Topics in Discrete Mathematics: Error-Correcting Codes.

Dan Fretwell

Spring semester 2017/18

1 Introduction

Information is a valuable asset, we use it every day. However the transfer of information from one source to another is a very delicate process. Why is this? Well as humans we are typically prone to error! It is very simple for us to spell something wrong or to use a wrong word, leading to confusion. Of course we can't take the blame for **all** errors, some occur naturally during transmission of a message.

Some errors can be **detected** and some can't. If you receive a message saying

Let's have a cheeky aNndos at 6pm,

then you would know an error has been made, however you wouldn't know how many. You can detect the error in the word "aNndos" but you cannot detect whether there is an error in say the number "6" (changing to "7" would give an equally valid message). In order to **correct** errors you would have to assume that at most 1 error has been made.

A clever idea is to use mathematics to detect and correct errors. Imagine you have a finite set X of words/symbols/commands that you wish to transmit. If C is another finite set (of **codewords**) then any function $e : X \to C$ can be considered as **encoding** the elements of X, i.e. a way of translating words. Of course the whole point is to also be able to **decode** the elements of C that we receive and so we should demand that e is injective, so that there is a decoding function $d : C' \to X$, where $C' = \mathrm{im}(e)$ satisfying $d \circ e = \mathrm{id}$.

Silly Example

You win the lottery and call to collect your prize. The automated message says, "Congratulations on your lucky win. Please press 1 to collect your prize and 0 to decline it."

In this example $X = \{\text{collect}, \text{decline}\}$, $C = \{0, 1\} = \mathbb{F}_2$ and the encoding function e sends:

$$\text{decline} \to 0,$$
$$\text{collect} \to 1.$$

The decoding function d is the inverse of this.

Of course this is probably the worst encoding algorithm in the world (as is the same for any automated system). You would be devastated if you pressed 0 by accident, or even worse if you pressed 1 but there was a fault in the connection that transmitted 0.

How can we improve this system? Well one obvious idea is to repeat your choice, i.e. have $C = \{00, 11\} \subset \mathbb{F}_2^2$. In this case it would be obvious if you made a single error, the message received would be 10 or 01. These are **not** codewords, they are elements of $\mathbb{F}_2^2 \backslash C$.

Unfortunately the person at the other end **cannot** correct the error. If 10 is received then it could have come from either of the codewords 00 or 11 by making one error!

Ok so we are making headway. Let's do the obvious thing and choose $C = \{000, 111\} \subset \mathbb{F}_2^3$. It should be clear that we can now detect up to two errors. What's new is that we can now correct up to one error. How do we do this? We take our received message and count the number of 1's; if there is one then the intended message was 000 and if there are two then the intended message is 111.

It should now be clear that we can continue these constructions forever. We may make a code $\mathcal{R}_n = \{000...0, 111...1\} \subset \mathbb{F}_2^n$ for any n. Intuitively these codes can detect $n - 1$ errors and correct $\lfloor \frac{n-1}{2} \rfloor$ errors.

One might think that we are done since we have found codes that detect/correct any number of errors. However let's take a step back and think about this; in order to guarantee that we can correct t errors we are using messages of length $2t$ or $2t+1$. What's more, all of this is only to guarantee that we send **one** piece of information correctly! This is really expensive storage-wise (imagine having to send $1,000$ pieces of information). The aim of coding theory is to find better mathematical codes that make the balance of storage to error correction much more managable.

In this course we will focus mainly on the mathematics behind codes (and not the encoding algorithms). We will see that good choices of C arise from vector spaces and for these we will find interesting results that tell us how good error detection/correction is. Along the way we will see real world examples of codes, such as ISBN numbers, Hamming codes and BCH codes (used in CD/DVD design and barcodes).

2 Fields and Vector Spaces

As mentioned, many good codes come from vector spaces. In this section we will see a reminder of the basics of fields and vector spaces.

Intuitively speaking a field is a set of objects such that we can do the usual operations of **addition**, **subtraction**, **multiplication** and **division**. There are two equivalent definitions.

Definition 2.1. A **field** is a set F with two binary operations $+, \times$ such that $(F, +)$ and $(F\backslash\{0\}, \times)$ are abelian groups.

Alternatively a **field** is a commutative ring F such that the units of F are $F\backslash\{0\}$, i.e. everything non-zero is invertible.

There are many examples of fields.

Example 2.2. The sets $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ are fields. The set of integers modulo p is also a field, denoted $\mathbb{F}_p$.

The sets $\mathbb{Z}, \mathbb{N}$ are **not** fields. Neither are the sets of integers modulo n for composite n.

The fields $\mathbb{Q}$ and $\mathbb{F}_p$ are the simplest fields in some sense.

Lemma 2.3. *Let F be a field. Then F contains either $\mathbb{Q}$ or $\mathbb{F}_p$ (for a unique p) as a subfield.*

Exercise - Prove this (Hint: consider whether or not sums of the form $1 + 1 + ... + 1$ can vanish).

Not only can we study **subfields** but we can build **extension fields** out of old fields. We "invent" new elements and then throw in all sums, differences, products and inverses in order to guarantee we have a field.

Definition 2.4. Let X be a set and F be a field. The **extension generated by** X is the smallest field containing both F and X, denoted $F(X)$.

In this course we will only care about fields of the form $F(\alpha)$ where α is **algebraic** over F, i.e. satisfies a polynomial equation over F. It is pretty easy to describe such fields.

Example 2.5. If $F = \mathbb{R}$ and $X = \{i\}$ then $\mathbb{R}(i) = \{a + ib \,|\, a, b \in \mathbb{R}\} = \mathbb{C}$.

If $F = \mathbb{Q}$ and $X = \{\sqrt{2}\}$ then $\mathbb{Q}(\sqrt{2}) = \{a + b\sqrt{2} \,|\, a, b \in \mathbb{Q}\}$.

If $F = \mathbb{Q}$ and $X = \{\sqrt[3]{2}\}$ then $\mathbb{Q}(\sqrt[3]{2}) = \{a + b\sqrt[3]{2} + c\sqrt[3]{2}^2 \,|\, a, b, c \in \mathbb{Q}\}$.

Fact - In general if α is algebraic over F then $F(\alpha) = \{a_0 + a_1\alpha + a_2\alpha^2 + ... + a_{d-1}\alpha^{d-1} \mid a_i \in F\}$ where d is the degree of a **minimal polynomial** of α over F, i.e. a polynomial with coefficients in F of smallest degree with respect to having α as a root. A minimal polynomial of α can also be described as an irreducible polynomial having α as a root.

In this course we will mainly consider finite fields, i.e. fields with finitely many elements. We have already seen examples of these, $\mathbb{F}_p$ for various primes p. However there exist others.

Facts -

1. **Every** finite field is of the form $\mathbb{F}_p(\alpha)$ for some α that is algebraic over $\mathbb{F}_p$. From this it is clear that every finite field has size p^n for some prime p and $n \geq 1$ (the a_i in the above description lie in $\mathbb{F}_p$ and so there are p choices for each).
2. In fact there is exactly **one** finite field of size p^n for each p, n, up to isomorphism. This field is denoted $\mathbb{F}_{p^n}$. We construct it by finding an irreducible polynomial f of degree n over $\mathbb{F}_p$ and constructing the field $\mathbb{F}_p(\alpha)$, where α is defined to be a root of f.

Example 2.6. We will construct the field $\mathbb{F}_9$. Since $9 = 3^2$ we construct this field by first finding an irreducible quadratic over $\mathbb{F}_3$. The polynomial $x^2 + 1$ works (check this). Then $\mathbb{F}_9 = \mathbb{F}_3(\alpha) = \{a + b\alpha \mid a, b \in \mathbb{F}_3\}$, where α satisfies $\alpha^2 + 1 = 0$.

Exercise - Check that this set of 9 elements really is a field by constructing addition and multiplication tables.

Beware - We know that $\mathbb{F}_p$ is the set of integers mod p however $\mathbb{F}_{p^n}$ is **not** the same as the integers mod p^n.

The definition of field says that the set $F^\times = F\backslash\{0\}$ should be a group under multiplication. For finite fields a very special thing happens.

Fact - $\mathbb{F}_{p^n}^\times$ is a **cyclic** group, i.e. there exists $\beta \in \mathbb{F}_{p^n}^\times$ whose powers generate the whole group. The element β is called a **primitive root** of $\mathbb{F}_{p^n}^\times$.

