda, \mu \in \mathbb{F}_q$.

Proof.

$$\begin{aligned} v.w &= \sum_{i=1}^{n} v_iw_i \\ &= \sum_{i=1}^{n} w_iv_i \end{aligned}$$

(since multiplication in $\mathbb{F}_q$ is commutative)

$$= w.v.$$

Also,

$$\begin{aligned} (\lambda v + \mu v').w &= \sum_{i=1}^{n} (\lambda v + \mu v')_i w_i \\ &= \sum_{i=1}^{n} (\lambda v_i + \mu v'_i) w_i \\ &= \lambda \sum_{i=1}^{n} v_iw_i + \mu \sum_{i=1}^{n} v'_iw_i \end{aligned}$$

(since multiplication in $\mathbb{F}_q$ is distributive over addition)

$$= \lambda(v.w) + \mu(v'.w).$$

□

Now, given a subspace C of $\mathbb{F}_q^n$, we define the *dual code* to be

$$C^\perp = \{w \in \mathbb{F}_q^n \mid v.w = \mathbb{0} \text{ for all } v \in C\}.$$

In linear algebra, we would call $C^\perp$ the subspace *orthogonal* to C. We'd like a simple criterion that tells us whether a word lies in $C^\perp$.

Lemma 5.2. *Suppose C is a linear $[n,k]$-code and G is a generator matrix matrix for C. Then*

$$w \in C^{\perp} \quad \Leftrightarrow \quad Gw^{\mathrm{T}} = 0.$$

Note that we think of the elements of $\mathbb{F}_q^n$ as row vectors; if w is the row vector $(w_1 \dots w_n)$, then w^{T} is the column vector

$$\begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}.$$

G is a $k \times n$ matrix, so Gw^{T} is a column vector of length k.

Proof. Suppose G has entries g_{ij}, for $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n$. Let g_i denote the ith row of G. Then $g_1, \dots, g_k$ are words which form a basis for C, and

$$g_i = g_{i1} \dots g_{in}.$$

Now

$$\begin{aligned}(Gw^{\mathrm{T}})_i &= \sum_{j=1}^{n} g_{ij} w_j \\ &= g_i.w,\end{aligned}$$

so $Gw^{\mathrm{T}} = 0$ if and only if $g_i.w = 0$ for all i.

($\Rightarrow$) If $w \in C^{\perp}$, then $v.w = 0$ for all $v \in C$. In particular, $g_i.w = 0$ for $i = 1, \dots, k$. So $Gw^{\mathrm{T}} = 0$.

($\Leftarrow$) If $Gw^{\mathrm{T}} = 0$, then $g_i.w = 0$ for $i = 1, \dots, k$. Now $g_1, \dots, g_k$ form a basis for C, so any $v \in C$ can be written as

$$v = \lambda_1 g_1 + \dots + \lambda_k g_k$$

for $\lambda_1, \dots, \lambda_k \in \mathbb{F}_q$. Then

$$\begin{aligned}v.w &= (\lambda_1 g_1 + \dots + \lambda_k g_k).w \\ &= \lambda_1(g_1.w) + \dots + \lambda_k(g_k.w)\end{aligned}$$

by Lemma 5.1

$$= 0 + \dots + 0.$$

So $w \in C^{\perp}$.

□

This lemma gives us another way to think of $C^{\perp}$ – it is the kernel of *any* generator matrix of C.

Examples.

1. Suppose $q = 3$ and C is the repetition code $\{000, 111, 222\}$. This has a 1×3 generator matrix $\begin{pmatrix}1 & 1 & 1\end{pmatrix}$, so

$$\begin{aligned}C^{\perp} &= \{w \in \mathbb{F}_3^3 \mid 111.w = 0\} \\ &= \{w \in \mathbb{F}_3^3 \mid w_1 + w_2 + w_3 = 0\} \\ &= \{000, 012, 021, 102, 111, 120, 201, 210, 222\},\end{aligned}$$

the linear $[3,2]$-code with basis $\{210, 201\}$.

2. Let $q = 2$ and $C = \{0000, 0101, 1010, 1111\}$. This has generator matrix
$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$
and we have
$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} w_1 + w_3 \\ w_2 + w_4 \end{pmatrix}.$$
So $C^\perp$ is the set of all words w with $w_1 + w_3 = w_2 + w_4 = 0$, i.e.
$$C^\perp = \{0000, 0101, 1010, 1111\}.$$
So $C^\perp = C$. This is something which can't happen for real vector spaces and their orthogonal complements.

3. Let $q = 2$, and $C = \{000, 001, 110, 111\}$. Then C has generator matrix
$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix},$$
and we may check that $C^\perp = \{000, 110\}$. So in this case $C^\perp < C$.

Note that in all these examples, $C^\perp$ is a subspace of $\mathbb{F}_q^n$, i.e. a linear code. In fact, this is true in general.

Theorem 5.3. *If C is an $[n, k]$-code over $\mathbb{F}_q$, then $C^\perp$ is an $[n, n-k]$-code over $\mathbb{F}_q$.*

Proof. Let G be a generator matrix of C. Then Lemma 5.2 says that $C^\perp = \ker(G)$. So by the rank–nullity theorem $C^\perp$ is a subspace of $\mathbb{F}_q^n$, i.e. a linear code, and the dimension of $C^\perp$ is n minus the rank of G. Recall that the rank of a matrix is the maximum l such that G has a set of l linearly independent rows. Well, G has k rows, and since they form a basis for C, they must be linearly independent. So G has rank k, and the theorem is proved. □

Example. Suppose $q = 2$ and C is the repetition code $\{00\ldots0, 11\ldots1\}$ of length n. G has generator matrix
$$\begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix},$$
and so
$$C^\perp = \{v \in \mathbb{F}_2^n \mid v_1 + \cdots + v_n = 0\},$$
the parity-check code. By Theorem 5.3, this is an $[n, n-1]$-code, so contains 2^{n-1} words.

Theorem 5.4. *Let C be a linear code. Then $(C^\perp)^\perp = C$.*

Proof.

Claim. $C \subseteq (C^\perp)^\perp$.

Proof. $C^\perp = \{w \in \mathbb{F}_q \mid v.w = 0 \text{ for all } v \in C\}$. This means that $v.w = 0$ for all $v \in C$ and $w \in C^\perp$. Another way of saying this is that if $v \in C$, then $w.v = 0$ for all $w \in C^\perp$. So

$$v \in \{x \mid w.x = 0 \text{ for all } w \in C^\perp\} = (C^\perp)^\perp.$$

If C is an $[n, k]$-code, then Theorem 5.3 says that $C^\perp$ is an $[n, n-k]$-code. By applying Theorem 5.3 again, we find that $(C^\perp)^\perp$ is an $[n, k]$-code. So we have $(C^\perp)^\perp \geqslant C$ and $\dim(C^\perp)^\perp = \dim C$ have the same dimension, and so $(C^\perp)^\perp = C$. □

Now we make a very important definition.

Definition. Let C be a linear code. A *parity-check matrix* for C is a generator matrix for $C^\perp$.

We will see in the rest of the course that parity-check matrices are generally more useful than generator matrices. Here is an instance of this.