Example 2.7. In $\mathbb{F}_9$, as constructed above, the element α is not a primitive root since

$$\begin{aligned}\alpha^2 &= 2\\ \alpha^3 &= 2\alpha\\ \alpha^4 &= 2\alpha^2 = 1.\end{aligned}$$

However $\beta = 1 + \alpha$ is a primitive root since

$$\begin{aligned}
\beta^2 &= 2\alpha \\
\beta^3 &= 1 + 2\alpha \\
\beta^4 &= 2 \\
\beta^5 &= 2 + 2\alpha \\
\beta^6 &= \alpha \\
\beta^7 &= 2 + \alpha \\
\beta^8 &= 1
\end{aligned}$$

We now move on to vector spaces. The notion of vector space is meant to capture the algebraic properties of the spaces $\mathbb{R}^n$ of real vectors. In particular we can **add** and **scalar multiply** vectors and these operations have lots of basic properties.

Definition 2.8. A **vector space** over a field F is a non-empty set V with two operations; **addition** and **scalar multiplication** by elements of F such that:

- $(V, +)$ is an abelian group.
- $\alpha(\beta\mathbf{v}) = (\alpha\beta)\mathbf{v}$ for all $\alpha, \beta \in F$ and $\mathbf{v} \in V$.
- $1\mathbf{v} = \mathbf{v}$ for all $\mathbf{v} \in V$.
- $\alpha(\mathbf{v} + \mathbf{w}) = \alpha\mathbf{v} + \alpha\mathbf{w}$ for all $\alpha \in F$ and $\mathbf{v}, \mathbf{w} \in V$.
- $(\alpha + \beta)\mathbf{v} = \alpha\mathbf{v} + \beta\mathbf{v}$ for all $\alpha, \beta \in F$ and $\mathbf{v} \in V$.

Example 2.9. Many sets of mathematical objects give vector spaces.

- Vectors: F^n is a vector space over F under the usual addition and scaling operations on vectors.
- Polynomials: $F[x]$ is a vector space over F under the usual addition and scaling operations on polynomials.
- Matrices: $M_{m,n}(F)$ is a vector space over F under the usual addition and scaling operations on matrices.
- Field extensions: If K is a field containing F then K is a vector space over F under the usual addition and multiplication operations in a field.

In $\mathbb{R}^n$ we are able to study all vectors by choosing a **coordinate system**. The one we usually use are **Cartesian coordinates**, which write vectors uniquely as $a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + ... + a_n\mathbf{e}_n$ with $a_i \in \mathbb{R}$ and $\mathbf{e}_i = (0, ..., 0, 1, 0, ..., 0)$. Of course this is a special choice of coordinate system, one could use almost any set of n vectors to create a coordinate system. However bad choices exist.

Example 2.10. In $\mathbb{R}^2$ every vector can be written as $a(1,0)+b(0,1)$ for unique $a, b \in \mathbb{R}$. One could use another coordinate system and write every vector uniquely as $c(1,1)+d(1,0)$. However other choices are not so good.

For example not every vector can be written as $a(1,0)$. Since $\mathbb{R}^2$ is a "2-dimensional" thing we shouldn't expect one vector to work. One vector should only generate a line, a "1-dimensional" thing.

How about $(1,0),(0,1),(1,1)$? Again this is a bad choice. Of course we **can** write every vector in the form $a(1,0)+b(0,1)+c(1,1)$ but not **uniquely**. We seem to have too many vectors so that there is a **redundancy** or **dependence**. The vector $(1,1)$ could easily be discarded because it is already made from $(1,0)$ and $(0,1)$ and so provides nothing new.

Naturally we can consider the same question for general vector spaces; does there exist $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$ such that each $\mathbf{v} \in V$ is written **uniquely** as $\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n$ for $a_i \in F$?

Based on the above discussion we make the following definitions.

Definition 2.11.

- We say that $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n \in V$ **span** V if every $\mathbf{v} \in V$ can be written as $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n$ for some $a_i \in F$ (not necessarily uniquely).
- We say that $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n \in V$ are **linearly independent** if the only solution to the equation $a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n = \mathbf{0}$ is $a_i = 0$ for all i.
- We say that $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n \in V$ is a **basis** for V if the vectors span and are linearly independent.

Intuitively the notion of spanning V means every vector can be generated from the given set of vectors. Linear independence gets rid of the redundancy mentioned above and is telling us that things that can be generated can be done so uniquely. Then a basis combines the two, allowing every vector to be generated uniquely.

Fact - If a vector space has a finite basis then all bases have the same size, the **dimension** of the vector space.

Example 2.12. If F is a field then F^n is an n-dimensional vector space, with basis given by $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n$, where $\mathbf{e}_i = (0, ..., 0, 1, 0, ..., 0)$ with 1 in the ith position. Of course there are many different choices of basis, one could instead use $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$ with $\mathbf{v}_i = (1, 1, ..., 1, 0, ..., 0)$ where the first i entries are 1.

The space $F[x]$ is infinite dimensional with basis $1, x, x^2, x^3, ...$ There cannot be a finite basis.

Exercise - Prove the claims about $F[x]$.

Beware - The field of scalars matters.

Example 2.13. As an $\mathbb{R}$-vector space $\mathbb{C}$ is 2-dimensional. A basis is given by $1, i$ since $\mathbb{C} = \{a+ib \,|\, a, b \in \mathbb{R}\}$. However as a $\mathbb{C}$-vector space $\mathbb{C}$ is 1-dimensional. A basis is given by 1 since $\mathbb{C} = \{z \cdot 1 \,|\, z \in \mathbb{C}\}$.

We can use matrices to generate vector spaces. Given a matrix $A \in M_{m,n}(F)$ one can associate to it **two** vector spaces:

- The space $\text{RowSpace}(A) \subseteq F^n$ is the span of the rows of A. The dimension of this space, $\text{rank}(A)$ is the **rank** of A.
- The space $\text{NullSpace}(A) \subseteq F^n$ is the set of vectors such that $A\mathbf{v}^T = \mathbf{0}$. The dimension of this space, $\text{nullity}(A)$ is called the **nullity** of A.

Beware - $\text{rank}(A)$ is **not** necessarily the number of rows of A. Some of the rows might be linearly dependent! However it is clear that $\text{rank}(A) \leq m$.

An important result in linear algebra connects the two constructions. We will not prove it in this course.

Theorem 2.14. *(Rank-Nullity Theorem) Let $A \in M_{m,n}(F)$. Then rank(A) + nullity$(A) = n$.*

Roughly speaking this theorem says that when solving k independent linear homogeneous equations in n variables you expect the space of solutions to be $(n-k)$-dimensional. For example a homogeneous linear equation in 3 variables should define a plane whereas 2 linear homogeneous equations should define a line if the equations are independent and a plane if they are dependent (i.e. the same equation twice).

3 Linear codes

We are now ready to start studying codes in detail. First we should make it clear what a code actually is.

Definition 3.1. Let F be a finite set (an "alphabet"). An (n, M)-**code** over F is a subset $C \subseteq F^n$ such that $|C| = M$.

So an (n, M)-code over F is intuitively a set of M "words", each of length n, with "letters" in F.

Example 3.2. For any alphabet F and $n \geq 1$ it is clear that F^n is an $(n, |F|^n)$-code. It is clearly the unique such code with these parameters.

Example 3.3. Roughly speaking, the English language is a code over the alphabet $F = \{A, B, C, D, ..., Z\}$. However the parameters are not well defined since the length of words is variable. You could turn it into a code under our definition by padding every word with extra A's so as to make every word the same length. The parameter M is constantly changing (new words are invented every day).

As a code the English language is poor, there are errors we **cannot** detect, hence the need to encode words mathematically.

Example 3.4. If $F = \{0, 1\}$ then the code $\mathcal{R}_n$ from the introduction is an $(n, 2)$-code. As discussed this is a better code since it detects/corrects errors but it is inefficient.

The definition of $\mathcal{R}_n$ can be extended to any alphabet F in the obvious way to give an $(n, |F|)$-code. These codes are the **repetition codes**.

Exercise - Make your own code. Is it any good?

Ok so now we know what a code is we should define what we mean by error detection and correction.

Definition 3.5. Let C be an (n, M)-code over an alphabet F.

- We say that C **detects** r **errors** if for each $\mathbf{c} \in C$ changing any r letters or less does not produce a codeword, i.e. if $\mathbf{v}$ comes from $\mathbf{c}$ by changing r letters or less then $\mathbf{v} \in F^n \backslash C$.
- We say that C **corrects** r **errors** if for each $\mathbf{v} \in F^n \backslash C$, there is at most one $\mathbf{c} \in C$ such that changing r or less letters of $\mathbf{c}$ gives $\mathbf{v}$.

Exercise - Convince yourself that this definition really matches our intuition. Why do we need the "at most" in the definition of error correction?

Example 3.6. The codes in Examples 3.2 and 3.3 are not r-error detecting/correcting for any r. However as seen in the introduction the repetition codes detect $n - 1$ errors and correct $\lfloor \frac{n-1}{2} \rfloor$ errors.

It is our main goal to study families of codes and their error detection/correction properties. Clearly as it stands we can't do much, a code is simply a finite set of words of a certain length. We desire codes with more mathematical structure.

It turns out some of the best codes are vector spaces over finite fields. In this course we will only work with codes over $\mathbb{F}_p$ but one can study them over other finite fields.

Definition 3.7. Let p be prime. An $[n,k]$-**linear code** over $\mathbb{F}_p$ is a code $C \subseteq \mathbb{F}_p^n$ that is a subspace of dimension k.

So intuitively a linear code is one such that we can **add** and **scalar multiply** codewords to get other codewords. This is clearly more algebraic structure than just being a set.

Example 3.8. The repetition codes $\mathcal{R}_n$ over $\mathbb{F}_p$ are (n,p)-codes but also $[n,1]$-linear codes. A basis is given by the codeword 111...1.

Example 3.9. The code $\mathcal{E}_3 = \{000, 110, 101, 011\}$ is a $(3,4)$-code over $\mathbb{F}_2$ but also a $[3,2]$-linear code. A basis is given by the codewords 110 and 101. For reasons that will become clear later, this code is called the **even weight code** of length 3 over $\mathbb{F}_2$.

Exercise - How many errors can $\mathcal{E}_3$ detect and correct?

For linear codes it is extremely easy to convert between (n,M) and $[n,k]$ notation.

Lemma 3.10. *Let C be an $[n,k]$-linear code over $\mathbb{F}_p$. Then $M = p^k$, so that C is an (n,p^k)-code over $\mathbb{F}_p$.*

Proof. By definition C is an $[n,k]$-linear code over $\mathbb{F}_p$. Choose a basis $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k$ for C. Then:

$$C = \{\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + ...\alpha_k\mathbf{v}_k \,|\, \alpha_i \in \mathbb{F}_p\}.$$

Clearly there are p^k choices for $(\alpha_1, \alpha_2, ..., \alpha_k)$ and each gives a different codeword by linear independence of the $\mathbf{v}_i$. Thus $|C| = M = p^k$. □

One advantage of using linear codes is that they are much easier to store, since they have a basis.

Definition 3.11. Let C be an $[n,k]$-linear code. Then a **generator matrix** for C is a matrix $G \in M_{k,n}(\mathbb{F}_p)$ such that the rows of G form a basis for C, i.e. G is of rank k and $C = \text{RowSpace}(G)$.

Example 3.12.

- $\mathcal{R}_n$ has a generator matrix: $G = (\ 1 \quad 1 \quad 1 \quad ... \quad 1\ )$.
- $\mathcal{E}_3$ has a generator matrix: $G = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$.

Of course, one can reverse the above construction and **create** linear codes from matrices. If one chooses any matrix $G \in M_{k,n}(\mathbb{F}_p)$ then we get a linear code by taking $C = \text{RowSpace}(G)$. Be careful though, the dimension of this code will be $\text{rank}(G) \leq k$.

Example - Convince yourself of this by making examples to show that the dimension can be k or strictly less than k.

Note that a code can have many generator matrices since there are many different choices of basis.

Exercise - How many generator matrices does an $[n, k]$-linear code over $\mathbb{F}_p$ have?

Generator matrices allow us to store codes efficiently. Rather than having a list of **all** codewords we simply need a list of the **basis** codewords. From this it is simple to generate the list of all codewords by matrix multiplication.

Lemma 3.13. *Let C be an $[n, k]$-linear code over $\mathbb{F}_p$ with generator matrix G. Then*

$$C = \{\boldsymbol{w}G \mid \boldsymbol{w} \in \mathbb{F}_p^k\}.$$

Exercise - Prove this.

Consider now the following question; given we have received a message $\mathbf{v}$, how do we check that $\mathbf{v} \in C$ for a given code C? Of course for arbitrary codes the only obvious way to do this is to simply check whether $\mathbf{v}$ appears in the list of codewords. This is obviously not very efficient in general.

For linear codes we have a much better method for answering this question. The idea is to realise C as the **null space** of a matrix. Then we can test whether $\mathbf{v} \in C$ or not by doing a simple matrix multiplication.

Definition 3.14. A **parity check matrix** for an $[n, k]$-linear code C over $\mathbb{F}_p$ is a matrix $H \in M_{n-k,n}(\mathbb{F}_p)$ such that $C = \text{NullSpace}(H)$, in other words

$$C = \{\mathbf{v} \in \mathbb{F}_p^n \mid H\mathbf{v}^T = \mathbf{0}\}.$$

Exercise - Why should H have $n - k$ rows? (Hint: Rank-Nullity).

Intuitively you should think of a parity check matrix as telling you a bunch of linear equations a vector must solve in order to be a codeword.

Example 3.15. The even weight code $\mathcal{E}_3$ mentioned earlier has a parity check matrix $H = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$, i.e. $\mathcal{E}_3$ is the set of vectors $v = (x_1, x_2, x_3) \in \mathbb{F}_2^3$ satisfying $x_1 + x_2 + x_3 = 0$.

We can clearly generalise this definition. Let $\mathcal{E}_n \subseteq \mathbb{F}_2^n$ be the linear code with parity check matrix $H = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \end{pmatrix}$. This family of codes are called the **even weight codes**.

Exercise - What are the parameters for the even weight code $\mathcal{E}_n$? How many words does it have?

It is not yet obvious that every linear code has a parity check matrix. We will prove this later. However note that, just as with generator matrices, parity check matrices are **not** unique.

It should be clear that, given a parity check matrix for C we can construct a generator matrix for C. Simply solve the equations!

Example 3.16. Let C be the linear code over $\mathbb{F}_3$ with parity check matrix:

$$H = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

Then $\mathbf{v} = (x_1, x_2, x_3)$ is a codeword if and only if:

$$\begin{aligned} x_1 + 2x_2 &= 0 \\ x_2 + x_3 &= 0 \end{aligned}$$

Solving these equations gives $x_1 = x_2$ and $x_3 = 2x_2 = 2x_1$, so the general solution is $\mathbf{v} = (x_1, x_1, 2x_1)$. Hence $C = \{(0,0,0), (1,1,2), (2,2,1)\}$ and so a generator matrix is given by $G = \begin{pmatrix} 1 & 1 & 2 \end{pmatrix}$.

One might imagine that the opposite calculation, constructing a parity check matrix from a generator matrix is equally as simple. However with our current knowledge it is not obvious how to do this (imagine being given lots of vectors and asking for the linear equations that they solve). We will see how to do this later.

We end this chapter with a well known example of a code used in the real world.

Example 3.17. The International Standard Book Number (ISBN) is a code used to catalogue books worldwide. It is a code of length 10 over $\mathbb{F}_{11}$ (where in practice the letter X is used to denote the number 10). The first nine digits of an ISBN codeword tell us information about the book, such as country of origin, publisher and title. The tenth digit is called the **check digit** and is there to serve as error detection.

The ISBN code is usually defined by the parity check matrix:

$$H_{\text{ISBN}} = \begin{pmatrix} 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}.$$

Let's show that this works. The book, "The Ultimate Justin Bieber Fan Book 2016" has ISBN 1530220513 and indeed we see that:

$$10(1)+9(5)+8(3)+7(0)+6(2)+5(2)+4(0)+3(5)+2(1)+1(3) = 121 \equiv 0 \bmod 11.$$

Exercise - What is the maximum number of errors that the ISBN code can detect/correct? Prove your assertions.

4 Hamming Distance and the Sphere packing bound

Now we are ready to examine the error detection/correction parameters for arbitrary codes. The idea is simple but clever; measure how much pairs of codewords **differ** and this should tell us how many errors are allowed to be made.