Lemma 5.5. *Let C be a code, H a parity-check matrix for C, and v a word in $\mathbb{F}_q^n$. Then $v \in C$ if and only if $Hv^{\mathrm{T}} = 0$.*

Proof. By Theorem 5.4 we have $v \in C$ if and only if $v \in (C^\perp)^\perp$. Now H is a generator matrix for $C^\perp$, and so by Lemma 5.2 we have $w \in (C^\perp)^\perp$ if and only if $Hw^{\mathrm{T}} = 0$. □

But can we find a parity-check matrix? The following lemma provides a start.

Lemma 5.6. *Suppose C is a linear $[n, k]$-code with generator matrix G, and let H be any $n-k$ by n matrix. Then H is a parity-check matrix for C if and only if*

- *the rows of H are linearly independent, and*
- *$GH^{\mathrm{T}} = 0$.*

Proof. Let $h_1, \dots, h_{n-k}$ be the rows of H. Then the ith column of GH^{T} is Gh_i^{T}, so $GH^{\mathrm{T}} = 0$ if and only if $Gh_i^{\mathrm{T}} = 0$ for $i = 1, \dots, n-k$.

($\Rightarrow$) If H is a parity-check matrix for C, then it is a generator matrix for $C^\perp$, so its rows $h_1, \dots, h_{n-k}$ form a basis for $C^\perp$; in particular, they are linearly independent. Also, since $h_1, \dots, h_{n-k}$ lie in $C^\perp$, we have $Gh_i^{\mathrm{T}} = 0$ for each i by Lemma 5.2, so $GH^{\mathrm{T}} = 0$.

($\Leftarrow$) Suppose that $GH^{\mathrm{T}} = 0$ and the rows of H are linearly independent. Then $Gh_i^{\mathrm{T}} = 0$ for each i, and so $h_1, \dots, h_{n-k}$ lie in $C^\perp$ by Lemma 5.2. So the rows of H are linearly independent words in $C^\perp$. But the dimension of $C^\perp$ is $n-k$ (the number of rows of H), so in fact these rows form a basis for $C^\perp$, and hence H is a generator matrix for $C^\perp$, i.e. a parity-check matrix for C.

□

This helps us to find a parity-check matrix if we already have a generator matrix. If the generator matrix is in standard form, then a parity-check matrix is particularly easy to find.

Lemma 5.7. *Suppose C is a linear code, and that*

$$G = (I_k|A)$$

is a generator matrix for C in standard form: I_k is the k by k identity matrix, and A is some k by $n-k$ matrix. Then the matrix

$$H = (-A^{\mathrm{T}}|I_{n-k})$$

is a parity-check matrix for C.

Proof. Certainly H is an $n-k$ by n matrix. The last $n-k$ columns of H are the standard basis vectors, and and so are linearly independent. So H has rank at least $n-k$, and hence the rows of H must be linearly independent. It is a simple exercise (which you should do!) to check that $GH^{\mathrm{T}} = 0$, and we can appeal to Lemma 5.6. □

Example.

- Recall the generator matrix
$$\begin{pmatrix}10110\\01101\end{pmatrix}$$
for the binary $(5,4,3)$-code discussed earlier. You should check that
$$\begin{pmatrix}11100\\10010\\01001\end{pmatrix}$$
is a parity-check matrix – remember that in $\mathbb{F}_2$, $+$ and $-$ are the same thing.
- The ternary 'parity-check' $[3,2]$-code
$$\{000, 012, 021, 102, 111, 120, 201, 210, 222\}$$
has generator matrix
$$\begin{pmatrix}102\\012\end{pmatrix}$$
and parity-check matrix
$$(111).$$

In view of Lemma 5.7, we say that a parity-check matrix is in standard form if it has the form

$$(B|I_{n-k}).$$

5.2 Syndrome decoding

Definition. Suppose C is a linear $[n,k]$-code over $\mathbb{F}_q$, and H is a parity-check matrix for C. Then for any word $w \in \mathbb{F}_q^n$, the *syndrome* of w is the vector

$$S(y) = yH^{\mathrm{T}} \in \mathbb{F}_q^{n-k}.$$

We saw above that $w \in (C^{\perp})^{\perp}$ if and only if $Hw^{\mathrm{T}} = 0$, i.e. if and only if $S(y) = 0$. So the syndrome of a word tells us whether it lies in our code. In fact, the syndrome tells us which coset of our code the word lies in.

Lemma 5.8. *Suppose C is a linear $[n,k]$-code, and v, w are words in $\mathbb{F}_q^n$. Then v and w lie in the same coset of C if and only if $S(v) = S(w)$.*

Proof.

$$\begin{aligned} v \text{ and } w \text{ lie in the same coset} &\Leftrightarrow v \in w + C \\ &\Leftrightarrow v = w + x, \text{ for some } x \in C \\ &\Leftrightarrow v - w \in C \\ &\Leftrightarrow H(v^{\mathrm{T}} - w^{\mathrm{T}}) = 0 \\ &\Leftrightarrow Hv^{\mathrm{T}} - Hw^{\mathrm{T}} = 0 \\ &\Leftrightarrow Hv^{\mathrm{T}} = Hw^{\mathrm{T}} \\ &\Leftrightarrow S(v) = S(w). \end{aligned}$$

□

In view of this lemma, we can talk about the *syndrome of a coset* to mean the syndrome of a word in that coset. Note that if C is an $[n,k]$-code, then a syndrome is a row vector of length $n-k$. So there are q^{n-k} possible syndromes. But we saw after Proposition 4.14 that there are also q^{n-k} different cosets of C, so in fact each possible syndrome must appear as the syndrome of a coset.

The point of this is that we can use syndromes to decode a linear code without having to write out a Slepian array. We construct a *syndrome look-up table* as follows.

1. Choose a parity-check matrix H for C.
2. Choose a set of coset leaders (i.e. one leader from each coset) and write them in a list.
3. For each coset leader, calculate its syndrome and write this next to it.

Example. Let $q = 3$, and consider the repetition code

$$C\{\mathtt{000}, \mathtt{111}, \mathtt{222}\}$$

again. This has generator matrix

$$G = (\mathtt{111}),$$

and hence parity-check matrix

$$H = \begin{pmatrix} \mathtt{102} \\ \mathtt{012} \end{pmatrix}.$$

There are 9 cosets of C, and a set of coset leaders, with their syndromes, is

leader	syndrome
000	00
001	22
002	11
010	01
020	02
100	10
200	20
012	12
021	21

.

Given a syndrome look-up table for a code C, we can construct a decoding process as follows.

- Given a word $w \in \mathbb{F}_q^n$, calculate the syndrome $S(w)$ of w.
- Find this syndrome in the syndrome look-up table; let v be the coset leader with this syndrome.
- Define $g(w) = w - v$.

Lemma 5.9. *The decoding process g that we obtain for C using a syndrome look-up table with coset leaders $l_1, \dots, l_r$ is the same as the decoding process f that we obtain using a Slepian array with coset leaders $l_1, \dots, l_r$.*

Proof. When we decode w using g, we find the coset leader with the same syndrome as w; by Lemma 5.8, this means the leader in the same coset as w. So

$$g(w) = w - (\text{chosen leader in the same coset as } w).$$

When we decode using f, we set

$$f(w) = (\text{codeword at the top of the same column as } w).$$

The construction of a Slepian array means that

$$w = (\text{leader at the left of the same row as } w) + (\text{codeword at the top of the same column as } w),$$

so

$$\begin{aligned} f(w) &= w - (\text{leader at the left of the same row as } w) \\ &= w - (\text{leader in the same coset as } w) \end{aligned}$$

(since the words in any one row of a Slepian array form a coset)

$$= g(w).$$

□

So we have seen the advantages of linear codes – although there might often be slightly larger non-linear codes with the same parameters, it requires a lot more work to describe them and to encode and decode.