The quantitative notion of the difference between two words is the following:

Definition 4.1. Let $\mathbf{v},\mathbf{w} \in F^n$ be words. The **Hamming distance** between them is

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

Example 4.2. Let $F = \{0,1\}$ then $d(0101,1010) = 4$ and $d(1111,1110) = 1$.

The Hamming distance has all of the properties that a measure of distance is expected to have.

Lemma 4.3. *The Hamming distance is a* ***metric*** *on F^n, i.e.*

- $d(\boldsymbol{v},\boldsymbol{w}) \geq 0$, *with equality if and only if* $\boldsymbol{v} = \boldsymbol{w}$.
- $d(\boldsymbol{v},\boldsymbol{w}) = d(\boldsymbol{w},\boldsymbol{v})$.
- $d(\boldsymbol{v},\boldsymbol{w}) \leq d(\boldsymbol{v},\boldsymbol{z}) + d(\boldsymbol{z},\boldsymbol{w})$ *for any* $z \in F^n$.

Exercise - Prove these properties.

Since we now have a notion of distance on F^n we also have the notion of spheres centered on words.

Definition 4.4. Let $\mathbf{v} \in F^n$. The **Hamming sphere** centred on $\mathbf{v}$, radius $r \geq 0$ is

$$S(\mathbf{v},r) = \{\mathbf{w} \in F^n \,|\, d(\mathbf{v},\mathbf{w}) \leq r\}.$$

Since the Hamming distance is integer valued it is enough to take the radius r to be an integer too. In such a case there is a simple formula for the size of a Hamming sphere.

Lemma 4.5. *If* $r \in \mathbb{N}$ *and* $\boldsymbol{v} \in F^n$ *then*

$$|S(\boldsymbol{v},r)| = \sum_{m=0}^{r} \binom{n}{m} (|F|-1)^m.$$

Proof. For $0 \leq m \leq r$ let $S_m = \{\mathbf{w} \in F^n \,|\, d(\mathbf{v},\mathbf{w}) = m\}$. Then clearly $|S(\mathbf{v},r)| = |S_0| + |S_1| + |S_2| + ... + |S_r|$. It suffices to prove that $|S_m| = \binom{n}{m}(|F|-1)^m$.

How many ways can we change $\mathbf{v}$ in **exactly** m places?

- First we must choose m letters of $\mathbf{v}$ to change. This is done in $\binom{n}{m}$ ways.
- Secondly for each letter we would like to change, there are $|F|-1$ ways to change it. Thus there are $(|F|-1)^m$ ways to change a particular set of m letters.

Thus in total there are $\binom{n}{m}(|F|-1)^m$ unique ways to change $\mathbf{v}$ in exactly m places. So $|S_m| = \binom{n}{m}(|F|-1)^m$, as required. □

Exercise - When $r = 0$ what is $|S(\mathbf{v}, r)|$? Why is this obvious from the definition?

Now let C be an (n, M)-code over F. Then the Hamming distance makes sense on C. It is possible to give an alternative definition of error detection/correction on C that uses Hamming spheres.

Definition 4.6. Let C be as above.

- C **detects** r **errors** if $S(\mathbf{c}, r) \cap C = \{\mathbf{c}\}$ for each $\mathbf{c} \in C$.
- C **corrects** r **errors** if whenever $\mathbf{c}, \mathbf{c}' \in C$ are such that $S(\mathbf{c}, r) \cap S(\mathbf{c}', r) \neq \varnothing$, then $\mathbf{c} = \mathbf{c}'$.

Exercise - Check that this definition is equivalent to the one we made earlier.

We need to measure how **different** codewords are. It makes sense to define the following:

Definition 4.7. The **Hamming distance** (or **minimum distance**) of C is

$$d(C) = \min\{d(\mathbf{v}, \mathbf{w}) \,|\, \mathbf{v}, \mathbf{w} \in C, \mathbf{v} \neq \mathbf{w}\}.$$

When C is understood we will write $d = d(C)$.

The Hamming distance of a code tells us how much of a difference there is between **all** codewords. It should be no surprise that this number should tell us how many errors C can detect/correct.

Theorem 4.8. *Let C be an (n, M)-code over F with Hamming distance d. Then C detects $d-1$ errors and corrects $t = \lfloor \frac{d-1}{2} \rfloor$ errors.*

Proof. Suppose C does not detect $d-1$ errors. Then there exist distinct codewords $\mathbf{c}, \mathbf{c}' \in C$ such that $\mathbf{c}' \in S(\mathbf{c}, d-1) \cap C$. But then

$$d \leq d(\mathbf{c}, \mathbf{c}') \leq d-1,$$

giving a contradiction.

Suppose C does not correct t errors, then there exists distinct codewords $\mathbf{c}, \mathbf{c}'$ such that $S(\mathbf{c}, t) \cap S(\mathbf{c}', t) \neq \phi$. Choose $\mathbf{v} \in S(\mathbf{c}, t) \cap S(\mathbf{c}', t)$. Then

$$\begin{aligned} d \leq d(\mathbf{c}, \mathbf{c}') &\leq d(\mathbf{c}, \mathbf{v}) + d(\mathbf{v}, \mathbf{c}') \\ &= d(\mathbf{c}, \mathbf{v}) + d(\mathbf{c}', \mathbf{v}) \\ &\leq 2t \\ &\leq 2\left(\frac{d-1}{2}\right) \\ &= d - 1. \end{aligned}$$

This is a contradiction. □

Exercise - Convince yourself that the numbers in the Theorem are maximal.

Since d is important we add it to the list of parameters.

Definition 4.9. An (n, M, d)-**code** over F is an (n, M)-code over F with Hamming distance d. Similarly for linear codes.

Definition 4.10. The quantity $t = \lfloor \frac{d-1}{2} \rfloor$ is called the **error correcting index** of C.

In the proof of the above Theorem we observed that the spheres of radius t, centered on codewords, do **not** intersect. These spheres are all subsets of F^n and so we can immediately tell something about the maximum number of codewords for a code with error correcting index t.

Corollary 4.11. *(The Sphere Packing Bound) Let C be an (n, M, d)-code over F. Then*

$$M \sum_{m=0}^{t} \binom{n}{m} (|F| - 1)^m \leq |F|^n.$$

Proof. Let $C = \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_M\}$ and consider the spheres $S_i = S(\mathbf{c}_i, t)$ for $1 \leq i \leq M$. These spheres do not overlap as subsets of F^n, so that

$$|S_1| + |S_2| + ... + |S_M| \leq |F^n| = |F|^n.$$

But $|S_i| = \sum_{m=0}^{t} \binom{n}{m} (|F| - 1)^m$ and this is independent of M and so the result follows. □

One can view the LHS of this inequality as telling us how many of the words of F^n in total are "close" enough to codewords. Most codes are quite wasteful since this number can be much less than the RHS (the total number of words). However some codes are lucky enough to have all words sufficiently close to codewords.

Definition 4.12. A code is **perfect** if it achieves equality in the Sphere Packing Bound.

We will see examples of perfect codes later.

Now that we have seen the importance of d we might ask how we can **compute** it? Clearly for arbitrary codes we have no choice but to compare every distinct pair of codewords, giving $\binom{M}{2}$ calculations to do. This is extremely inefficient. Fortunately for linear codes there are quicker ways!

Definition 4.13. The **weight** of $\mathbf{v} \in \mathbb{F}_p^n$ is

$$\mathrm{wt}(\mathbf{v}) = |\{i \,|\, v_i \neq 0\}| = d(\mathbf{v}, \mathbf{0}).$$

Exercise - Now do you see why $\mathcal{E}_n$ is called the even weight code?

The following result allows us to work out d by doing $M-1$ calculations rather than $\binom{M}{2}$ calculations.

Proposition 4.14. *Let C be a linear code over $\mathbb{F}_p$. Then*

$$d = \min\{wt(\boldsymbol{c}) \,|\, \boldsymbol{c} \in C, \boldsymbol{c} \neq \boldsymbol{0}\}.$$

Proof. This is clear once we note that, for $\mathbf{v}, \mathbf{w} \in \mathbb{F}_p^n$

$$d(\mathbf{v}, \mathbf{w}) = d(\mathbf{v} - \mathbf{w}, \mathbf{0}) = \mathrm{wt}(\mathbf{v} - \mathbf{w}).$$

□

It should be noted that the above result will **only** work for linear codes. We needed vector space operations, this makes **no** sense for arbitrary codes!

Beware - It is **not** enough to work out the weights of basis vectors to get d, it could be that the basis vectors have high weight yet some sum of them has low weight.

There is in fact a second way to find the Hamming distance of a linear code, this time from a parity check matrix.

Proposition 4.15. *Let C be a linear code over $\mathbb{F}_p$ with parity check matrix H. Suppose H posesses a set of h linearly dependent columns but that all sets of $h-1$ columns are linearly independent. Then $d = h$.*

Proof. Consider h linearly dependent columns of H, labelled $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_h$. Then there exist $\alpha_i \in \mathbb{F}_p$, not all zero, such that $\alpha_1\mathbf{h}_1 + \alpha_2\mathbf{h}_2 + ... + \alpha_h\mathbf{h}_h = 0$. In fact each α_i must be non-zero since otherwise H posesses $h-1$ linearly dependent columns, contrary to our assumption.

Consider the weight h vector $\mathbf{v} \in \mathbb{F}_p^n$ with α_i in the position corresponding to $\mathbf{h}_i$ and 0 elsewhere. We have $H\mathbf{v}^T = \mathbf{0}$, implying $\mathbf{v} \in C$. Thus $d \leq h$.

To get the other inequality we note that, since d is the Hamming distance of C there must exist a codeword $\mathbf{c} \in C$ of weight d. Then $H\mathbf{c}^T = \mathbf{0}$, which shows that H has d linearly dependent columns, so that $d \geq h$. □

5 Dual Codes

So far we have seen two ways to build linear codes, as row spaces or as a null spaces of matrices. In some sense the two constructions are linked to each other. For example a generator matrix for one code can easily be a parity check for another, and visa versa.

Example 5.1. The repetition code $\mathcal{R}_3$ over $\mathbb{F}_2$ has generator matrix $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ and parity check matrix $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$.

The even weight code $\mathcal{E}_3$ over $\mathbb{F}_2$ has generator matrix $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ and generator matrix $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.

We will see in this section that this is no coincidence. Each linear code has a **dual code** where the roles of G and H are swapped. In order to capture this swapping of generator and parity check matrix we use **orthogonality**.

Definition 5.2. Let $C \subseteq \mathbb{F}_p^n$ be a linear code. Then the **dual code** is defined by

$$C^\perp = \{\mathbf{v} \in \mathbb{F}_p^n \mid \mathbf{c}\mathbf{v}^T = 0, \text{ for all } \mathbf{c} \in C\}.$$

Exercise - Is it obvious that $C^\perp$ is a linear code?

Exercise - Check by hand that $\mathcal{R}_3$ and $\mathcal{E}_3$ are dual using this definition.

We now show that this definition gives us what we want.

Theorem 5.3. *Let C be an $[n, k]$-linear code over $\mathbb{F}_p$.*

1. *Any generator matrix for C is a parity check matrix for $C^\perp$. In particular $C^\perp$ is an $[n, n-k]$-linear code over $\mathbb{F}_p$. .*
2. *Any parity check matrix for C is a generator matrix for $C^\perp$.*
3. *$(C^\perp)^\perp = C$.*

Proof. 1. Let G be a generator matrix for C. We show that $C^\perp = \text{NullSpace}(G)$, so that G is a parity check matrix for $C^\perp$.

To do this note that by definition $\mathbf{v} \in C^\perp$ if and only if $\mathbf{c}\mathbf{v}^T = 0$ for all $\mathbf{c} \in C$. But $C = \text{RowSpace}(G)$ and the rows of G are linearly independent hence this is equivalent to $\mathbf{c}\mathbf{v}^T = 0$ for each row $\mathbf{c}$ of G. Clearly this is equivalent to $G\mathbf{v}^T = \mathbf{0}$ which is equivalent to $C^\perp = \text{NullSpace}(G)$.

To get the parameters for $C^\perp$ note that $C^\perp$ is linear by the above exercise. It has length n so it remains to find $\dim(C^\perp)$. But G is a parity check matrix and so by the Rank-Nullity theorem $\dim(C^\perp) = \text{nullity}(G) = n - \text{rank}(G) = n - k$.

2. Let H be a parity check matrix for C. We show that $C^\perp = \text{RowSpace}(H)$, so that H is a generator matrix for $C^\perp$.

   To do this note that each $\mathbf{c} \in C$ satisfies $H\mathbf{c}^T = \mathbf{0}$ by definition. Thus, letting the rows of H be $\mathbf{v}_1, ..., \mathbf{v}_{n-k}$ we see that $\mathbf{v}_i\mathbf{c}^T = \mathbf{c}\mathbf{v}_i^T = 0$ for each i. So $\mathbf{v}_i \in C^\perp$ for each i, hence $\text{RowSpace}(H) \subseteq C^\perp$ since $C^\perp$ is a vector space. We must have equality since both spaces have dimension $n - k$.

3. Let $\mathbf{c} \in C$. Then by definition each $\mathbf{v} \in C^\perp$ satisfies $\mathbf{v}\mathbf{c}^T = 0$ and so $\mathbf{c} \in (C^\perp)^\perp$. Thus C is a subspace of $(C^\perp)^\perp$. However both have dimension k so we have equality.

□

Corollary 5.4. *Let C be as above.*

1. *Generator/parity check matrices for $C^\perp$ correspond to parity check/generator matrices for C respectively.*
2. *Every linear code has a parity check matrix.*

Proof. 1. This is clear since $C = (C^\perp)^\perp$.

2. By the first part any generator matrix for $C^\perp$ is a parity check matrix for C. Every linear code has a generator matrix and so we are done.

□

Not only have we proved the existence of parity check matrices but the above results tell us how to construct them.

Example 5.5. Let C be the $[4, 2]$-linear code over $\mathbb{F}_5$ with generator matrix

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

We wish to find a parity check matrix for C.

We know G is a parity check matrix for $C^\perp$ by the above theorem. We also know how to produce a generator matrix from a parity check matrix (Example 3.16). Doing this gives us the following generator matrix H for $C^\perp$:

$$H = \begin{pmatrix} 3 & 1 & 1 & 0 \\ 2 & 3 & 0 & 1 \end{pmatrix}.$$

By the Corollary this matrix is a parity check matrix for C.

Exercise - Check by hand that H is a parity check matrix for C.

6 Weight enumerators

We now have three ways of finding the Hamming distance of a linear code. We may compute it explicitly from the definition, or we may compute the minimum weight, or we may use a parity check matrix. However we haven't focused so much on the finer structure of a code. For example, given a code C how many codewords are there of weight m?

We create a bookkeeping device for these quantities. In this course we only consider this for binary codes (i.e. codes over $\mathbb{F}_2$) however the results can be extended.

Definition 6.1. Let C be an $[n, k, d]$-linear code over $\mathbb{F}_2$. It's **weight enumerator** is the polynomial

$$W_C(x, y) = \sum_{\mathbf{c} \in C} x^{n-\mathrm{wt}(c)} y^{\mathrm{wt}(c)} = \sum_{m=0}^{n} A_m x^{n-m} y^m.$$

Lemma 6.2. *The polynomial $W_C(x, y)$ has the following properties.*

1. $A_m = |\{\mathbf{c} \in C \mid wt(c) = m\}|$. *In particular* $W_C(1, 0) = A_0 = 1$.
2. $W_C(1, 1) = \sum_{m=0}^{n} A_m = |C|$.
3. $W_C(0, 1) = A_n = \begin{cases} 1 & \text{if } (1, 1, ..., 1) \in C \\ 0 & \text{otherwise} \end{cases}$
4. $d(C) = min(\{m \neq 0 \mid A_m \neq 0\})$.

Example 6.3.
1. The repetition code $C = \mathcal{R}_n$ over $\mathbb{F}_2$ has $W_C(x, y) = x^n + y^n$.
2. The even weight code $\mathcal{E}_3 = \{000, 110, 101, 011\}$ has $W_C(x, y) = x^3 + 3xy^2$.

One might hope that there is a link between the weight enumerators of a binary linear code and it's dual. We will prove the following well known identity.

Theorem 6.4. *(MacWilliams identity) Let C be a $[n, k]$-linear code over $\mathbb{F}_2$. Then:*

$$W_{C^\perp}(x, y) = \frac{1}{2^k} W_C(x + y, x - y).$$

Example 6.5. The binary codes $\mathcal{R}_3$ and $\mathcal{E}_3$ are dual. By the above we know that:

$$W_{\mathcal{R}_3}(x, y) = x^3 + y^3,$$
$$W_{\mathcal{E}_3}(x, y) = x^3 + 3xy^2.$$

Note that:

$$\frac{1}{2}((x + y)^3 + (x - y)^3) = \frac{1}{2}(2x^3 + 6xy^2) = x^3 + 3xy^2.$$

To prove the theorem we do a bit of finite Fourier analysis. We first make a discrete analogue of the Fourier transform.