Now we'll look at some specific examples of codes.

6 Some examples of linear codes

We've already seen some dull examples of codes. Here, we shall see some more interesting codes which are good with regard to certain bounds. The Hamming codes are perfect (i.e. give equality in the Hamming bound); in fact, they are almost the only known perfect codes. MDS codes are codes which give equality in the Singleton bound. We'll also see another bound – the Gilbert–Varshamov bound – which gives lower bounds for $A_q(n, d)$ in certain cases.

6.1 Hamming codes

We begin with binary Hamming codes, which are slightly easier to describe.

Definition. Let r be any positive integer, and let H_r be the r by $2^r - 1$ matrix whose columns are all the different non-zero vectors over $\mathbb{F}_2$. Define the *binary Hamming code* $\mathrm{Ham}(r, 2)$ to be the binary $[2^r - 1, 2^r - r - 1]$-code with H_r as its parity-check matrix.

Example.

- For $r = 1$, we have
$$H = (1),$$
so that $\mathrm{Ham}(1, 2)$ is the $[1, 0]$-code $\{\mathtt{0}\}$.

- For $r = 2$, we can choose
$$H = \begin{pmatrix}\mathtt{101}\\ \mathtt{011}\end{pmatrix},$$
and $\mathrm{Ham}(2, 2)$ is simply the repetition code $\{\mathtt{000}, \mathtt{111}\}$.

- $r = 3$ provides the first non-trivial example. We choose h to be in standard form:
$$H = \begin{pmatrix}\mathtt{1101100}\\ \mathtt{1011010}\\ \mathtt{0111001}\end{pmatrix}.$$
Then we can write down the generator matrix
$$G = \begin{pmatrix}\mathtt{1000110}\\ \mathtt{0100101}\\ \mathtt{0010011}\\ \mathtt{0001111}\end{pmatrix}.$$
Hence
$$\mathrm{Ham}(3, 2) = \{\mathtt{0000000}, \mathtt{1000110}, \mathtt{0100101}, \mathtt{1100011}, \mathtt{0010011}, \mathtt{1010101}, \mathtt{0110110}, \mathtt{1110000},$$
$$\mathtt{0001111}, \mathtt{1001001}, \mathtt{0101010}, \mathtt{1101100}, \mathtt{0011100}, \mathtt{1011010}, \mathtt{0111001}, \mathtt{1111111}\}.$$

Note that the Hamming code is not uniquely defined – it depends on the order you choose for the columns of H. But choosing different orders still gives equivalent codes, so we talk of *the* Hamming code $\mathrm{Ham}(r, 2)$.

Now we consider q-ary Hamming codes for an arbitrary prime power q. We impose a relation on the non-zero vectors in $\mathbb{F}_q^r$ by saying that $v \equiv w$ if and only if $v = \lambda w$ for some non-zero $\lambda \in \mathbb{F}_q$.

Lemma 6.1. $\equiv$ *is an equivalence relation.*

Proof. We need to check three things.

1. $v \equiv v$ for all $v \in \mathbb{F}_q^r$. This follows because $v = 1v$ and $1 \in \mathbb{F}_q \setminus \{\mathtt{0}\}$.
2. $(v \equiv w) \Rightarrow (w \equiv v)$ for all $v, w \in \mathbb{F}_q^r$. This follows because if $v = \lambda w$ for $\lambda \in \mathbb{F}_q \setminus \{\mathtt{0}\}$, then $w = \lambda^{-1} v$ with $\lambda^{-1} \in \mathbb{F}_q \setminus \{\mathtt{0}\}$.

3. $(v \equiv w \equiv x) \Rightarrow (v \equiv x)$ for all $v, w, x \in \mathbb{F}_q^r$. This follows because if $v = \lambda w$ and $w = \mu x$ for $\lambda, \mu \in \mathbb{F}_q \setminus \{\mathtt{0}\}$, then $v = (\lambda\mu)x$ with $\lambda\mu \in \mathbb{F}_q \setminus \{\mathtt{0}\}$.

$\square$

Now we count the equivalence classes of non-zero words (the zero word lies in an equivalence class on its own).

Lemma 6.2. *Under the equivalence relation* $\equiv$*, there are exactly* $\dfrac{q^r - 1}{q - 1}$ *equivalence classes of non-zero words.*

Proof. Take $v \in \mathbb{F}_q^r \setminus \{0\}$. The equivalence class containing v consists of all words λv for $\lambda \in \mathbb{F}_q \setminus \{\mathtt{0}\}$. There are $q - 1$ possible choices of λ, and these give distinct words: if $\lambda \neq \mu$, then (since $v \neq 0$) $(\lambda v \neq \mu v)$. So there are exactly $q-1$ words in each equivalence class. There are $q^r - 1$ non-zero words altogether, so the number of equivalence classes is $\dfrac{q^r - 1}{q - 1}$. $\square$

Now choose vectors $v_1, \dots, v_N$, where $N = (q^r - 1)/(q - 1)$, choosing exactly one from each equivalence class. Define H to be the r by N matrix with columns $v_1, \dots, v_N$, and define the *q-ary Hamming code* $\mathrm{Ham}(r, q)$ to the $[N, N - r]$-code over $\mathbb{F}_q$ with parity-check matrix H. Again, the actual code depends on the order of $v_1, \dots, v_N$ (and on the choice of $v_1, \dots, v_N$), but different choices give equivalent codes.

Example.

- Let $q = 5$ and $r = 2$. H may be chosen to equal

$$\begin{pmatrix} 111110 \\ 123401 \end{pmatrix},$$

so that $\mathrm{Ham}(2, 5)$ is the $[6, 4]$-code over $\mathbb{F}_5$ with generator matrix

$$\begin{pmatrix} 100044 \\ 010043 \\ 001042 \\ 000141 \end{pmatrix}.$$

- Let $q = 3$ and $r = 3$. H may be chosen to be

$$\begin{pmatrix} 0011111111100 \\ 1100111222010 \\ 1212012012001 \end{pmatrix},$$

and $\mathrm{Ham}(3, 3)$ is the $[13, 10]$-code with generator matrix

$$\begin{pmatrix} 1000000000022 \\ 0100000000021 \\ 0010000000202 \\ 0001000000201 \\ 0000100000220 \\ 0000010000222 \\ 0000001000221 \\ 0000000100210 \\ 0000000010212 \\ 0000000001211 \end{pmatrix}.$$

It may not be obvious how to choose the vectors $v_1, \dots, v_N$, but there is a trick: choose all non-zero vectors whose first non-zero entry is a 1. You might like to prove as an exercise that this always works, but you can use this trick without justification in the exam.

Lemma 6.3. Ham(r, q) *is an* $[N, N-r]$*-code over* $\mathbb{F}_q$*, where as above* $N = (q^r - 1)/(q-1)$.

Proof. H is an $r \times N$ matrix, and so Ham$(r, q)^\perp$ is an $[N, r]$-code. So Ham(r, q) is an $[N, N-r]$-code, by Theorem 5.3. □

Now we'll see the key property of Hamming codes.

Theorem 6.4. Ham(r, q) *has minimum distance at least* 3.