Definition 6.6. Let $f : \mathbb{F}_2^n \to V$ be a function, where V is a vector space (over a field K with $\mathrm{char}(K) \neq 2$). Then the **Hadamard transform** of f is the function $\hat{f} : \mathbb{F}_2^n \to V$ given by:

$$\hat{f}(\mathbf{u}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} (-1)^{\mathbf{u}\cdot\mathbf{v}} f(\mathbf{v}).$$

The following result is often known as the Poisson summation formula for codes. There are analogues for all other types of Fourier transform.

Lemma 6.7. *Let C be a $[n, k]$-binary code. Then*

$$\sum_{\boldsymbol{c} \in C^\perp} f(\boldsymbol{c}) = \frac{1}{2^k} \sum_{\boldsymbol{c} \in C} \hat{f}(\boldsymbol{c}).$$

Proof. Clearly

$$\sum_{\mathbf{c} \in C} \hat{f}(\mathbf{c}) = \sum_{\mathbf{c} \in C} \sum_{\mathbf{v} \in \mathbb{F}_2^n} (-1)^{\mathbf{c}\cdot\mathbf{v}} f(\mathbf{v}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} f(\mathbf{v}) \left(\sum_{\mathbf{c} \in C} (-1)^{\mathbf{c}\cdot\mathbf{v}} \right).$$

We claim that the second sum is $|C| = 2^k$ if $\mathbf{v} \in C^\perp$ and 0 otherwise. Given the claim the right hand side is equal to $2^k \sum_{\mathbf{c} \in C^\perp} f(\mathbf{c})$ and so we are done.

To prove the claim let $S(\mathbf{v}) = \sum_{\mathbf{c} \in C} (-1)^{\mathbf{c}\cdot\mathbf{v}}$. If $\mathbf{v} \in C^\perp$ then $\mathbf{c} \cdot \mathbf{v} = 0$ for all $\mathbf{c} \in C$ and so $S(\mathbf{v}) = |C| = 2^k$.

If $\mathbf{v} \notin C^\perp$ then there exists $\mathbf{c}' \in C$ such that $\mathbf{c}' \cdot \mathbf{v} = 1$. It then follows that:

$$-S(\mathbf{v}) = (-1)^{\mathbf{c}'\cdot\mathbf{v}} S(\mathbf{v}) = \sum_{\mathbf{c} \in C} (-1)^{(\mathbf{c}'+\mathbf{c})\cdot\mathbf{v}} = S(\mathbf{v})$$

since $\mathbf{c}' + C = C$. Hence $S(\mathbf{v}) = 0$ as required. □

Proof. (of the theorem) Take $V = \mathbb{C}[x, y]$ and let $f(\mathbf{u}) = x^{n-\mathrm{wt}(\mathbf{u})} y^{\mathrm{wt}(\mathbf{u})}$. We show that $\hat{f}(\mathbf{u}) = (x + y)^{n-\mathrm{wt}(\mathbf{u})} (x - y)^{\mathrm{wt}(\mathbf{u})}$, then the lemma gives the result after summing.

To prove the claim, first note that

$$\hat{f}(\mathbf{u}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} (-1)^{\mathbf{u}\cdot\mathbf{v}} x^{n-\mathrm{wt}(\mathbf{v})} y^{\mathrm{wt}(\mathbf{v})}.$$

Note that since C is a binary code we have that $\mathrm{wt}(\mathbf{v}) = \sum_{i=1}^n v_i$ and so $n - \mathrm{wt}(\mathbf{v}) = \sum_{i=1}^n (1 - v_i)$.

Thus:

$$\hat{f}(\mathbf{u}) = \sum_{\mathbf{v}\in\mathbb{F}_2^n} (-1)^{u_1v_1+\ldots+u_nv_n} \prod_{i=1}^{n} x^{1-v_i}y^{v_i} = \sum_{\mathbf{v}\in\mathbb{F}_2^n} \prod_{i=1}^{n} (-1)^{u_iv_i} x^{1-v_i}y^{v_i}$$
$$= \prod_{i=1}^{n} \sum_{w=0}^{1} (-1)^{u_iw} x^{1-w}y^{w}$$

If $u_i = 0$ then the sum is $x + y$, if $u_i = 1$ then it is $x - y$. Thus the product equals $(x + y)^{n-\mathrm{wt}(\mathbf{u})}(x - y)^{\mathrm{wt}(\mathbf{u})}$ as required. □

It should be noted that there are analogues of the MacWilliams identity for non-binary linear codes, and even for non-linear codes.

7 Syndrome decoding, Hamming Codes and beyond

So far we know how to find generator/parity check matrices for linear codes and from this information we can compute the error correction index t. However we have yet to discover exactly **how** to correct errors!

In order to **detect** errors we use the parity check matrix. It turns out that we can also use it to **correct** errors.

Definition 7.1. Let C be a $[n, k]$-linear code over $\mathbb{F}_p$ with parity check matrix H. The **syndrome** of $\mathbf{v} \in \mathbb{F}_p^n$ is the vector $H\mathbf{v}^T \in \mathbb{F}_p^{n-k}$.

Of course the syndrome of $\mathbf{c} \in C$ is always $\mathbf{0}$ by definition of H but other words will have non-zero syndrome. However vectors may have the same syndrome.

Lemma 7.2. *$\boldsymbol{v}, \boldsymbol{w}$ have the same syndrome if and only if $\boldsymbol{v} - \boldsymbol{w} \in C$.*

Proof. This is straight-forward since $H\mathbf{v}^T = H\mathbf{w}^T$ if and only if $H(\mathbf{v}-\mathbf{w})^T = \mathbf{0}$, which occurs if and only if $\mathbf{v} - \mathbf{w} \in C$. □

The above lemma tells us that the syndrome of a vector $\mathbf{v}$ is uniquely determined by its **coset**

$$\mathbf{v} + C = \{\mathbf{v} + \mathbf{c} \,|\, \mathbf{c} \in C\}.$$

The fact that allows us to correct errors is the following.

Proposition 7.3. *If C has error correcting index t then the syndromes of vectors of weight t or less are distinct.*

Proof. Suppose $\mathbf{v}, \mathbf{v}'$ are vectors of weight t or less and that both have the same syndrome. Then $\mathbf{v} - \mathbf{v}' = \mathbf{c}$ for some codeword $\mathbf{c} \in C$. But then

$$\begin{aligned} d \leq \mathrm{wt}(\mathbf{c}) = \mathrm{wt}(\mathbf{v} - \mathbf{v}') &= d(\mathbf{v}, \mathbf{v}') \\ &\leq d(\mathbf{v}, \mathbf{0}) + d(\mathbf{v}', \mathbf{0}) \\ &\leq 2t \\ &\leq d - 1, \end{aligned}$$

giving a contradiction. □

The error vectors of weight $t+1$ or less will **not** have distinct syndromes (hence why we can't correct that number of errors). However the above result lets us create an algorithm for correcting t or less errors:

1. Find a complete set of **coset leaders**, i.e. minimal weight representatives $\mathbf{v}_i$ for the cosets of C.

2. Find the syndromes of the $\mathbf{v}_i$ and store these in a table, whose rows are indexed by syndrome.

3. Given a message $\mathbf{v}$ which has at most t errors we correct these errors as follows. Work out the syndrome $\mathbf{w} = H\mathbf{v}^T$. In the row corresponding to $\mathbf{w}$ find $\mathbf{v}_i$ of weight t or less. This is **unique** by the above results.

4. It follows that $\mathbf{v} = \mathbf{c} + \mathbf{v}_i$ for some **unique** codeword $\mathbf{c}$. Then $\mathbf{c} = \mathbf{v} - \mathbf{v}_i$ is the intended message.

Example 7.4. Correcting one error is easy (assuming your code allows it). Here $\mathbf{v} = \mathbf{c} + \alpha\mathbf{e}_i$ for a **unique** $\mathbf{c} \in C$, with $\alpha \in \mathbb{F}_p$ and $\mathbf{e}_i = (0...010...0)$ (i.e. a weight 1 coset leader).