Proof. Since Ham(r, q) is linear, it's enough to show that the minimum weight of a non-zero codeword is 3, i.e. that there are no codewords of weight 1 or 2. We have a parity-check matrix H, and by Lemma 5.5 a word w lies in Ham(r, q) if and only if $Hw^{\mathrm{T}} = 0$. So all we need to do is show that $Hw^{\mathrm{T}} \neq 0$ whenever w is a word of weight 1 or 2.

Suppose that w has weight 1, with

$$\begin{aligned} w_i &= \lambda \neq \mathbb{0}, \\ w_j &= \mathbb{0} \text{ for } j \neq i. \end{aligned}$$

Recall that the columns of H are $v_1, \dots, v_N$. We calculate that $Hw^{\mathrm{T}} = \lambda v_i$. Now $v_i \neq 0$ by construction, and $\lambda \neq 0$, and so $Hw^{\mathrm{T}} \neq 0$, so $w \notin$ Ham(r, q).

Next suppose w has weight 2, with

$$\begin{aligned} w_i &= \lambda \neq \mathbb{0}, \\ w_j &= \mu \neq \mathbb{0}, \\ w_k &= \mathbb{0} \text{ for } k \neq i, j. \end{aligned}$$

Then we calculate that $Hw^{\mathrm{T}} = \lambda v_i + \mu v_j$. If this equals 0, then we have

$$v_i - \frac{\mu}{\lambda} v_j$$

(since $\lambda \neq \mathbb{0}$), and this implies that $v_i \equiv v_j$ (since $\mu \neq \mathbb{0}$). But we chose $v_1, \dots, v_N$ to be from different equivalence classes; contradiction. So $Hw^{\mathrm{T}} \neq 0$, and $w \notin$ Ham(r, q). □

Theorem 6.5. Ham(r, q) *is a perfect* 1*-error-correcting code.*

Proof. Ham(r, q) is 1-error-correcting since its minimum distance is greater than 2. To show that it is perfect, we have to show that equality holds in the Hamming bound, i.e.

$$|\,\mathrm{Ham}(r, q)| = \frac{q^N}{\binom{N}{0} + (q-1)\binom{N}{1}},$$

where $N = \dfrac{q^r - 1}{q - 1}$.

Since $\mathrm{Ham}(r,q)$ is an $[N,N-r]$-code, the left-hand side equals q^{N-r} by Lemma 4.5. The right-hand side equals

$$\begin{aligned}\frac{q^N}{1+(q-1)N} &= \frac{q^N}{1+(q-1)\frac{q^r-1}{q-1}}\\ &= \frac{q^N}{1+q^r-1}\\ &= q^{N-r},\end{aligned}$$

and the equation follows. □

For binary Hamming codes, there is quite a neat way to do syndrome decoding. First, we need to know what the coset leaders look like in a Hamming code.

Lemma 6.6. *Suppose $C = \mathrm{Ham}(r,q)$, and that D is a coset of C. Then D contains a unique word of weight at most* 1.

Proof. First we'll show that the number of words in $\mathbb{F}_q^r$ equals the number of cosets, so that there is one word of weight at most 1 per coset on average. Then we'll show that any coset contains at most one word of weight 1. This will then imply that each coset contains exactly one word of weight 1.

C is an $[N,N-r]$-code, so there are q^r cosets of C (see the discussion after Proposition 4.14). Now we look at the number of words of weight at most 1. There is one word of weight 0. To specify a word of weight 1, we need to choose what the non-zero entry in the word will be ($q-1$ choices), and where it will occur (N choices). So the number of words of weight at most 1 is

$$1+(q-1)N = 1+(q-1)\frac{q^r-1}{q-1} = q^r.$$

For the second part, suppose v and w are weight at most 1 lying in the same coset. Then $v \in w+C$, so $v = w+x$ for some $x \in C$, i.e. $v-w \in C$. Now v and $-w$ are words of weight at most 1, and so $v-w$ has weight at most 2. But $d(C) \geqslant 3$, so the only word in C of weight at most 2 is the zero word. So $v-w = 0$, i.e. $v = w$, and so a coset contains at most one word of weight at most 1. □

The preceding lemma tells us that the coset leaders for a Hamming code must be precisely all the words of weight at most 1. Now we restrict our attention to the case $q = 2$. Recall that the columns of H_r are precisely all the different non-zero column vectors over $\mathbb{F}_2$ – these give the binary representations of the numbers $1, 2, \ldots, 2^r - 1$. We order the columns of H_r so that column i gives the binary representation of the number i.

Example. Suppose $r = 3$. Then we choose

$$\begin{pmatrix}0001111\\0110011\\1010101\end{pmatrix}.$$

Now suppose $w \in \mathbb{F}_2^N$. Since $\mathrm{Ham}(r,2)$ is perfect 1-error-correcting, w is either a codeword or is distance 1 away from a unique codeword. So either $w \in \mathrm{Ham}(r,2)$ or $w - e_j \in \mathrm{Ham}(r,2)$ for some (unique) j, where e_j is the word which has a 1 in position j and 0s elsewhere.

If $w \notin \mathrm{Ham}(r,2)$, we want to be able to work out what j is.

Lemma 6.7. *Let $S(w)$ be the syndrome of w. If $w \in C$, then $S(w) = 0$. Otherwise, $S(w)$ gives the binary representation of j.*

Proof. Since $w - e_j$ lies in the code, e_j must lie in the same coset as w. So e_j has the same syndrome as w. But clearly the syndrome of e_j is the jth row of H^T, and we picked H so that the jth row of H^T is the binary representation of j. □

Example. Continuing the last example: suppose the codeword 0101010 is transmitted (exercise: check that this is a codeword, given the way we've chosen H_r). Suppose that the fifth digit gets distorted to a 1, so we receive 0101110. We calculate the syndrome

$$\mathtt{0101110}\begin{pmatrix}\mathtt{001}\\\mathtt{010}\\\mathtt{011}\\\mathtt{100}\\\mathtt{101}\\\mathtt{110}\\\mathtt{111}\end{pmatrix} = (\mathtt{101}),$$

which is the binary representation of the number 5. So we know to change the fifth digit to recover the codeword.

6.2 Existence of codes and linear independence

The proof that the Hamming codes have distance 3 relied on the following fact about their parity-check matrices: if we take a set consisting of at most 2 columns of H, then this set is linearly independent. This leads to a more general theorem on the existence of linear codes.

Theorem 6.8. *Suppose C is an $[n, k]$-code with parity-check matrix H. Then C has minimum distance at least d if and only if any $d - 1$ columns of H are linearly independent.*

In particular, an $[n, k, d]$-code exists if and only if there is a sequence of n vectors in $\mathbb{F}_q^{n-k}$ such that any $d - 1$ of them are linearly independent.

Proof. Let $c_1, \dots, c_n$ be the columns of H, and suppose first that there are columns $c_{i_1}, \dots, c_{i_{d-1}}$ which are linearly dependent, i.e. there are scalars $\lambda_1, \dots, \lambda_{d-1}$ (not all zero) such that

$$\lambda_1 c_{i_1} + \cdots + \lambda_{d-1} c_{i_{d-1}} = 0.$$

Let w be the word which has $\lambda_1, \dots, \lambda_{d-1}$ in positions $i_1, \dots, i_{d-1}$, and 0s elsewhere. Then the above equation is the same as saying $Hw^\mathrm{T} = 0$, which, since H is a parity-check matrix for C, is the same as saying $w \in C$. But then w is a non-zero word of weight at most $d - 1$, while C is a code of minimum distance at least d; contradiction.

The other direction is basically the same: if C does not have minimum distance at least d, then C has a non-zero codeword w of weight $e < d$, and the equation $Hw^\mathrm{T} = 0$ provides a linear dependence between some e of the columns of H; if e vectors are linearly dependent, then any $d - 1$ vectors including these e are certainly linearly dependent.

The second paragraph of the proposition is now immediate – if we have such a code, then the columns of a parity-check matrix are such a set of vectors, while if we have such a set of vectors, then the matrix with these vectors as its columns is the parity-check matrix of such a code. □

Notice that we say 'sequence' rather than 'set', since two columns of a matrix might be the same. However, if there are two equal columns, then they are linearly dependent, and so the minimum distance of our code would be at most 2.

Theorem 6.8 tells us that one way to look for good linear codes is to try to find large sets of vectors such that large subsets of these are linearly independent. This is often referred to as the 'main linear coding theory problem'. We use this now to prove the Gilbert–Varshamov bound. The bounds we saw earlier – the Hamming, Singleton and Plotkin bounds – were all essentially of the form 'if a code with these parameters exists, then the following inequality holds', and and so gave upper bounds on $A_q(n, d)$. The Gilbert–Varshamov bound says 'if this inequality holds, then a code with these parameters exists', and so it gives some lower bounds on $A_q(n, d)$.

First we prove a very simple lemma.

Lemma 6.9. *For any $n, i > 0$,*

$$\binom{n}{i} \geqslant \binom{n-1}{i}.$$

Proof. This follows from the equation

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}.$$

This is proved either by writing the binomial coefficients in terms of factorials or by considering $\binom{n}{i}$ as the number of ways of choosing a subset of size i from the set $\{1, \dots, n\}$. Each such subset either contains the number n or it doesn't; if it doesn't, then the set is actually a subset of $\{1, \dots, n-1\}$ of size i, which may be chosen in $\binom{n-1}{i}$ ways. If it does, then the remainder of the subset is a subset of $\{1, \dots, n-1\}$ of size $i-1$, which may be chosen in $\binom{n-1}{i-1}$ ways. □

Theorem 6.10 (Gilbert–Varshamov bound)**.** *Suppose q is a prime power, and n, r, d are positive integers satisfying*

$$\binom{n-1}{0} + (q-1)\binom{n-1}{1} + (q-1)^2\binom{n-1}{2} + \cdots + (q-1)^{d-2}\binom{n-1}{d-2} < q^r. \tag{$*$}$$

Then an $[n, n-r, d]$-code over $\mathbb{F}_q$ exists.

Proof. By Theorem 6.8, all we need to do is find a sequence of n vectors in $\mathbb{F}_q^r$ such that any $d-1$ of them are linearly independent. In fact, we can do this in a completely naïve way. We begin by choosing any non-zero vector $v_1 \in \mathbb{F}_q^r$. Then we choose any vector v_2 such that v_1 and v_2 are linearly independent. Then we choose any v_3 such that any $d-1$ of v_1, v_2, v_3 are linearly independent, and so on. We need to show that this always works, i.e. at each stage you can choose an appropriate vector. Formally, this amounts to the following.

We prove the Gilbert–Varshamov bound by induction on n. For the case $n = 1$, we just need to be able to find a non-zero vector v_1; we can do this, since $r > 0$.

Now suppose $n > 1$ and that the theorem is true with n replaced by $n-1$, i.e. whenever

$$\binom{n-2}{0} + (q-1)\binom{n-2}{1} + (q-1)^2\binom{n-2}{2} + \cdots + (q-1)^{d-2}\binom{n-2}{d-2} < q^r, \tag{$\dagger$}$$

we can find a sequence $v_1, \dots, v_{n-1}$ of vectors in $\mathbb{F}_q^r$ such that any $d-1$ of them are linearly independent.

Assume that the inequality $(*)$ holds. Recall that for any i we have

$$\binom{n-1}{i} = \binom{n-2}{i} + \binom{n-2}{i-1};$$

this implies that

$$\binom{n-1}{i} \geqslant \binom{n-2}{i},$$

and so

$$\begin{aligned} q^r &> \binom{n-1}{0} + (q-1)\binom{n-1}{1} + (q-1)^2\binom{n-1}{2} + \dots + (q-1)^{d-2}\binom{n-1}{d-2} \\ &\geqslant \binom{n-2}{0} + (q-1)\binom{n-2}{1} + (q-1)^2\binom{n-2}{2} + \dots + (q-1)^{d-2}\binom{n-2}{d-2}. \end{aligned}$$

Hence if $(*)$ holds then so does $(\dagger)$. So by our inductive hypothesis we can find a sequence $v_1, \dots, v_{n-1}$ of vectors of which any $d-1$ are linearly independent.

Given a vector $v_n \in \mathbb{F}_q^r$, we say that it is *good* if any $d-1$ of the vectors $v_1, \dots, v_n$ are linearly independent, and *bad* otherwise. All we need to do is show that there is a good vector. We'll do this by counting the bad vectors, and showing that the number of bad vectors v_n is strictly less than the total number of choices of $v_n \in \mathbb{F}_q^r$, i.e. q^r; then we'll know that there is a good vector.

Suppose v_n is bad. This means that some $d-1$ of the vectors $v_1, \dots, v_n$ are linearly dependent, i.e. there exist $1 \leqslant i_1 < \dots < i_{d-1} \leqslant n$ and $\lambda_1, \dots, \lambda_{d-1} \in \mathbb{F}_q$ not all zero such that

$$\lambda_1 v_{i_1} + \dots + \lambda_{d-1} v_{i_{d-1}} = 0.$$

By our assumption, any $d-1$ of the vectors $v_1, \dots, v_{n-1}$ are linearly independent, so the above sum must involve v_n with non-zero coefficient, i.e. $i_{d-1} = n$ and $\lambda_{d-1} \neq 0$. So we have

$$v_n = -\left(\frac{\lambda_1}{\lambda_{d-1}}\right) v_{i_1} - \left(\frac{\lambda_2}{\lambda_{d-1}}\right) v_{i_2} - \dots - \left(\frac{\lambda_{d-2}}{\lambda_{d-1}}\right) v_{i_{d-2}}.$$

By discarding any values j for which $\lambda_j = 0$ and re-labelling, we can write

$$v_n = \mu_1 v_{i_1} + \dots + \mu_e v_{i_e}$$

for some $0 \leqslant e \leqslant d-2$, some $1 \leqslant i_1 < \dots < i_{d-2} \leqslant n-1$ and some *non-zero* $\mu_i \in \mathbb{F}_q$. (n.b. the case $e = 0$ is allowed – it gives v_n equals to the empty sum, i.e. $v_n = 0$.)

So every bad vector can be written as a linear combination with non-zero coefficients of e of the vectors $v_1, \dots, v_{n-1}$, for some $e \leqslant d-2$. So the number of bad vectors is at most the number of such linear combinations. (Note that we say 'at most' because it might be that a bad vector can be written in several different ways as a linear combination like this.)

How many of these linear combinations are there? For a given e, we can choose the numbers $i_1, \dots, i_e$ in $\binom{n-1}{e}$ different ways. Then we can choose each of the coefficients μ_i in $q-1$ different ways (since μ_i must be chosen non-zero). So for each e the number of different linear combinations

$$\mu_i v_{i_1} + \dots + \mu_e v_{i_e}$$

is $(q-1)^e\binom{n-1}{e}$. We sum over e to obtain

$$\begin{aligned}(\text{number of bad vectors}) &\leqslant \binom{n-1}{0} + (q-1)\binom{n-1}{1} + (q-1)^2\binom{n-1}{2} + \cdots + (q-1)^{d-2}\binom{n-1}{d-2} \\ &< q^r\end{aligned}$$

by $(*)$. So not every vector is bad, and so we can find a good vector. □

6.3 MDS codes

We begin by recalling the Singleton bound, and applying it to linear codes.