In this case the syndrome of $\mathbf{v}$ will be $\mathbf{w} = H(\alpha\mathbf{e}_i)^T = \alpha\mathbf{h}_i$, where $\mathbf{h}_i$ is the ith column of H. Since $t = 1$ we know that the columns of H are pairwise independent (by Proposition 4.15 and the fact that $d \geq 3$) and so there is no confusion which column the syndrome is a multiple of (and what the multiple is).

For example, let C be the linear code over $\mathbb{F}_3$ with parity check matrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 2 & 1 \end{pmatrix}.$$

Then $t = 1$ and so the above argument will work.

The vector $\mathbf{c} = (1, 1, 1, 0)$ is a codeword (check this). Let's make one error, changing the 4th position to give $\mathbf{v} = (1, 1, 1, 2)$.

The syndrome is $\mathbf{w} = H\mathbf{v}^T = (2, 0, 2)^T$. This is visibly $2\mathbf{h}_4$ and so $\mathbf{c} = \mathbf{v} - 2\mathbf{e}_4 = (1, 1, 1, 0)$, as expected.

Example 7.5. Let's see an example where we can correct more than one error. Consider the $[5, 1, 5]$-code over $\mathbb{F}_2$ with parity check matrix

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

This code is actually $\mathcal{R}_5 = \{(0, 0, 0, 0, 0), (1, 1, 1, 1, 1)\}$, the repetition code of length 5 (Why?).

There are 16 cosets of C in $\mathbb{F}_2^5$ given by coset leaders $\{\mathbf{0}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4, \mathbf{e}_5, \mathbf{e}_1 + \mathbf{e}_2, \mathbf{e}_1 + \mathbf{e}_3, \mathbf{e}_1 + \mathbf{e}_4, \mathbf{e}_1 + \mathbf{e}_5, \mathbf{e}_2 + \mathbf{e}_3, \mathbf{e}_2 + \mathbf{e}_4, \mathbf{e}_2 + \mathbf{e}_5, \mathbf{e}_3 + \mathbf{e}_4, \mathbf{e}_3 + \mathbf{e}_5, \mathbf{e}_4 + \mathbf{e}_5\}$.

The syndrome table is as follows:

$\mathbf{w}$	Coset Leaders
0000	$\mathbf{0}$
1000	$\mathbf{e}_1$
0100	$\mathbf{e}_2$
0010	$\mathbf{e}_3$
0001	$\mathbf{e}_4$
1100	$\mathbf{e}_1 + \mathbf{e}_2$
1010	$\mathbf{e}_1 + \mathbf{e}_3$
1001	$\mathbf{e}_1 + \mathbf{e}_4$
0110	$\mathbf{e}_2 + \mathbf{e}_3$
0101	$\mathbf{e}_2 + \mathbf{e}_4$
0011	$\mathbf{e}_3 + \mathbf{e}_4$
1110	$\mathbf{e}_4 + \mathbf{e}_5$
1101	$\mathbf{e}_3 + \mathbf{e}_5$
1011	$\mathbf{e}_2 + \mathbf{e}_5$
0111	$\mathbf{e}_1 + \mathbf{e}_5$
1111	$\mathbf{e}_5$

Notice how each vector of weight 2 or less is in a unique row, as expected. Let's demonstrate how easy it is to correct 2 or less errors.

Suppose we receive the vectors $(0,0,0,0,0), (1,0,1,1,1), (0,1,0,0,1)$. The corresponding syndromes are $0000, 0100$ and 1011. Under the assumption that 2 or less errors have been made in each vector we conclude (from the table) that the first vector is a codeword, the second has one error in the 2nd bit and the third has two errors in bits 2 and 5. We may then correct to give the intended message $(0,0,0,0,0), (1,1,1,1,1), (0,0,0,0,0)$.

We **cannot** correct any more than 2 errors. Let's demonstrate this, assuming that 3 or less errors are made. Suppose the intended message is $(1,1,1,1,1)$ but instead we receive $(1,0,0,0,1)$. Then the syndrome is 0111 and so the table tells us that bits 1 and 5 are incorrect. But then we would translate the message to $(0,0,0,0,0)$, which is wrong.

The problem is that both $\mathbf{e}_1 + \mathbf{e}_5$ and $\mathbf{e}_2 + \mathbf{e}_3 + \mathbf{e}_4$ are in the same coset and so the corresponding errors are indistinguishable by taking syndromes.

Hamming codes

Let $r \geq 2$. The **Hamming code** Ham_r over $\mathbb{F}_2$ has parity check matrix

$$H_r = \begin{pmatrix} 0 & 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & 0 & \ldots & 1 & 1 \\ . & . & . & \ldots & . & . \\ . & . & . & \ldots & . & . \\ . & . & . & \ldots & . & . \\ 0 & 1 & 1 & \ldots & 1 & 1 \\ 1 & 0 & 1 & \ldots & 0 & 1 \end{pmatrix},$$

i.e. for $1 \leq i \leq 2^r - 1$ the ith column of H_r is the number i in binary.

Lemma 7.6. *Ham_r is a $[2^r-1, 2^r-r-1, 3]$-linear code over $\mathbb{F}_2$.*

Proof. It is clear that H_r has rank r since columns $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_4, ..., \mathbf{h}_{2^{r-1}}$ are linearly independent. Thus by the Rank-Nullity theorem $\dim(\mathrm{Ham}_r) = (2^r-1) - r = 2^r - r - 1$.

To prove that $d = 3$ we note that $\mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3 = \mathbf{0}$, so that $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ are linearly dependent. Also no two columns of H_r are linearly dependent (since we are working over $\mathbb{F}_2$ this would force two columns to be equal, which is not the case by definition). By Proposition 4.15 this tells us that $d = 3$. □

Recall that earlier we defined a code to be **perfect** if it gives equality in the Sphere Packing Bound. Hamming codes are an infinite family of such codes.

Theorem 7.7. *The codes Ham_r are perfect for all r.*

Proof. We have $|F| = 2, t = 1$ and $M = 2^{2^r-r-1}$. The LHS of the Sphere Packing Bound is

$$2^{2^r-r-1}\left(\binom{2^r-1}{0}(2-1)^0 + \binom{2^r-1}{1}(2-1)^1\right) = 2^{2^r-r-1}(1+(2^r-1)) = 2^{2^r-1},$$

which is the RHS of the Sphere Packing Bound, as required. □

We have seen that $d = 3$ and $t = 1$ for all Hamming codes. We can use the method of Example 7.4 to correct a single error. However things are much simpler.

Proposition 7.8. *Suppose $\boldsymbol{v}$ comes from a codeword of Ham_r by changing the ith bit. Then the syndrome of $\boldsymbol{v}$ gives the number i in binary.*

Proof. This is simple since the syndrome of $\mathbf{v}$ is $\mathbf{h}_i$, the ith column of H (we are working over $\mathbb{F}_2$ so that $\alpha = 1$). But $\mathbf{h}_i$ is the number i in binary, by definition of H_r. □

Example 7.9. Consider Ham_3. Suppose we receive $\mathbf{v} = (1,1,1,1,0,1,1)$. Then the syndrome is $H_3\mathbf{v}^T = (1,0,1)^T \neq \mathbf{0}$ so an error has been made.

Note that writing 5 in binary gives 101 and so, under the assumption that **exactly** one error has been made, the 5th bit is wrong and the intended codeword would be $(1,1,1,1,1,1,1)$.

One can make non-binary Hamming codes but we will not see them in this course.

2-error correcting BCH codes

The Hamming codes are interesting codes. They are perfect codes, they are very easy to generate and to use. Unfortunately they only correct one error, but there is a natural way to generalise the Hamming codes to a more general class of codes called BCH codes. These can be created to correct at least t errors for any t. We will finish the course by looking at how this is done for $t = 2$.

Recall that the columns of H_r are the numbers $1 \leq i \leq 2^r - 1$ written in binary. One can think of these as the non-zero vectors in $\mathbb{F}_2^r$. But the finite field $\mathbb{F}_{2^r}$ also has the structure of an r-dimensional vector space over $\mathbb{F}_2$. So alternatively we can think of the columns of H_r as the elements of $\mathbb{F}_{2^r}^\times$, written in terms of a basis.

Example 7.10. The polynomial $x^3 + x + 1$ is irreducible over $\mathbb{F}_2$ and so $\mathbb{F}_8 = \mathbb{F}_2(\alpha) = \{a + b\alpha + c\alpha^2 \mid a, b, c \in \mathbb{F}_2\}$ with $\alpha^3 + \alpha + 1 = 0$. It is easily checked that $\alpha = x$ is a primitive root for $\mathbb{F}_8^\times$.