Theorem 6.11 (Singleton bound for linear codes)**.** *An* $[n, k, d]$*-code over* $\mathbb{F}_q$ *satisfies*

$$d \leqslant n - k + 1.$$

Proof. If C is an $[n, k, d]$-code over $\mathbb{F}_q$, then C is a q-ary (n, M, d)-code, where $M = q^k$ by Lemma 4.5. Hence by Theorem 2.8(2) we have

$$q^k \leqslant q^{n+1-d},$$

i.e. $k \leqslant n + 1 - d$. □

The aim of this section is to look at codes which give equality in this bound. In an $[n, k]$-code, the number $n - k$ is called the *redundancy* of the code – it can be thought of as the number of redundant digits we add to our source word to make a codeword.

Definition. A *maximum distance separable code* (or MDS code) of length n and redundancy r is a linear $[n, n - r, r + 1]$-code.

Obviously, the main question concerning MDS codes is: for which n, r does an MDS code of length n and redundancy r exist? The answer is not known in general, but we shall prove some results and construct some MDS codes. Theorem 6.8 says that an MDS code of length n and redundancy r exists if and only if we can find a sequence of n vectors in $\mathbb{F}_q^r$ of which any r are linearly independent. Clearly if we can do this for a given value of n, then we can do it for any smaller value of n, just by deleting some of the vectors. This implies that for each r (and q) there is a maximum n (possibly infinite) for which an MDS code of length n and redundancy r exists. Given r, q, we write $\max(r, q)$ for this maximum. The main question about MDS codes is to find the values of $\max(r, q)$.

Note that $\max(r, q)$ may be infinite, even though there are only finitely many vectors in $\mathbb{F}_q^n$, because we are allowed repeats in our sequence of vectors. Note, though, that if we have a repeated vector in our sequence, then the code cannot possibly have distance more than 2. Here is our main theorem on the values $\max(r, q)$.

Theorem 6.12.

1. *If* $r = 0$ *or* 1*, then* $\max(r, q) = \infty$.
2. *If* $r > q$*, then* $\max(r, q) = r + 1$.
3. *If* $2 \leqslant r \leqslant q$*, then* $\max(r, q) \geqslant q + 1$.

It is conjectured that the bounds in (3) actually give the right values of $\max(r, q)$, except in the cases where q is even and $r = 3$ or $q - 1$, in which case MDS codes of length $q + 2$ can be constructed.

The three parts of Theorem 6.12 have different proofs, and the first two are quite easy.

Proof of Theorem 6.12(1). We must show that for $r = 0$ or 1 and for any n and q, there exists an MDS code of length n and redundancy r. For $r = 0$, this means we need an $[n, n, 1]$-code. But the whole of $\mathbb{F}_q^n$ is such a code, as we saw in Theorem 2.1.

For $r = 1$, we want to construct a sequence of n vectors in $\mathbb{F}_q^1$ such that the set formed by any one of them is linearly independent. We just take each of the vectors to be the vector (1). □

Proof of Theorem 6.12(2). To show that $\max(r, q) \geqslant r + 1$, we need to show that an MDS code of length $r + 1$ and redundancy r exists. But this is an $[r + 1, 1, r + 1]$-code, and the repetition code is such a code.

To show that $\max(r, q) \leqslant r + 1$, we have to show that we can't find an MDS code of length $r + 2$ and redundancy r, i.e. an $[r + 2, 2, r + 1]$-code. If we can find such a code C, then any code $\mathcal{D}$ equivalent to C is also an $[r + 2, 2, r + 1]$-code. So by taking a generator matrix for C and applying matrix operations MO1–5, we may find an $[r + 2, 2, r + 1]$-code with a generator matrix in standard form. This has a parity-check matrix H in standard form, i.e. of the form $(B|I_r)$, where B is some $r \times 2$ matrix over $\mathbb{F}_q$.

Let v, w be the columns of B. By Theorem 6.8 any r of the columns of H are linearly independent, i.e. any r of the vectors $v, w, e_1, \dots, e_r$ are linearly independent, where $e_1, \dots, e_r$ are the columns of the identity matrix, i.e. the standard basis vectors.

Suppose the entries of v are $v_1, \dots, v_r$. First we show that $v_1, \dots, v_r$ are all non-zero. If not, then $v_j = \mathbb{0}$ for some j. Then we have

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{j-1} \\ \mathbb{0} \\ v_{j+1} \\ \vdots \\ v_r \end{pmatrix} = v_1 e_1 + v_2 e_2 + \dots + v_{j-1} e_{j-1} + v_{j+1} e_{j+1} + \dots + v_r e_r,$$

and so the r vectors $v, e_1, \dots, e_{j-1}, e_{j+1}, \dots, e_r$ are linearly dependent. Contradiction. So each v_j is non-zero. Similarly we find that the entries $w_1, \dots, w_r$ of w are non-zero. This means that $\frac{v_1}{w_1}, \dots, \frac{v_r}{w_r}$ are non-zero elements of $\mathbb{F}_q$. Now there are only $q - 1 < r$ distinct non-zero elements of $\mathbb{F}_q$, and so we must have $\frac{v_i}{w_i} = \frac{v_j}{w_j}$ for some $i < j$. We re-write this as $\frac{v_j}{v_i} - \frac{w_j}{w_i} = \mathbb{0}$. Now consider the vector $\frac{v}{v_i} - \frac{w}{w_i}$. The kth component of this vector equals $\frac{v_k}{v_i} - \frac{w_k}{w_i}$, and so we have

$$\frac{v}{v_i} - \frac{w}{w_i} = \sum_{k=1}^{r} \left(\frac{v_k}{v_i} - \frac{w_k}{w_i} \right) e_k.$$

Now for $k = i$ and $k = j$ we have $\frac{v_k}{v_i} - \frac{w_k}{w_i} = \mathbb{0}$, and so we may ignore the i and j terms to get

$$\frac{v}{v_i} - \frac{w}{w_i} = \sum_{k=1}^{i-1} \left(\frac{v_k}{v_i} - \frac{w_k}{w_i} \right) e_k + \sum_{k=i+1}^{j-1} \left(\frac{v_k}{v_i} - \frac{w_k}{w_i} \right) e_k + \sum_{k=j+1}^{r} \left(\frac{v_k}{v_i} - \frac{w_k}{w_i} \right) e_k.$$

So the r vectors $v, w, e_1, \dots, e_{i-1}, e_{i+1}, \dots, e_{j-1}, e_{j+1}, \dots, e_r$ are linearly dependent; contradiction. □

For the proof of Theorem 6.12(3), we can be a bit more clever. Given a sequence of vectors $v_1, \dots, v_n$, we have an easy way to check whether some r of them are linearly independent. We let A be the matrix which has these vectors as its columns. Then A is a square matrix, and so has a determinant. And the columns of A are linearly independent if and only if the determinant is non-zero. We need to look at a particular type of determinant, called a 'Vandermonde determinant'.

Proposition 6.13. *Suppose $x_1, \dots, x_r$ are distinct elements of a field $\mathbb{F}$. Then the determinant*

$$\begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_r \\ x_1^2 & x_2^2 & \dots & x_r^2 \\ \vdots & \vdots & & \vdots \\ x_1^{r-1} & x_2^{r-1} & \dots & x_r^{r-1} \end{vmatrix}$$

is non-zero.

Now we can construct our codes.

Proof of Theorem 6.12(3). We need to construct a $[q+1, q+1-r, r+1]$-code. Label the elements of $\mathbb{F}_q$ as $\lambda_1, \dots, \lambda_q$ in some order, and let

$$H = \begin{pmatrix} 1 & 1 & \dots & 1 & 0 \\ \lambda_1 & \lambda_2 & \dots & \lambda_q & 0 \\ \lambda_1^2 & \lambda_2^2 & \dots & \lambda_q^2 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ \lambda_1^{r-2} & \lambda_2^{r-2} & \dots & \lambda_q^{r-2} & 0 \\ \lambda_1^{r-1} & \lambda_2^{r-1} & \dots & \lambda_q^{r-1} & 1 \end{pmatrix}.$$

Let C be the code with H as its parity-check matrix. Then C is a $[q+1, q+1-r]$, code, and we claim that C has minimum distance at least $r+1$. Recall from Theorem 6.8 that this happens if and only if any r columns of H are linearly independent. H has r rows, and so any r columns together will form a *square* matrix, and we can check whether the columns are linearly independent by evaluating the determinant. So choose r columns of H, and let J be the matrix formed by them.

If the last column of H is not one of the columns chosen, then

$$J = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_r \\ x_1^2 & x_2^2 & \dots & x_r^2 \\ \vdots & \vdots & & \vdots \\ x_1^{r-1} & x_2^{r-1} & \dots & x_r^{r-1} \end{pmatrix}$$

for some distinct $x_1, \dots, x_r$, and so $\det(J) \neq 0$ by Proposition 6.13. If the last column of H is one of the columns chosen, then we have

$$J = \begin{pmatrix} 1 & 1 & \dots & 1 & 0 \\ x_1 & x_2 & \dots & x_{r-1} & 0 \\ x_1^2 & x_2^2 & \dots & x_{r-1}^2 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{r-2} & x_2^{r-2} & \dots & x_{r-1}^{r-2} & 0 \\ x_1^{r-1} & x_2^{r-1} & \dots & x_{r-1}^{r-1} & 1 \end{pmatrix}$$

for some distinct $x_1, \dots, x_{r-1}$. Let J' be the matrix formed by the first $r-1$ rows and the first $r-1$ columns. Then $\det(J') \neq 0$ by Proposition 6.13, and so the first $r-1$ rows of J are linearly independent. Now consider the rows of J; suppose that

$$\mu_1.(\text{row } 1) + \cdots + \mu_{r-1}.(\text{row } r-1) + \mu_r.(\text{row } r) = 0.$$

Looking at the last entry of each row, we see that

$$0 + \cdots + 0 + \mu_r = 0.$$

Hence

$$\mu_1.(\text{row } 1) + \cdots + \mu_{r-1}.(\text{row } r-1) = 0,$$

but this implies $\mu_1 = \cdots = \mu_{r-1} = 0$, since the first $r-1$ rows of J are linearly independent. And so all the rows of J are linearly independent, so $\det J \neq 0$, as required. □

6.4 Reed–Muller codes

The Reed–Muller codes are binary codes, and we need to define a new operation on binary words. If $v = v_1 \dots v_n$ and $w = w_1 \dots w_n$ are words in $\mathbb{F}_2^n$, then we define the product $v * w$ to be the word $(v_1 w_1) \dots (v_n w_n)$. Note that $v.w$ is the weight of $v * w$ modulo 2.

Now suppose n is a positive integer and $0 \leqslant i < n$. Let $x_i(n)$ be the word of length 2^n which consists of chunks of 0s and 1s alternately, the chunks being of length 2^i.

Example. We have

$$\begin{aligned} x_0(2) &= \mathtt{0101}, \\ x_1(2) &= \mathtt{0011}, \\ x_0(3) &= \mathtt{01010101}, \\ x_1(3) &= \mathtt{00110011}, \\ x_2(3) &= \mathtt{00001111}. \end{aligned}$$

We will write $x_i(n)$ as x_i when it is clear what the value of n is. We consider products of the words $x_i(n)$. For example,

$$\begin{aligned} x_0(2)x_1(2) &= \mathtt{0001}, \\ x_0(3)x_2(3) &= \mathtt{00000101}. \end{aligned}$$

We include the word $11\dots1$, which we regard as the 'empty product', and write as $1(n)$. Note that we only bother with products of *distinct* $x_i(n)$s, since $x_i(n)x_i(n) = x_i(n)$.

Definition. The rth-order Reed–Muller code $\mathcal{R}(r, n)$ is the binary linear code of length 2^n spanned by all products of at most r of the words $x_0(n), \dots, x_{n-1}(n)$.

Note that in 'at most r' we include 0, so we include the product of none of the words $x_0(n), \dots, x_{n-1}(n)$, i.e. the word $1(n)$.

Example. Take $n = 3$. Then the products of the words x_0, x_1, x_2 are as follows:

$$\begin{aligned}
1 &= \texttt{11111111},\\
x_0 &= \texttt{01010101},\\
x_1 &= \texttt{00110011},\\
x_0 * x_1 &= \texttt{00010001},\\
x_2 &= \texttt{00001111},\\
x_0 * x_2 &= \texttt{00000101},\\
x_1 * x_2 &= \texttt{00000011},\\
x_0 * x_1 * x_2 &= \texttt{00000001}.
\end{aligned}$$

So

$$\begin{aligned}
&\mathcal{R}(0,3) = \langle \texttt{11111111} \rangle,\\
&\mathcal{R}(1,3) = \langle \texttt{11111111}, \texttt{01010101}, \texttt{00110011}, \texttt{00001111} \rangle,
\end{aligned}$$

and

$$\mathcal{R}(2,3) = \langle \texttt{11111111}, \texttt{01010101}, \texttt{00110011}, \texttt{00001111}, \texttt{00010001}, \texttt{00000101}, \texttt{00000011} \rangle.$$

We want to work out the dimension of $\mathcal{R}(r, n)$. In fact, the spanning set we've chosen is a basis, but this is not immediately obvious. Let's have a look at the size of this spanning set: for each $0 \leqslant i \leqslant r$, we take all products of i of the words $x_0, \dots, x_{n-1}$. The number of ways of choosing these i words is $\binom{n}{i}$. By summing for all i, we find that

$$\dim \mathcal{R}(r,n) \leqslant \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{r}.$$

We're going to prove that in fact equality holds above.

The next lemma is much less complicated than it looks.

Lemma 6.14. *Suppose* $0 \leqslant i_1 < \cdots < i_s < n$ *and let* $x = x_{i_1}(n) * x_{i_2}(n) * \cdots * x_{i_s}(n)$. *If* $i_s < n-1$, *then the* x *is simply the word* $x_{i_1}(n-1) * x_{i_2}(n-1) * \cdots * x_{i_s}(n-1)$ *written twice. If* $i_s = n-1$, *then* x *is the word* $\texttt{00}\dots\texttt{0}$ *of length* 2^{n-1} *followed by the word* $x_{i_1}(n-1) * x_{i_2}(n-1) * \cdots * x_{i_{s-1}}(n-1)$.