Under the basis $1, \alpha, \alpha^2$ we find that the powers of α are

$$\begin{aligned}
\alpha &= (0,1,0)\\
\alpha^2 &= (0,0,1)\\
\alpha^3 &= (1,1,0)\\
\alpha^4 &= (0,1,1)\\
\alpha^5 &= (1,1,1)\\
\alpha^6 &= (1,0,1)\\
\alpha^7 &= (1,0,0)
\end{aligned}$$

The matrix

$$H = \begin{pmatrix} \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix},$$

agrees with H_3 up to reordering the columns. Thus their null spaces agree, so that the Hamming code Ham_3 is realised as having H as parity check matrix.

In general we can realise Ham_r as having parity check matrix

$$H_r = \begin{pmatrix} \alpha & \alpha^2 & \ldots & \alpha^{2^r-1} \end{pmatrix},$$

where α is a primitive root for $\mathbb{F}_{2^r}^\times$. To get the matrix in terms of 0's and 1's we simply expand using a basis.

We can also explain very simply using this parity check matrix why syndrome decoding works for one error. The possible syndromes of weight 1 coset leaders are simply powers of α, which are all distinct (since α is a primitive root). Further an error in the ith bit gives us syndrome α^i.

Example 7.11. In Ham_3 we have the following codeword $\mathbf{c} = (1,1,1,1,1,1,1)$. Let us change the 4th bit to get $\mathbf{v} = (1,1,1,0,1,1,1)$. Then the syndrome can be written as

$$H\mathbf{v}^T = \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7 = \alpha^4,$$

confirming that the 4th bit is indeed wrong.

Clearly Ham_3 cannot correct two errors since we can have different pairs of (i,j) values such that $\alpha^i + \alpha^j = \alpha^k$, for example $\alpha + \alpha^2 = \alpha^3 + \alpha^6 = \alpha^4$.

How could we modify this to correct **two** errors? The answer is quite simple; we add another row to H_r.

We consider a parity check matrix of the form

$$J_r = \begin{pmatrix} \alpha & \alpha^2 & \dots & \alpha^{2^r-1} \\ f(\alpha) & f(\alpha^2) & \dots & f(\alpha^{2^r-1}) \end{pmatrix},$$

where $f : \mathbb{F}_{2^r}^\times \to \mathbb{F}_{2^r}^\times$.

What is a **good** choice for f? Well we want to be able to correct **two** errors, in other words if errors are made in the ith and the jth places then we ought to be able to find i, j from the syndrome.

To this end let $\mathbf{v} = \mathbf{c} + \mathbf{e}_i + \mathbf{e}_j$ for distinct i, j. Then the syndrome is

$$\mathbf{0} \neq \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = H\mathbf{v}^T = H(\mathbf{e}_i + \mathbf{e}_j)^T = h_i + h_j = \begin{pmatrix} \alpha^i + \alpha^j \\ f(\alpha^i) + f(\alpha^j) \end{pmatrix}.$$

So we should choose f such that we can **uniquely** solve the equations

$$\begin{aligned} \alpha^i + \alpha^j &= w_1, \\ f(\alpha^i) + f(\alpha^j) &= w_2. \end{aligned}$$

(Under the assumption that two distinct solutions exist).

Of course as with life we come across many **bad** choices first.

Example 7.12. Clearly taking $f(x) = x$, i.e. the identity function, is a bad choice. If $w_1 \neq w_2$ then there would be **no** solution and if $w_1 = w_2$ then the two equations are identical. We have already seen that $\alpha^i + \alpha^j = w$ does not have a unique solution.

Exercise - Extend this argument to show that the functions $f(x) = cx$ are no good for $c \in \mathbb{F}_{2^r}$.

Example 7.13. Let us try higher powers. The function $f(x) = x^2$ is also a bad choice since $\alpha^{2i} + \alpha^{2j} = (\alpha^i + \alpha^j)^2$ and so there is no solution unless $w_1^2 = w_2$, in which case we again are left with the equation $\alpha^i + \alpha^j = w$.

However a **good** choice is given by the third power.

Example 7.14. Let $f(x) = x^3$. If the system

$$\begin{aligned} \alpha^i + \alpha^j &= w_1 \\ \alpha^{3i} + \alpha^{3j} &= w_2 \end{aligned}$$

has a solution then it is unique.

Why is this? Using the factorisation $x^3+y^3 = (x+y)(x^2-xy+y^2)$ we see that the second equation is equivalent to $w_1(w_1^2 - \alpha^{i+j}) = w_2$.

We can't have $w_1 = 0$ since then $w_2 = 0$ and so $\mathbf{w} = \mathbf{0}$. So we can rearrange to get $\alpha^{i+j} = w_1^2 - \frac{w_2}{w_1}$.

Now recall that if one knows $A = s + t$ and $B = st$ for numbers s, t then s and t are roots of the quadratic equation $x^2 - Ax + B$ (check this). Thus α^i, α^j are solutions to the quadratic equation

$$x^2 - w_1 x + \left(w_1^2 - \frac{w_2}{w_1} \right) = 0.$$

This equation has at most 2 roots and so i, j are uniquely determined.

Of course we can also correct exactly one error. If $\mathbf{v} = \mathbf{c} + \mathbf{e}_i$ then the syndrome is given by $w_1 = \alpha^i$ and $w_2 = \alpha^{3i}$ and so $w_2 = w_1^3$ and the error occurred in position i.

Putting all of this together we see that the linear code with parity check matrix J_r and $f(x) = x^3$ can correct up to 2 errors and the algorithm for doing so is as follows:

1. Receive $\mathbf{v}$ and calculate its syndrome $\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$.
2. If $w_1 = w_2 = 0$ then no error has occurred.
3. If $w_1 \neq 0$ and $w_2 = w_1^3$ then one error occurred in position i, corresponding to $w_1 = \alpha^i$.
4. If $w_1 \neq 0$ and $w_2 \neq w_1^3$ then two errors occurred and their positions are given by i, j where α^i, α^j are solutions to the quadratic

$$x^2 - w_1 x + \left(w_1^2 - \frac{w_2}{w_1} \right) = 0.$$

Example 7.15. Let's demonstrate the above for $r = 3$. In this case

$$J_3 = \begin{pmatrix} \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 \\ \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha & \alpha^4 & \alpha^7 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Let $\mathbf{v}_1 = (1,1,1,1,1,1,1)$, $\mathbf{v}_2 = (1,1,0,1,1,1,1)$ and $\mathbf{v}_3 = (1,0,1,1,1,0,1)$. The corresponding syndromes are

$$\mathbf{w}_1 = \begin{pmatrix} \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7 \\ \alpha^3 + \alpha^6 + \alpha^2 + \alpha^5 + \alpha + \alpha^4 + \alpha^7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
$$\mathbf{w}_2 = \begin{pmatrix} \alpha + \alpha^2 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7 \\ \alpha^3 + \alpha^6 + \alpha^5 + \alpha + \alpha^4 + \alpha^7 \end{pmatrix} = \begin{pmatrix} \alpha^3 \\ \alpha^2 \end{pmatrix}$$
$$\mathbf{w}_3 = \begin{pmatrix} \alpha + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^7 \\ \alpha^3 + \alpha^2 + \alpha^5 + \alpha + \alpha^7 \end{pmatrix} = \begin{pmatrix} \alpha^7 \\ \alpha^3 \end{pmatrix}$$

In the first case we have $w_1 = w_2 = 0$ so that $\mathbf{v}_1$ is a codeword, as expected. In the second case $w_1 \neq 0$ and $w_1^3 = \alpha^9 = \alpha^2 = w_2$, so that exactly one error has occured in position 3, as expected (since $w_1 = \alpha^3$).

In the third case we have $w_1 \neq 0$ and $w_1^3 \neq w_2$ so that exactly two errors have occurred. The corresponding quadratic is

$$x^2 - x + \alpha = 0.$$

One easily checks that the roots of this quadratic are α^2 and α^6 and so the errors occurred in positions 2 and 6, as expected.

The above code is an example of a **BCH code**. One can generalise the above ideas to design such codes that can correct any desired number of errors. For these reasons these codes are highly applicable in the real world. For example a special family of BCH codes called **Reed-Solomon codes** are used to encode data on CD's, DVD's and in barcodes.

Exercise - What are the parameters of the above code? (Hint: Can it correct 3 or more errors?)