Proof. If $i < n-1$, then $x_i(n)$ is the word $x_i(n-1)$ written twice. So if we take any product of words $x_i(n)$ with $i < n-1$, then we'll get the product of the corresponding $x_i(n-1)$s written twice. $x_{n-1}(n)$ consists of 2^{n-1} zeroes followed by 2^{n-1} ones. So if v is a word of length 2^n consisting of a word w written twice, then $v * x_{n-1}(n)$ consists of 2^{n-1} zeroes followed by w. The lemma follows. □

Proposition 6.15. *If* $0 \leqslant i_1 < \cdots < i_s < n$, *then in the word*

$$x_{i_1}(n) * x_{i_2}(n) * \cdots * x_{i_s}(n)$$

the first 1 *appears in position* $1 + 2^{i_1} + 2^{i_2} + \cdots + 2^{i_s}$.

Proof. We prove this by induction on n, with the case $n = 1$ being easy to check. So we suppose that $n > 1$ and the result holds with n replaced by $n - 1$. Let $x = x_{i_1}(n) * x_{i_2}(n) * \cdots * x_{i_s}(n)$. There are two cases to consider, according to whether $i_s = n - 1$ or not. If $i_s < n - 1$, then by Lemma 6.14 x consists of the word $x_{i_1}(n-1) * \cdots * x_{i_s}(n-1)$ written twice, so the first 1 appears in position $1 + 2^{i_1} + \cdots + 2^{i_s}$, by induction. If $i_s = n - 1$, then by Lemma 6.14 x consists of the word consisting of 2^{n-1} zeroes followed by the word $x_{i_1}(n-1) * \cdots * x_{i_{s-1}}(n-1)$. So by induction the first 1 appears in position $2^{n-1} + p$, where p is the position of the first 1 in $x_{i_1}(n-1) * \cdots * x_{i_{s-1}}(n-1)$. By induction $p = 1 + 2^{i_1} + \cdots + 2^{i_{s-1}}$, and the result follows. □

There are 2^n different products of the words $x_0(n), \dots, x_{n-1}(n)$ (since each $x_i(n)$ is either involved in the product or not involved in it). Recall that for a positive integer p there is a unique way to write $p - 1$ as a sum of distinct powers of 2, i.e. there are unique integers $i_1 < i_2 < \cdots < i_s$ such that

$$p - 1 = 2^{i_1} + \cdots + 2^{i_s}.$$

Hence there is a unique way to write

$$p = 1 + 2^{i_1} + \cdots + 2^{i_s}.$$

Combined with Proposition 6.15, this means that for each p there is exactly one word which is a product of words $x_i(n)$ and which has its first 1 in position p. We label the products of the words $x_0(n), \dots, x_{n-1}(n)$ as $y_1, \dots, y_{2^n}$ so that for each y_p has its first 1 in position p, for each p.

Corollary 6.16. *The words $y_1, \dots, y_{2^n}$ are linearly independent, and*

$$\dim \mathcal{R}(r, n) = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{r}.$$

Proof. Suppose

$$\lambda_1 y_1 + \cdots + \lambda_{2^n} y_{2^n} = 0,$$

with each λ_i equal to $\mathtt{0}$ or $\mathtt{1}$ and not all the λ_i being $\mathtt{0}$. Let p be minimal such that $\lambda_i = \mathtt{1}$. The word y_p has a $\mathtt{1}$ in position p, while the words $y_{p+1}, \dots, y_{2^n}$ each have a $\mathtt{0}$ in position p. Hence the word

$$\lambda_1 w_1 + \cdots + \lambda_{2^n} w_{2^n}$$

has a $\mathtt{1}$ in position p; contradiction.

Now for the second part. As we have seen, there are

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{r}$$

words which are products of at most r of the words $x_0(n), \dots, x_{n-1}(n)$. These span $\mathcal{R}(r, n)$ by definition, and by Proposition 6.15 they are linearly independent, and so they form a basis for $\mathcal{R}(r, n)$. □

Now we want to find the minimum distance of $\mathcal{R}(r, n)$.

Theorem 6.17. *$\mathcal{R}(r, n)$ has minimum distance 2^{n-r}.*

Proof. ***(Non-examinable)*** We use the fact that the minimum distance of a linear code equals the minimum weight of a non-zero codeword. First we want to show that there is a codeword of weight 2^{n-r}. We claim that the word

$$x_{n-r}(n) * x_{n-r+1}(n) * \cdots * x_{n-1}(n)$$

consists of $2^n - 2^{n-r}$ zeroes followed by 2^{n-r} ones. We prove this by induction on n, with the case $n = 1$ easy to check. Suppose $n > 1$. By Lemma 6.14 the word $x_{n-r}(n) * x_{n-r+1}(n) * \cdots * x_{n-1}(n)$ equals the word $\mathbb{0}\mathbb{0}\ldots\mathbb{0}$ of length 2^{n-1} followed by the word

$$x_{n-r}(n-1) * x_{n-r+1}(n-1) * \cdots * x_{n-2}(n-1).$$

By induction this word consists of $2^{n-1} - 2^{n-r}$ zeroes followed by 2^{n-r} ones, and the claim is proved. Hence $\mathcal{R}(r, n)$ contains a word of weight 2^{n-r}, so $d(\mathcal{R}(r, n)) \leqslant 2^{n-r}$.

Now we show that every non-zero word has weight at least 2^{n-r}. Again, we proceed by induction on n, with the case $n = 1$ being easy to check. So we suppose that $n > 1$ and that the theorem is true for $n - 1$. Suppose w is a non-zero word in $\mathcal{R}(r, n)$; then we can write

$$w = y_1 + \cdots + y_s,$$

where each y_i is a product of at most r of the words $x_0(n), \ldots, x_{n-1}(n)$. We let u be the sum of all the terms which do not include $x_{n-1}(n)$, and we let v be the sum of all the terms which do include $x_{n-1}(n)$. Then $w = u + v$, and by Lemma 6.14 we see that

- u consists of a word $u' \in \mathcal{R}(r, n-1)$ written twice, and
- v consists of 2^{n-1} zeroes followed by a word $v' \in \mathcal{R}(r-1, n-1)$.

Now we can prove that the weight of w is at least 2^{n-r}, by considering several cases.

- Suppose $v' = 0$. Then $u' \neq 0$, and w consists of u written twice, so $\text{weight}(w) = 2\,\text{weight}(u')$. By induction, the weight of u' is at least $2^{(n-1)-r}$, and so the weight of w is at least $2.2^{n-1-r} = 2^{n-r}$.
- Now suppose $u' = v'$. Then $u' \neq 0$, and w consists of the word u' followed by 2^{n-1} zeroes. So $\text{weight}(w) = \text{weight}(u')$. Now $u' = v' \in \mathcal{R}(r-1, n-1)$, and so by induction the weight of u' is at least $2^{(n-1)-(r-1)} = 2^{n-r}$.
- Finally suppose $u' \neq v' \neq 0$. Then w consists of the word u' followed by the word $u' + v'$, so $\text{weight}(w) = \text{weight}(u') + \text{weight}(u' + v')$. u' and $u' + v'$ are both non-zero words in $\mathcal{R}(r, n-1)$, and so by induction $\text{weight}(u')$ and $\text{weight}(u' + v')$ are both at least 2^{n-1-r}. Hence $\text{weight}(w) \geqslant 2^{n-1-r} + 2^{n-1-r} = 2^{n-r}$.

$\square$